zenon driver manual
DNP3_NG

v.8.20

COPADATA

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 customer service team, which 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 DNP3_NG

Driver for the protocol in accordance with IEEE1815 Distributed Network Protocol (DNP3).

The driver is the master at protocol level and supports serial communication with several outstations, as well as IP communication via TCP.

DEFINITION OF TERMS

In order for you to understand this document better, please find the definition of important terms in the following list.

Term	Definition
Event Class	A type of grouping in the outstation. There are **Event Classes** *1*,*2*, *3* and the **Static Class** *0*. An **Event Class** is assigned as a rule a buffer, in which value changes are stored for configured object groups with the variation configured in the **Outstation** . A master can request all value changes of this **Event Class** of the **Outstation** by means of a harvest inquiry for for example **Class** *1*. The **Static Class** *0* has an own task. It redelivers the last value in a harvest inquiry of the **master** respectively for nearly all **Points** . In the **Outstation** can be configured as a rule, which **Points** belong and/or which **Object Group** to which **Event Class** . There is no exact definition, however **Class** *1* is generally used for important messages, such as *binary inputs* for alarms.
Event poll	Read query from the **master** to the **outstation**, whereby the **master** requests the **outstation** to only send the amended values for a certain **event class** or for all **event classes** (*1*, *2* and *3*).
Integrity Poll	Read query from the **master** to the **outstation** for the **static class** *0*. The **Master** requests an initial display of all **points** with this query.
Master	Controlling station. A **master** sends read queries and control queries to the **outstation**.
Objekt Group	Data type or data type in the **outstation** with a defined functionality. **Object Group** *30* stands for a static analog input, for example. A value change for an analog input is assigned to **object group** *32*. **Object group** *40* serves, for example, for the reading of an analog output, **object group** *41* for writing to an analog output.
Outstation	SPS or RTU in DNP3. An outstation is a slave at protocol level and sends data to the **master** on request.
Point	Equivalent of a variable in the Outstation. A **Point** is addressed with one **Point Number** (Offset) per **Object Group** with e.g. **Point** *12* for **Object group** *30* and **Point** *12* for **Object Group** *32* both have the same analog input as a basis, however **point** *12* for **object group** *1* is a completely independent binary input. The general term for a value, time or status change is **DNP object**.
Unsolicited Response	Message from value changes of an **event class** that is spontaneously sent from the **outstation** to the **master**. to do this, the **master** must however first activate **unsolicited responses** in the **outstation**. The **outstation** must support **unsolicited responses** and these must be configured for it.

Term	Definition
Variation	Format, in which the **outstation** saves a static value or a value change in **class** *0* or **class** *1*, *2* or *3*. It can be configured in the **outstation** for each **object group** or for each **point**. The **variation** determines if a static value (**class** *0*) or a value change is an integer or a floating point. Or whether a time stamp is saved or not, or whether object flags are saved or not.

Find out more information in the chapter **DNP3/IEEE1815-2012 standard**. You can acquire this documentation of the IEEE. You get also access if you join the **DNP3 user group** : http://www.dnp.org/ (http://www.dnp.org/).

DEVICE PROFILE

Device profile is a standard document that describes, which functionality is supported with the DNP3 standard. For the DNP3_NG driver, you find the XML Device profile after zenon installation in the folder C:\ProgramData\COPA-DATA*zenonxxx\CommunicationProfiles\Dnp3\Driver\DNP3_NG.xml* (xxx corresponds the current zenon version number).

INFORMATION ON THE DNP332, DNP3_NG AND DNP_NG DRIVER

The DNP332 driver and the DNP3_NG will be replaced by the DNP3_TG from version 7.20. For reasons of compatibility, the DNP3_NG and the older DNP332 drivers are still included in the setup, but are no longer displayed in the driver selection list by default. Existing projects that are converted use the driver that was originally configured as before. The old driver can continue to be used normally in converted projects. However a switch to the new DNP3_TG driver is also possible.

DISPLAYING THE DNP332/DNP3_NG IN THE DRIVER LIST

If you want to use the DNP332/DNP3_NG driver in a new project in 7.20, the driver must be added to the driver list again:

1. Start the program **Driverinfo.exe** from the zenon installation medium; subfolder *\AdditionalSoftware\COPA-DATA DriverXML Editor*.

2. Open the driver XML file using the program called **Driverinfo**.
   Example: **TREIBER_DE.XML** from the folder *%CD_PROGRAMDATA7200%*
   In doing so, **DE** is the code for the language in the Editor and **7200** for the installed version, version 7.20.

3. Go to the *DNP3* folder and select **New Driver** in the context menu.

4. Enter **DNP332** or **DNP3_NG** in all three fields of the dialog and confirm by clicking on **OK**.

5. Save the changes by clicking on the **Save** symbol in the toolbar.
   The driver can now be selected again in the Editor.

6. Repeat this step for each language that you use in the Editor.

SWITCH TO THE NEW DNP3_TG DRIVER

The DNP3_TG driver is compatible with the old DNP3_NG driver and DNP332 driver in principle. The **Replace driver** function in the Editor can also be used to switch from the DNP332 driver or from the DNP3_NG driver to the DNP3_NG driver. After the driver replacement, some settings need to be made again manually. Some functions in the DNP3_TG driver are implemented differently and require a change to the project configuration.

When planning to replace a driver, please note the following:

General:

▸ Back up your project first

▸ Note the driver configuration of the old driver. These must be entered again once the driver has been replaced.

▸ Writing to a Frozen Counter variable no longer leads to an Immediate Freeze No Ack. This function can be achieved using a Commando variable.

DNP332 to DNP3_TG:

▸ If you use Select Before Operate with the DNP332 driver for the Command Processing, deactivate the **Select Before Operate** property for the variable. Instead, select the *Auto-SBO* entry for *Binary Output* and *Analog Output* variables in the **Command Mode** property.

▸ If, you have used driver data types for **Analog Inputs** or **Counter** with DNP332 and these no longer exist in the DNP3_NG driver, then you must amend the data types of the variables before the change. Background: With the DNP332 driver, it was possible to select data types that were not envisaged by the DNP3 standard. These data types can no longer be used with DNP3_NG.

DNP3_NG to DNP3_TG:

▸ The **Command Mode** property for *Analog Output* and *Binary Output* variables must be set manually. Alternatively, it is possible to export the variables in XML format before the driver switch and to import them again after the switch. The DNP3_TG driver also supports, in addition to Direct Operate and Auto SBO , Direct Operate No Ack.

▸ The property variation for the variable has been removed. Instead of configuring the variation for the variable, the variable can be excluded from the class poll with the **Classless Read** check box and explicitly read with the desired version with a user-defined command.

▸ The "class scan" variable scan no longer exists for the DNP3_TG. It is best to delete variables of this type before the driver switch. To trigger a class poll, explicit reading or also a Cold Restart, create Command variables from the DNP3_TG driver variables.

▸

3 Driver history

Date	Driver version	Change
11/9/2012	3754	Created driver documentation

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

4 DNP3_NG - data sheet

General:	
Driver file name	DNP3_NG.exe
Driver name	DNP3 Driver New Generation
PLC types	DNP3 / IEEE 1815 Outstations
PLC manufacturer	DNP3; GE Harris

Driver supports:	
Protocol	TCP/IP; DNP3; IEEE Std 1815

Driver supports:	
Addressing: Address-based	Address based
Addressing: Name-based	--
Spontaneous communication	X
Polling communication	X
Online browsing	X
Offline browsing	X
Real-time capable	X
Blockwrite	--
Modem capable	--
RDA numerical	--
RDA String	--
Hysteresis	X
extended API	--
Supports status bit **WR-SUC**	X
alternative IP address	X

Requirements:	
Hardware PC	Serial interface; standard network card
Software PC	--
Hardware PLC	--
Software PLC	--
Requires v-dll	X

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

5 Requirements

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

5.1 PC

HARDWARE

▶ Serial interface

▶ Ethernet TCP/IP

For dual endpoint, the configured listening socket must be configured in the firewall accordingly.

SOFTWARE

If not already present, copy the driver **DNP3_NG.exe** to the zenon program folder and ensure that **DNP3_NGV.dll** is also present.

CE

Copy the driver **DNP332.dll** to the zenon CE program directory. (The DN3_NGV.dll is not required for Runtime.)

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

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.
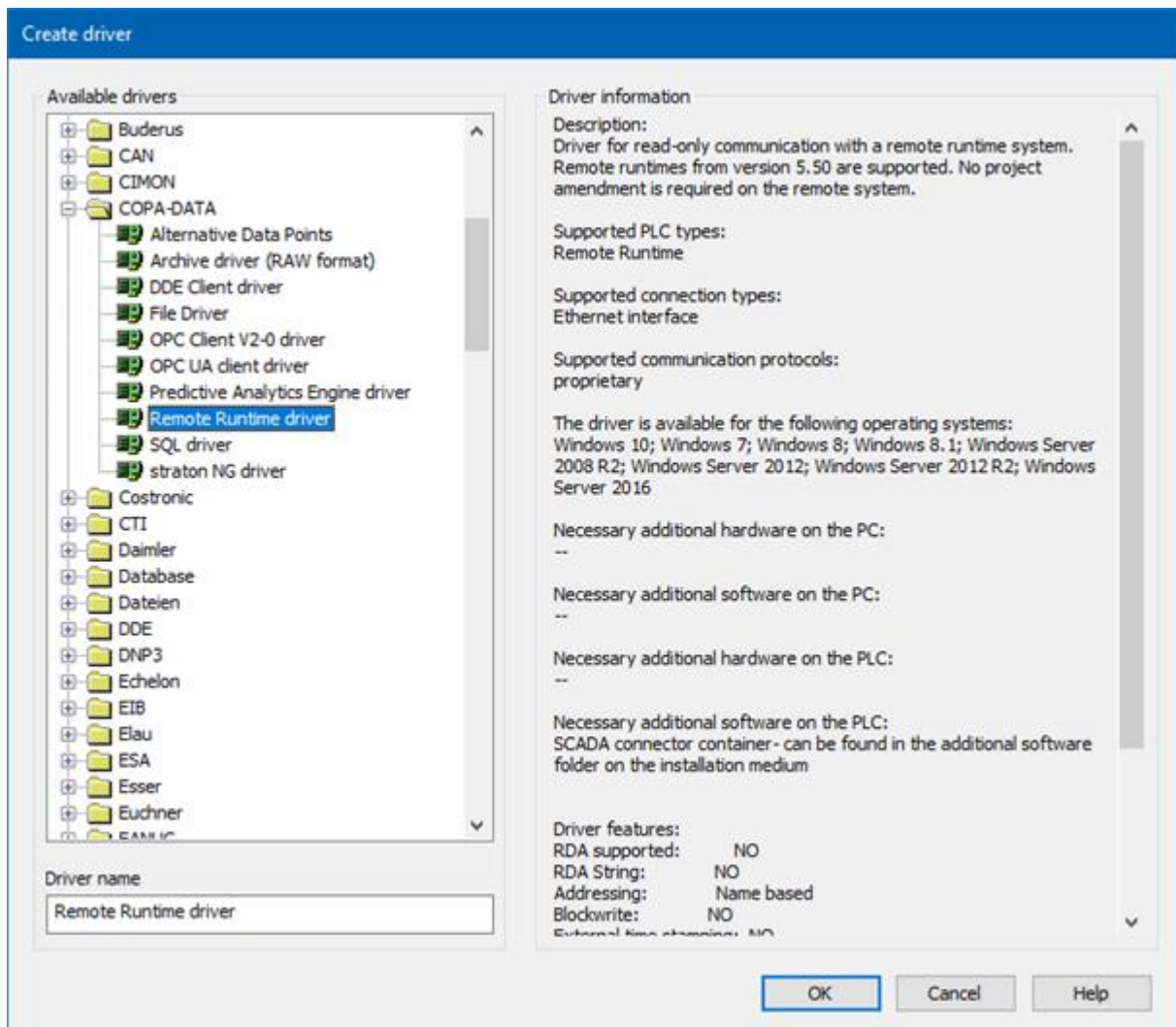
Parameter	Description
Available drivers	List of all available drivers. The display is in a tree structure: *[+]* expands the folder structure and shows the drivers contained therein. [-] reduces the folder structure Default: *No selection*

Parameter	Description
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.

   The **Create simple data type** dialog is opened.

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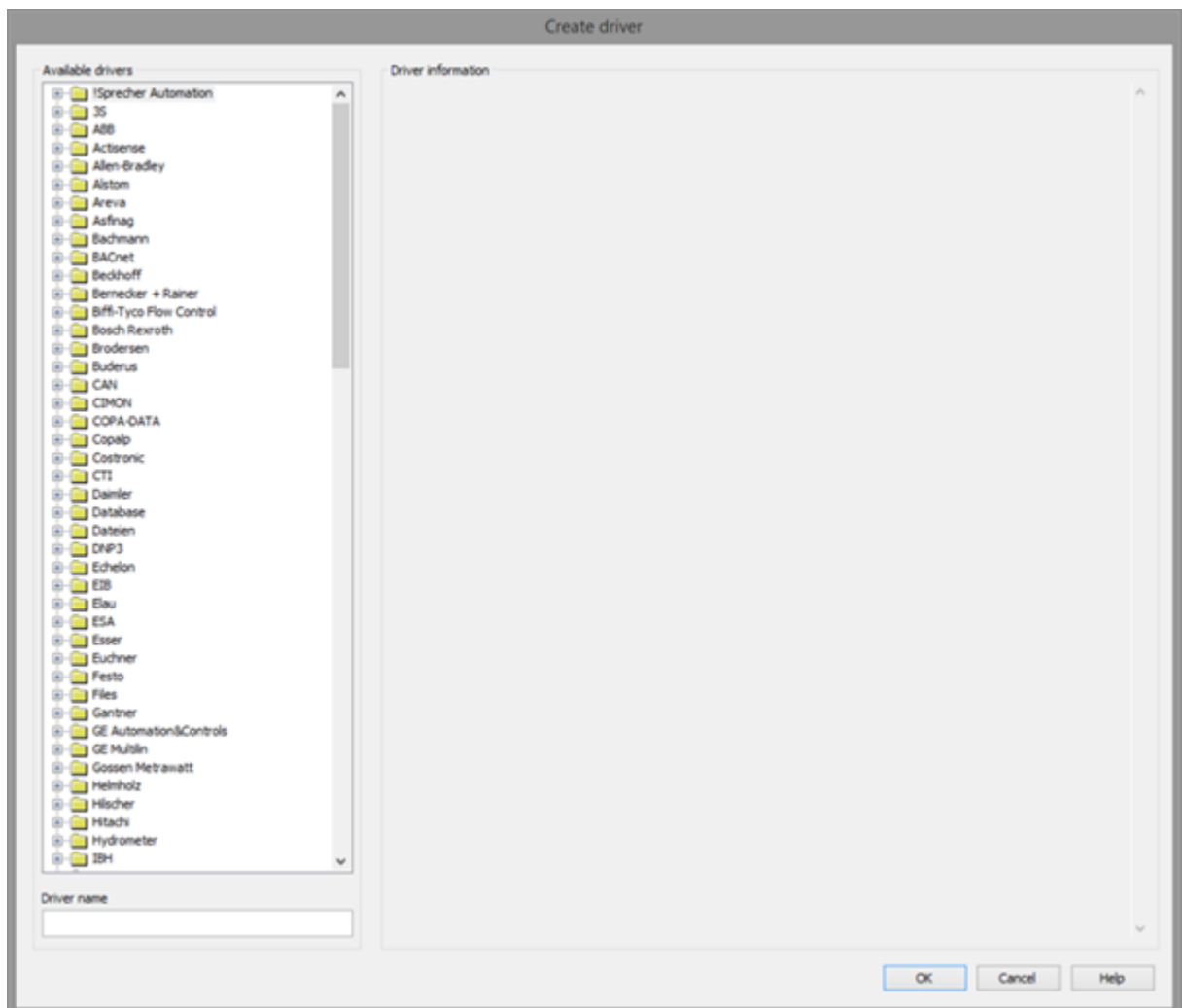
3. Select the desired driver and name it in the **Driver name** input field.
   This input field corresponds to the **Identification** property. The name of the selected driver is automatically inserted into this input field by default.
   The following is applicable for the **Driver name**:

   ▸ 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.
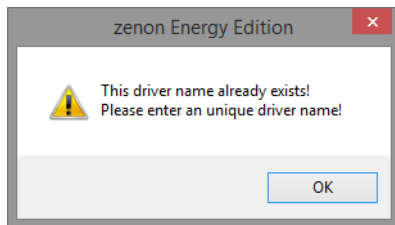
ZENON PROJECT

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

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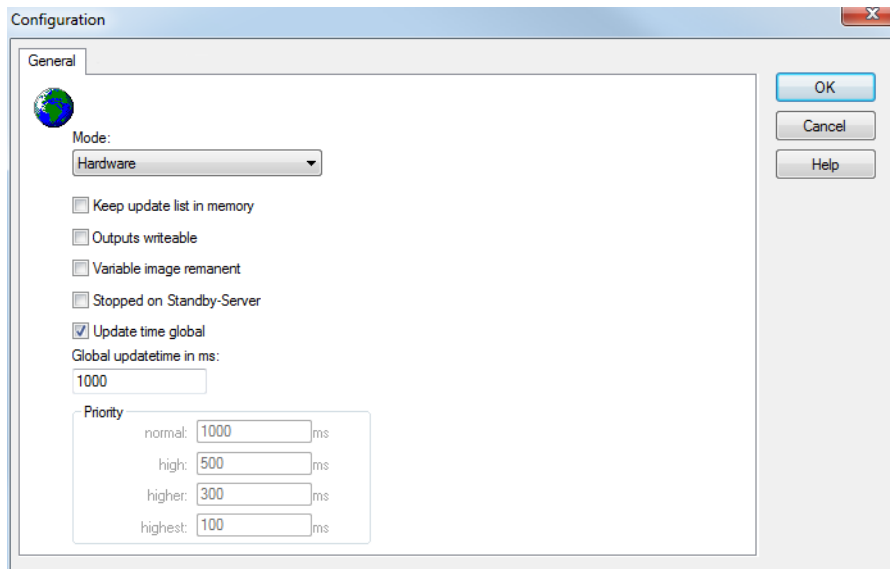- ▸ **MathDr32**
- ▸ **SysDrv**

> ♦ **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.

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 within a value range automatically.
	▸ *Simulation - programmed*: No communication is established to the PLC. The

Option	Description
	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.
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 **Communication details** object type ▸ the driver runs in simulation mode. (not programmed simulation) The following states are not restored at the start of the Runtime:

Option	Description
	▸ *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** 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. **Exceptions:** Spontaneous drivers ignore this option. They generally use the shortest possible update time. For details, see the **Spontaneous driver update time** section.

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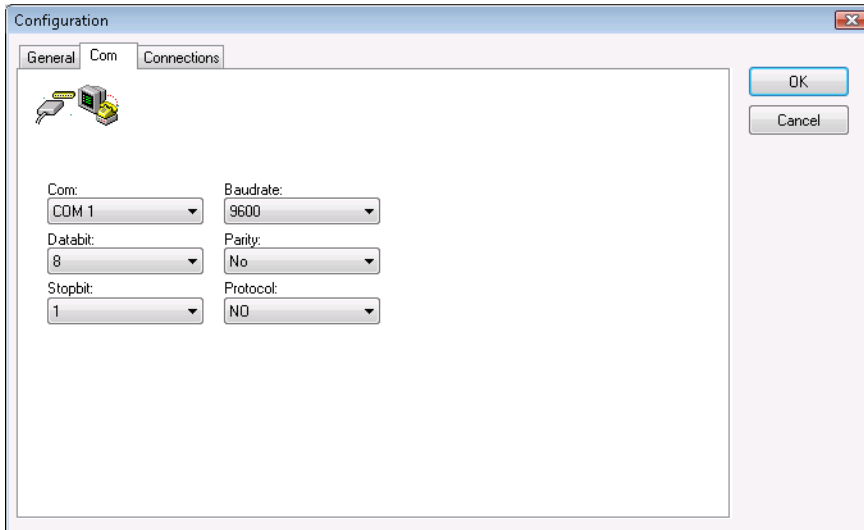
Option	Description
OK	Applies all changes in all tabs and closes the dialog.
Cancel	Discards all changes in all tabs and closes the dialog.
Help	Opens online help.

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 Com

Parameter	Description
Com	Selection Com port.
	Default: *1*
Baud rate	Selection baud rate. Amend to controller. Select from drop-down list:
	Default: *9600*
	Input range: *110* to *256000*
Data bits	Number of data bits. Adapting to PLC.
	Default: *8*
Stop bit	Selection stop bit. Adapting to PLC.
	Default: *1*
Parity	Selection parity. Adapting to PLC.
	Default: *No*
Protocol	Selection protocol. Adapting to PLC.
	Default: *No*

Information

The exact settings depend on the used PLCs. Take the valid values from the manual of your PLC.

6.2.3 Connections
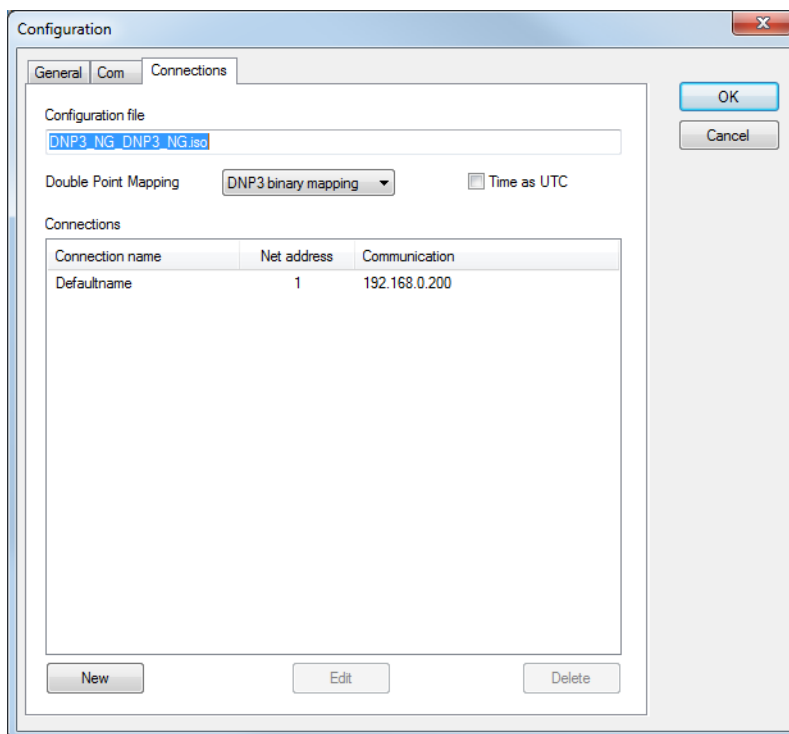
Parameter	Description
Configuration file	Name of the file, in which the connection information is stored. For your information only. Cannot be changed.
Double Point Mapping	Type of double point mapping (on page 27) to an integer value. Select from drop-down list: ▸ *SCADA default mapping* ▸ *DNP3 binary mapping* ▸ *Custom legacy mapping* Default:*DNP3 binary mapping*
Time as UTC	All times from and to the control unit are treated as UTC and not

Parameter	Description
	as local time.
Connections	Displays the configured connections.
New	Opens the dialog for creating a new connection (on page 23).
Edit	Opens dialog for editing the selected connection.
Delete	Deletes the selected connection.
OK	Accepts changes in all tabs and closes dialog.
Cancel	Discards all changes and closes the dialog.
Help	Opens online help.

🔅 Information

Maximum number of connections: 256 (0-255).

CREATE NEW CONNECTION

1. click on the button **New**
2. Enter the connection details.
3. Click on **Save**

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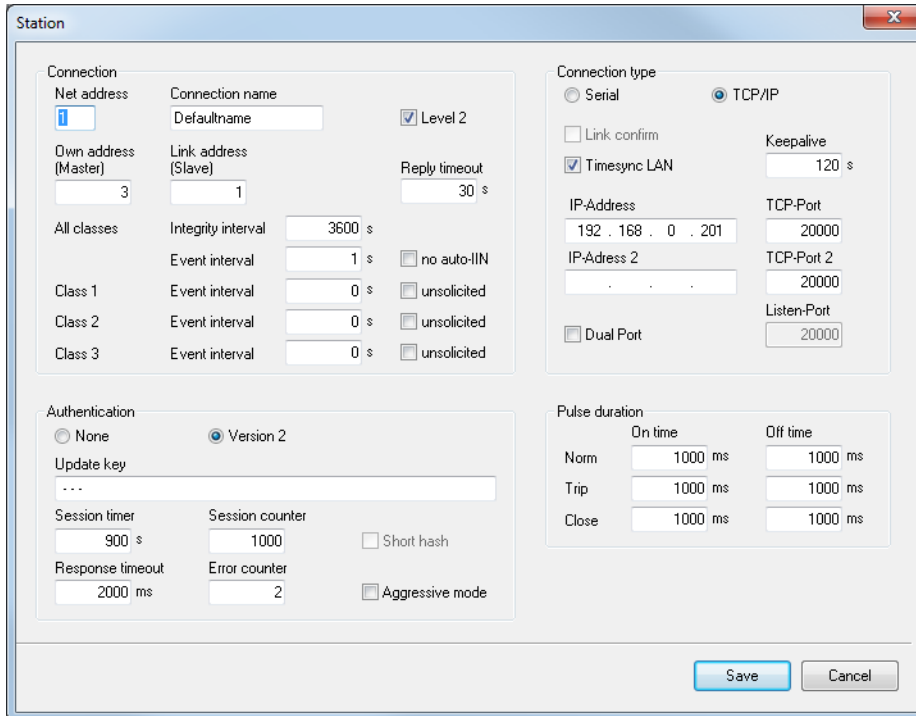
1. select the connection in the connection list
2. click on the button **Edit**
3. change the connection parameters
4. finish with **Save**

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

6.2.3.1 Configuration stations

Click on the button **New** in the configuration dialog opens the dialog for the station configuration.

Parameter	Description
Connection	Connection settings
Net address	Net address of the connection. Value between *0* and *255*.
Connection name	Name of connection. Freely selectable.
Level 2	**Active:Commands and functions for Levels 2** and **level 1** can be used.
Own address (master)	Own address (DNP3 master driver).
Link address (slave)	Link address of the PLC (of the DNP3 slave).
SBO Timeout	Time in seconds which is waited for an answer from the PLC after the *Select*. If the time expires, the *Select* is considered as denied. Default: *30 s*
All classes	Integration interval and event interval for all classes.

Parameter	Description
Integrity interval	Integrity poll of the driver in seconds. Default: *3600*
Event interval	Prompting the driver for new events in seconds. ▸ Value: *0 to 99999 s* ▸ 0: *no polling* Default: *1*
no Auto-IIN	Compatibility: Driver ingors Internal Indication Flags IIN1.1, IIN1.2 and IIN 1.3 Make sure to poll for events regularly to avoid pufferoverflows in the Outstation.
Class 1	
Event interval	Specific Intervals for event-polling in seconds for this class ▸ Value: *0 to 99999 s* ▸ 0: *no polling* Default: *0*
unsolicited	*Active*: For this class, unsolicited Events are accepted. (See also chapter adressing (on page 32) .)
Class 2	
Event interval	Specific Intervals for event-polling in seconds for this class ▸ Value: *0 to 99999 s* ▸ 0: *no polling* Default: *0*
unsolicited	*Active*: For this class, unsolicited Events are accepted. (See also chapter adressing (on page 32) .)
Class 3	
Event interval	Specific Intervals for event-polling in seconds for this class ▸ Value: *0 to 99999 s* ▸ 0: *no polling* Default: *0*
unsolicited	*Active*: For this class, unsolicited Events are accepted. (See

Parameter	Description
	also chapter adressing (on page 32) .)
Connection type	Type of connection. Possible selection via Radiobuttons: ‣ serial ‣ TCP/IP
Serial	*Actively*: serial connection is used
TCP/IP	*Active*: Connection via TCP/IP is used
Link Confirm	*Active*: Link Layer Confirmation is active. Only available for serial communication.
Link Timeout	Time in seconds for connection timneout Default: *30s*
Timesync LAN	*Active*: Use suitable time synchronization. Only available for TCP-connection. Use variation 3 of the time object. This is not accepted of all stations (for example Brodersen RTU) and can be deselected.
IP address	IP address of PLC.
TCP port	Port that is used for communication. Default: *20000*
IP address 2	Backup address for redundant connection to the PLC.
TCO port 2	TCP Port on the backup address
Dual Port	Active: Dual Endpoint communication is permitted.
List Port	TCP Port that is used as local Dual Endpoint. Only active if property **Dual Port** is active.
Authentication	Selection of the authentication: ‣ none ‣ Version 2
None	*Active*: No authentication filter is used.

Parameter	Description
Version 2	*Active*: Authentication version 2.
Update Key	The authentications key for the secured communication via version 2. 32 Hexadecimal Digits expected. **Permitted characters:** ▸ Digits: *0* to *9* ▸ Letters: *a* to *f* and *A* to *F* ▸ Special characters: Space, points and colon ▸ at the start: Header *0x* or *0X* permitted ▸ All remaining letters are interpreted as *0* and will possibly fail during generating the session key.
Session Timer	Validity of the session key in seconds. Value: *<700000 s* Default: *900 s*
Session counter	Select how often a session key may be used. Value: *<10000 s* Default: *1000*
Answer Timeout	Maximal time to the reply of an authentication inquiry in milliseconds. Value: *100 to 120000 ms* Default: *2000 ms*
Error counter	Specification how many mistakes are reported in the authentication. Value: *0 to 10* Default: *2*
Short Hash	*Active*: In serial communication, a shortened Hash value is used in the authentication. **Note:** Only for serial connection and only with **session Timer** until *1800 s* (*30 minutes*) allowed.
Aggressive Mode	*Active*: Authentication is carried out in the aggressive

Parameter	Description
	mode.
Pulse duration	Defines pulse duration for **Norm**, **Trip** and **Close** for each connection. **Norm**, **Trip** and **Close** define which relays are switched.

Norm	Equals NUL.
	
	Pulse duration norm:
	
	  ▶ **On time:** Period of time, in milliseconds, in which the output is ON
	     Default: *1000*
	
	  ▶ **Time off** Period of time, in milliseconds, in which the output is OFF
	     Default: *1000*

Trip	Pulse duration trip:
	
	  ▶ **On time:** Period of time, in milliseconds, in which the output is ON
	     Default: *1000*
	
	  ▶ **Time off** Period of time, in milliseconds, in which the output is OFF
	     Default: *1000*

Close	Pulse duration close:
	
	  ▶ **On time:** Period of time, in milliseconds, in which the output is ON
	     Default: *1000*
	
	  ▶ **Time off** Period of time, in milliseconds, in which the output is OFF
	     Default: *1000*

Save	Saves parameters for connection and deactivates editing mode.
Cancel	Cancels changes and deactivates editing mode without saving.

6.2.3.2 Double Point Mapping

Double Point Mapping, dependent on the selection in the configuration daialog (on page 21)

Offset 15	Offset 14	DNP3 binary	Custom Legacy	SCADA default
0	*0*	*0*	*0*	*2*
0	*1*	*1*	*2*	*0*
1	*0*	*2*	*1*	*1*
1	*1*	*3*	*3*	*3*

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

▶   as structure variables

VARIABLE DIALOG

To create a new variable, regardless of which type:
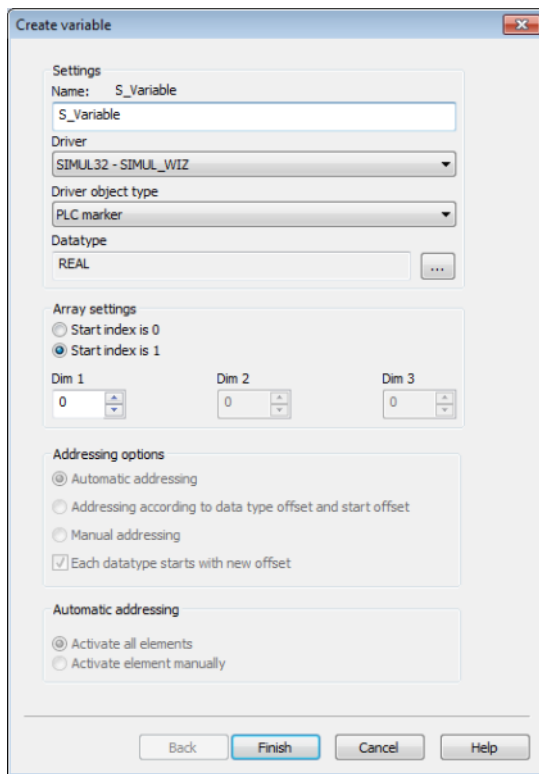
1. Select the **New variable** command in the **Variables** node in the context menu
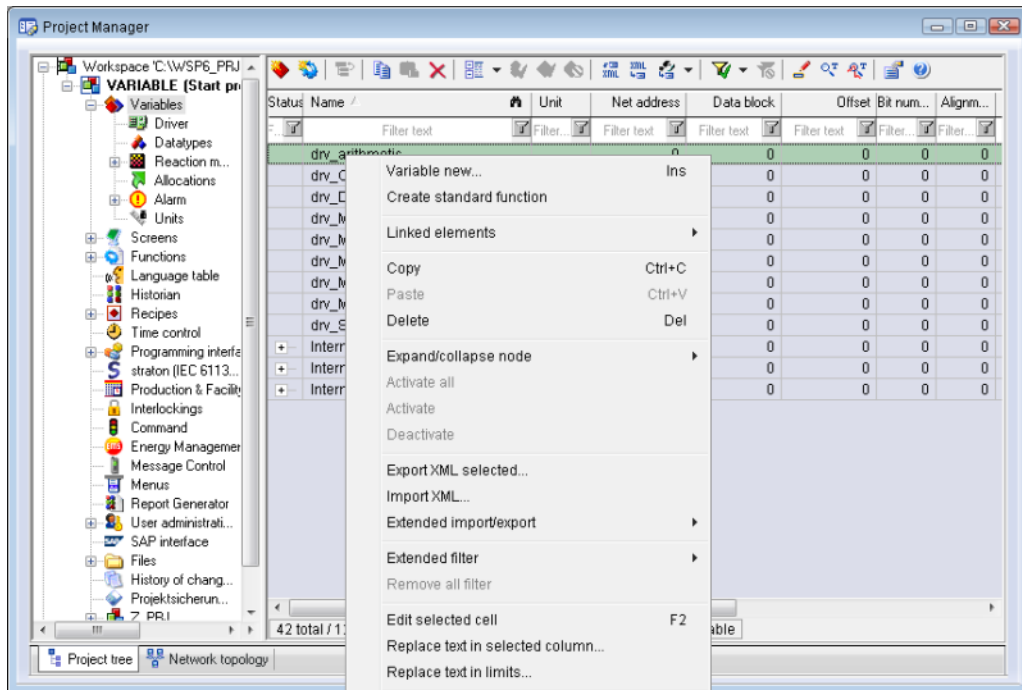
The dialog for configuring variables is opened

2. Configure the variable

3. The settings that are possible depend on the type of variables

CREATE VARIABLE DIALOG

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: Some drivers also allow addressing using the **Symbolic address** property.
Driver	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**) is automatically loaded.
Driver Object Type	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 selection dialog.
Array settings	Expanded settings for array variables. You can find details in the

Property	Description
	Arrays chapter.
Addressing options	Expanded settings for arrays and structure variables. You can find details in the respective section.
Automatic element activation	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**
- ▶ **EUROMAP63**

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

The offset determines the DNP Point number. In combination with the driver object type, the DNP object group is determined.

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	Network address of variables.
	This address refers to the bus address in the connection configuration of the driver. This defines the PLC, on which the variable resides.
	Attention: The DNP address of the outstation is configured separately during the connection.
Data block	not used for this driver
Offset	DNP Point in the Outstation. To see always in combination with the driver object type. For example: **Analog Input**, **Group**20, **Point**0 here corresponds to **Offset** 0 with an *analog input* variable.
Alignment	not used for this driver
Bit number	not used for this driver
String length	Only available for String variables. Maximum number of characters that the variable can take.
Driver connection/**Driver Object Type**	Object type of the variables. Depending on the driver used, is selected when the variable is created and can be changed here.
Driver connection/**Data**	Data type of the variable. Is selected during the creation of the

Group/Property	Description
Type	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.
Command Mode	For *Binary Output* and *Analog Output* only. Determines whether the driver carries out a *Direct operate*, or a *Select Before Operate*. Default:*direct operate*
Variation	Only for variables in report direction, they are not either available in the classpoll or are supposed to be read for a special variation. If you want a different variation than the reported variation from the outstation, the **Permanently read variable** property must be activated in the variable. Variables are read explicit classless, not optimized: ‣ after every Integrity poll or ‣ manually activated via offset *9* of the class poll variables

COMMUNICATION

The communication is mainly polling. The driver searches the outstation for events (value changes and status changes). The interval can be configed. These are assigned in the outstation classes (*1*, *2* or *3*).

According to configuration in the Outstation more than one value for a single DNP object can be sent as an answer. (**Sequence Of Events Buffer** in contrast to **Latest Value-in** of the outstation).

The driver processes the received values and sends them to the Runtime.

If the Outstation supports this, you can activate **unsolicited** (on page 23) responses in the driver. In this case, the driver sends, at the end of the startup routine after the Integrity Poll has been concluded, the request to activate **unsolicited** responses for the respective class. The Outstation can send subsequently value changes without request of the master (Polling of eventclasses) to the master. In this case, it is not unconditionally required, to poll events regulary. The Polling cycle can then generaly be adjusted to *0* or higher (Polling for event classes deactivated). The polling for events can be activated also manually by the Runtime (Class poll variables).

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	Write	Supported data types	Comment
Analog Inputs	68	X	--	*REAL*	
Analog Output Status	69	X	X	*UDINT, DINT, REAL, UINT, INT*	
Binary Inputs	64	X	--	*BOOL*	
Binary Inputs Double	71	X	--	*USINT*	
Binary Output Statuses	65	X	X	*BOOL, USINT*	
Frozen Counters	67	X	X	*BOOL, UDINT, DINT, USINT, UINT, INT, SINT*	
Running Counters	66	X	X	*BOOL, UDINT, DINT, USINT, UINT, INT, SINT*	
String Data	70	X	X	*STRING*	
Communication details	35	X	X	*BOOL, SINT, USINT, INT, UINT, DINT, UDINT, REAL, STRING*	Variables for the static analysis of the communication; Values are transferred between driver and Runtime (not to the PLC).

Driver Object Type	Channel type	Read	Write	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 55) chapter.

Key:

X: supported

--: not supported

COMMANDS

BINARY OUTPUTS

Command processing with *binary output* variables:

▸ Only double messages can be used.

▸ The **Select Before Operate** property must be deactivated for the variable; the DNP3_NG driver uses *Auto-SBO* if necessary.

Use of the **qualifier of command** is recommended for the command processing in order to determine the type:

Type	QoC
PULSE OFF/PULSE ON	*1*
LATCH OFF/LATCH ON	*0*
TRIP/CLOSE	*2*

The **Qualifier of Command** option can be used with both *binary output* variables of the *BOOL* data type as well as the *USINT* data type.

ANALOG OUTPUTS

Use the *write set value* action text for *analog outputs*. According to the **Command Mode** option for the variable, for *Analog Outputs* either a *direct operate* or a *select before operate* is carried out. The **Select Before Operate** option must be deactivated for the variable. The **Qualifier of Command** option is not available for the *Write set value* action and has no influence with *Analog Outputs*.

For two-stage command processing, the command is only sent to the outstation at the second stage of the command, including Select   with *Auto-SBO*.

BEHAVIOR OF COMMAND MODE

The following is applicable for the **Command Mode** property for a variable:

▶ *direct Operate:* The value is written directly. (Default)

▶ *automatic SBO:* When writing from the stack, a Select is sent first, which is then confirmed with an immediate *Operate* if the answer is positive.

This applies for binary and analog outputs.

ASSIGNMENT OF WRITE SET VALUE TO BINARY OUTPUT WITHOUT COMMAND PROCESSING

▶ For a binary output variable of BOOL datatype, *LatchON* is sent for *High* or *LatchOFF* for *Low*.

▶ For a USINT data type binary output variable, the set value is handled in accordance with the table below

value USINT	Action	Comment
0	None	
1	Pulse On	
2	Pulse Off	Not fully compatible. Is not necessarily supported by the outstation.
3	Latch On	
4	Latch Off	
65	Close	
129	Trip	

The **Command Mode** is also taken into account for direct writing of set values without a command processing

RULES FOR SETTING THE STATUS BIT WHEN USING THE COMMAND PROCESSING

Status bits are set according to the rules in the table:

▸ As soon as a *Select* is sent, *SE* and *CoT_act* are set.

▸ If an error occurs when sending, or a negative answer has been received, *SE*, *P/N* and *CoT_actcon* are set (*4*).

▸ If a timeout follows instead of an answer, *SE*, *P/N* and *CoT_actterm* are set (*5*).

▸ The state is *SE* and *CoT_actcon* (*1, 2, 3*) after a positive response. The *Operate* is now sent automatically and the state is set to *CoT_act*. If an error occurs when sending, or a negative response was received, *P/N* and *CoT_actcon* are set (*2*).

▸ If, instead of a response, a timeout follows or if the answer a timeout of *Select*, *P/N* and *CoT_actterm* are set (*3*).

▸ If the response to the *Operate* is positive, the state is initially set to *CoT_actcon* and then to *CoT_actterm* (*1*).

▸ The state after a *DirectOperate* is also handled (*1, 2, 3*).

▸ In the event of an error in sending, the invalid bit is set (*2, 4*).

Fall	Action	Status	Success	Status	Action	Status	Success	Status	Status
1	select	act SE	ack	actcon SE	operate	act	ack	actcon	actterm
2	select	act SE	ack	actcon SE	operate	act	nack	actcon P/N	
3	select	act SE	ack	actcon SE	operate	act	t/o	actterm P/N	
4	select	act SE	nack	actcon SE P/N					
5	select	act SE	t/o	actterm SE P/N					

Meaning of the terms in the **Success** column:

▸ ack = positive

▸ nack = negative

▸ t/o = Timeout

CHANNEL TYPE

The term **Kanaltyp** is the internal numerical name of the driver object type. It is also used for the extended DBF import/export of the variables.

"**Kanaltyp**" is used for advanced CSV import/export of variables in the "**HWObjectType**" column.

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
	BOOL	8
	USINT	9
	SINT	10
	UINT	2
	INT	1
	UDINT	4
	DINT	3
	ULINT	27
	LINT	26
	REAL	5
	LREAL	6
	STRING	12
	WSTRING	21
	DATE	18
	TIME	17
	DATE_AND_TIME	20
	TOD (Time of Day)	19

DRIVER OBJECT TYPES AND SUPPORTED IEC DATA TYPES FOR VARIABLES IN ZENON

Driver object types	Channel type	Supported data types (DataType)	Read	Write	Comment
Binary Input Group 1 Event Group 2	64	BOOL	X	--	
Binary Inputs Double	71	USINT	X	--	USINT-Mapping of 2 binary states with sequenced offsets. (No Couble-Bit Binary Input! Group 3 / 4)
Binary Output status Group 10 Event Group 11 Command Group 12	65	BOOL	--	X	Value 1: LATCH_ON Value 0: LATCH_OFF
Binary Output status Group 10 Event Group 11 Comamnd Group 12	65	USINT	--	X	Value 1: PULSE_ON Value 2: PULSE_OFF Value 3: LATCH_ON Value 4: LATCH_OFF Value 65: CLOSE Value 129: TRIP Value 1 for PULSE_ON, LATCH_ON and CLOSE, value 0 for PULSE_OFF, LATCH_OFF and TRIP
Running Counter Group 20 Event Group 22	66	UINT, UDINT	X	--	
Frozen Counter Group 21	67	UINT, UDINT	X	X	Writing freezes the counter

Driver object types	Channel type	Supported data types (DataType)	Read	Write	Comment
Event Group 23					
Analog Input Group 30 Event Group 32	*68*	*INT, DINT, REAL, LREAL*	X	--	
Analog Output status Group 40 Event Group 41 Command Group 42	*69*	*INT, DINT, REAL, LREAL*	--	X	written value is mirrored as a response after successful writing
String Data Group 110 Event Group 111	*70*	*STRING*	X	X	sent values are not mirrored. Get the latest values via update.
Class Scans	*73*	*BOOL*	--	X	status INVALID until Integrity Poll has finished after driver start. Classpoll value "1" when successful Cold Restart: Value "1" during Cold Restart
Device Attributes Group 0	*72*	*UINT, STRING*	X	--	

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.

EXAMPLE BINARY INPUTS DOUBLE

The driver does mapping via driver object type *Binary Inputs Double* according to the settings in the driver configuration.

For example: Using USINT Offset 14 combine the binary states with offset 14 and 15.

The mapping of two *Binary Inputs* to a *Binary Input Double* is a driver-internal function and should not be confused with the DNP3 *group 3/4*, driver object type*Double-Bit Binary Inputs*, which are currently not supported by the driver.

RESULTS

Binary status bits SCADA default mapping	Binary status bits DNP3 binary logic mapping	Binary status bits Custom legacy mapping	Status in zenon USINT with *Binary Inputs Double)*
▸ both bits: *off (false)*	▸ first bit (offset 14): *off (false)* ▸ second Bit (Offset 15): *on (true)*	▸ first Bit (Offset 14): *on (true)* ▸ second Bit (Offset 15): *off (false)*	*2 - intermediate*
▸ both bits: *on (true)*	▸ both bits: *on (true)*	▸ both bits: *on (true)*	*3 - faulty*
▸ first Bit (Offset 14): *on (true)* ▸ second Bit (Offset 15): *off (false)*	▸ both bits: *off (false)*	▸ both bits: *off (false)*	*0 - off*
▸ first Bit (Offset 14): *off (false)* ▸ second Bit (Offset 15): *on (true)*	▸ first Bit (Offset 14): *on (true)* ▸ second Bit (Offset 15): *off (false)*	▸ first Bit (Offset 14): *off (false)* ▸ second Bit (Offset 15): *on (true)*	*1 - on*

DEVICE ATTRIBUTES

Currently device attributes are only read from the driver. Index *1* is always used (standard set of device attributes). The **Variation** matches the offset of the variable. The user must select the correct data type. When creating manually, the network address must be set according to the station number.

Variation   *254* (all Device Attributes) can be read. Notice: This value is not send directly to the variable in the runtime. Instead, existing **Device Attributes** are automatically updated with the response to the read request from **Variation**   *254*.

BINARY OUTPUTS

The writing of *Binary Outputs* is always carried out by means of a **CROB** (group 12), with a choice of *Direct Operate* or *Select Before Operate*. In doing so, the setting for the variable-specific property **Command Mode** (accessible via XML export/import and VBA **COMMAND_MODE**) is used. Direct writing to group *10* is not supported by the driver.

After successful writing, the variable receives the value *0* for *Binary Outputs* with   **PULSE_OFF**, **LATCH_OFF** and **TRIP**, and value *1* for **PULSE_ON**, **LATCH_ON**   and**CLOSE**. The value is also updated if a *binary output status* (group 10) or *binary output status event* (group 11) object is received.

Binary Output Command Events (Group 13) are currently parsed by the driver, but not sent to the Runtime.

ANALOG OUTPUTS

The writing of Analog Outputs is always carried out by means of a group 41, with a choice of *Direct Operate*   or *Select Before Operate*. In doing so, the setting for the variable-specific property **Command Mode** (accessible via XML export/import and VBA **COMMAND_MODE**) is used.

After successful writing, the variable first receives the value written in Runtime. The value is also updated if an *Analog Output Status*   (group 40) or *Analog Output Status*   (Group 42) object is received.

Analog Output Command Events(Group 43) are currently parsed by the driver, but not sent to the Runtime.

CLASS SCANS

Variables of the *Class Scan* driver object type are control variables to influence driver behavior and are not read by the outstation. *Class scan* variables must be created manually. In doing so, the network address must correspond to the station address in the driver configuration. The offset of the variable determines the function that is executed by the driver.

Class Scan variables have the status *INVALID*   in Runtime for as long as the integrity poll has not yet successfully been concluded by the driver. As soon as the *Integrity Poll* has been concluded, the variables are given the status *SPONT*.

The following offsets are currently supported:

▸   **Offset 0:**
    An integrity poll will be triggered when writing to this variable (Read Request Group 60,

Variation 2,3,4,1). If reading is successful, the variable receives the return value *1*. Any classless variables that may exist are then read.

▶ **Offset 1:**
During writing to this variable a class 1 poll will be triggered Read Request Group 60, Variation 2. If reading is successful, the variable receives the return value *1*.

▶ **Offset 2:**
During writing to this variable a class 2 poll will be triggered Read Request Group 60, Variation 3. If reading is successful, the variable receives the return value *1*.

▶ **Offset 3:**
During writing to this variable a class 3 poll will be triggered Read Request Group 60, Variation 4. If reading is successful, the variable receives the return value *1*.

▶ **Offset 9:**
During writing to this variable a reading of all classless variables will be triggered. If reading is successful, the variable receives the return value *1*.

▶ **Offset 13:**
A cold restart command is sent to the outstation when writing to this variable (Function Code 13). The object of group 52 contained in the response from the outstation (time delay) is evaluated by the driver and the driver sends no new requests to the outstation for the period specified in the Time Delay object. In Runtime, the value of this variable is "0" as long as Cold Restart is active; the value becomes *1* after that.

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 manual in the Variables 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** 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 instructions of 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 instructions of the import 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	Type	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	Num	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	Float	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	Type	Field size	Comment 48
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	Type	Field size	Comment	
MAXGRAD	N	16	Gradient overflow for counter REMA.	49

> **⚠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.

7.4.3 Online import

To import variables online from the PLC:

1. select the driver

2. Select **Import variables from driver** in the toolbar or in the context menu

3. The dialog for the import is opened
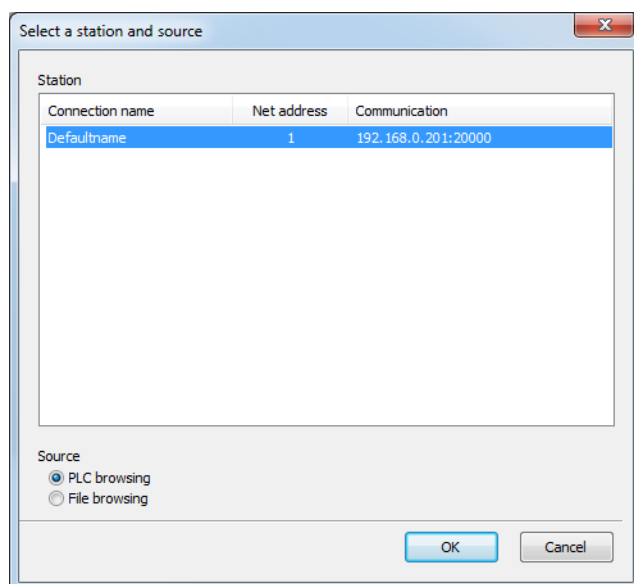
4. Select the desired connection

5. Select **Selection of PLC**

6. Confirm the selection by clicking **OK**

7. The dialog for variable selection is opened
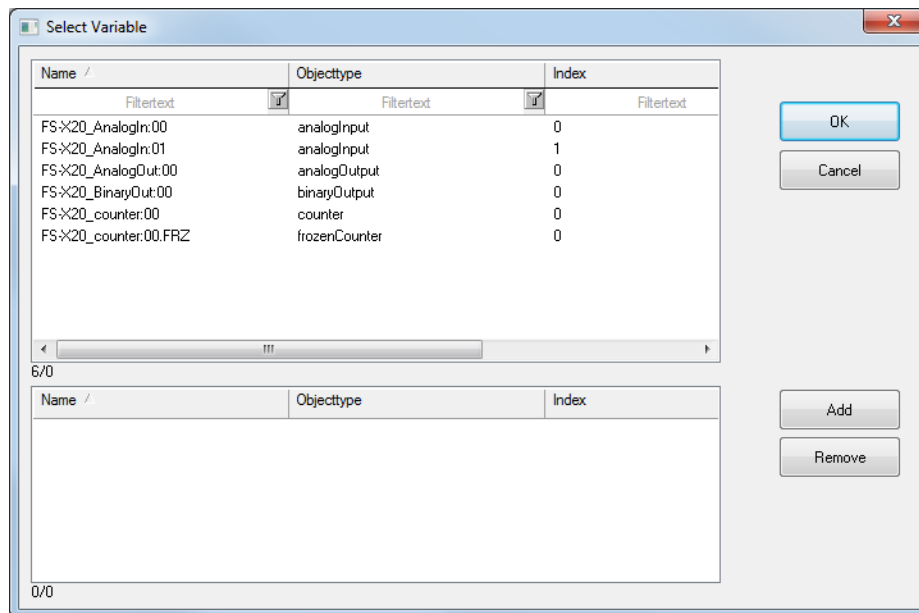
8. select the desired variables (multiple selection is possible)

9. Add selected variables via click on button **Add** to the list of the variables to be imported.
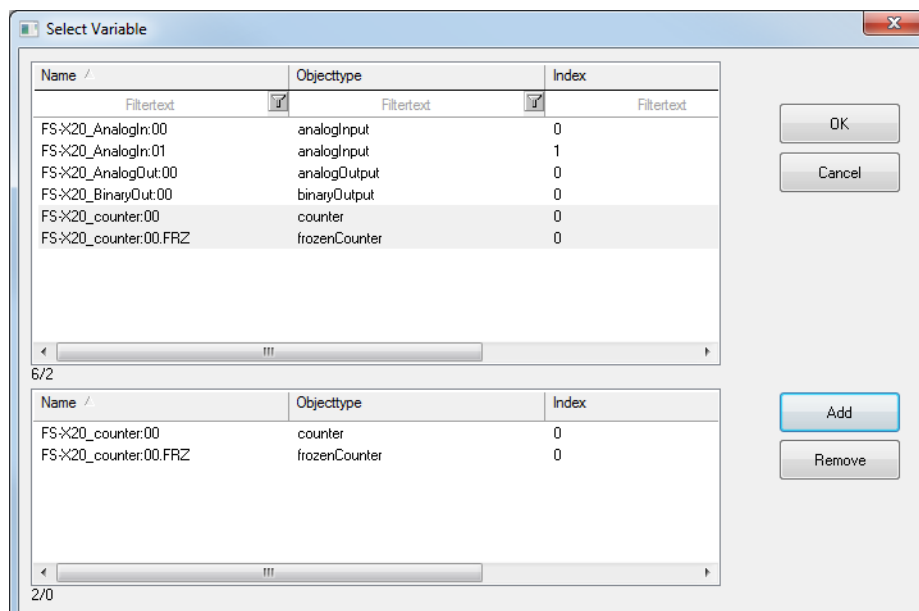
10. You can also deselect variables again by clicking on **Remove**.

11. start the import by clicking on the **OK** button

The selected variables are generated automatically during import in the zenon project and are assigned the selected driver. The **Net address** of the variables is configured according to the selected station in the driver configuration (on page 23).

RULES FOR THE ONLINE IMPORT

For Online-Import:

▶ The response to an integrity poll is evaluated.

▶ The name is created from **Net address**, connection name, group number and index.

▶ The identification contains **Net address**, index and a description of the object type.

▶ Ensure that Runtime is not active if you start an online import; under certain circumstances, the outstation only supports a master or a connection from the same computer.

▶ Ensure that the response timeout in the driver configuration is set higher accordingly if you are using an outstation with a large point database and a slow (serial) connection.

7.4.4 Offline import

The driver supports the Offline import of variable out of a **DNP3 XML Device profile** file for the versions:

▶ 2.07 (January 2012)

▶ 2.08 (July 2012)

To import variables from a **DNP3 XML Device profile** file:

1. select the driver

2. Select **Import variables from driver** in the toolbar or in the context menu

3. The dialog for the import is opened
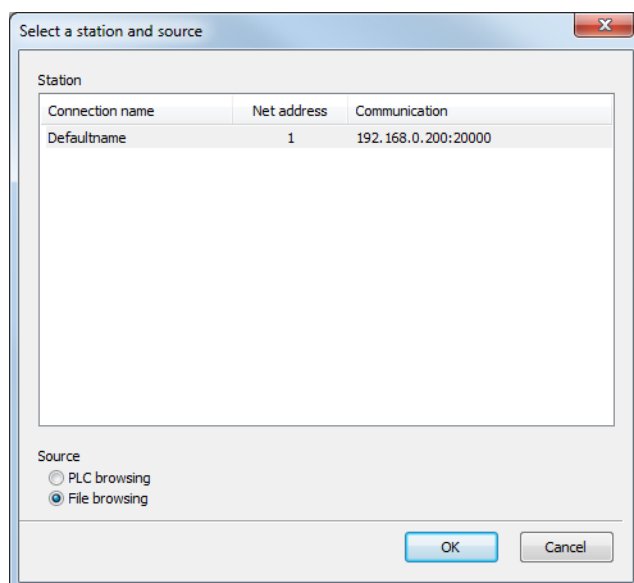
4. Select the desired connection

5. select **Import from file**

6. the dialog for file selection will be opened

7. select the desired file and confirm this selection by clicking **OK**

8. The dialog for variable selection is opened
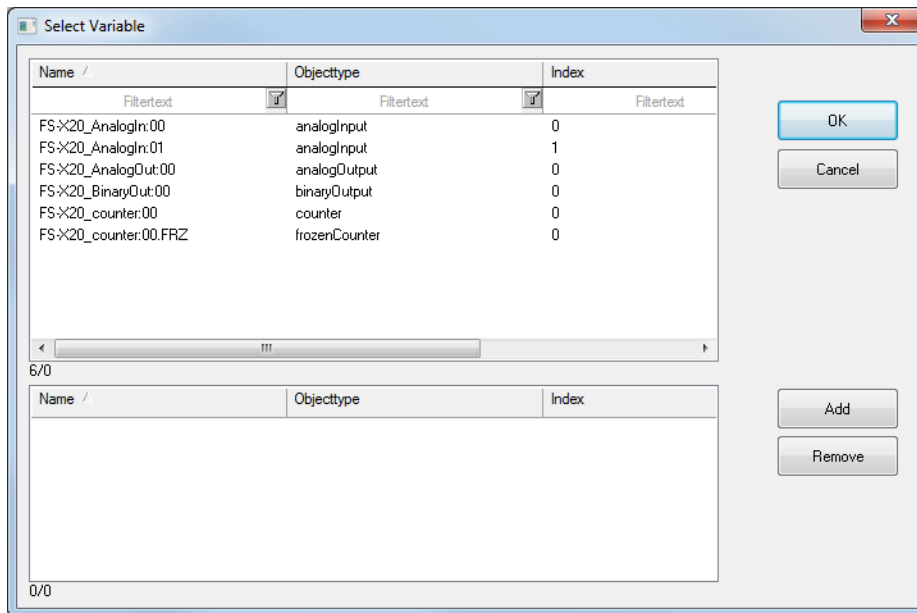
9. select the desired variables (multiple selection is possible)

10. Add selected variables via click on button **Add** to the list of the variables to be imported.
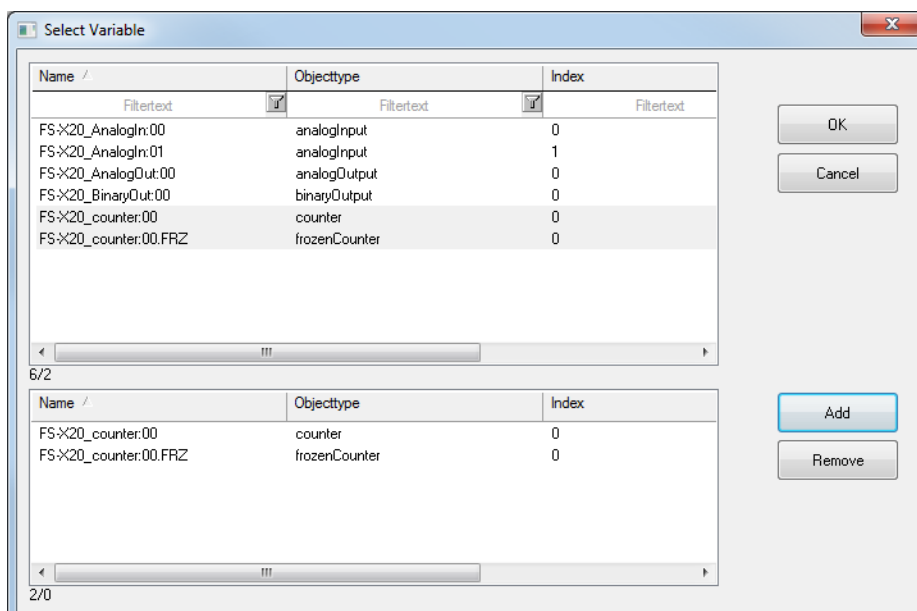
11. You can also deselect variables again by clicking on **Remove**.

12. start the import by clicking on the **OK** button

The selected variables are generated automatically during import in the zenon project and are assigned the selected driver. The **Net address** of the variables is configured according to the selected station in the driver configuration (on page 23).

RULES FOR THE OFFLINE IMPORT

For Offline-Import:

▸ Variable definition must contain a name.

▸ The variable name is made up of a composite name comprising the XML device profile document in accordance with the following scheme:

devicename_variablename

It must be ensured that all DNP variables have a unique name in the document, including throughout the group.

▸ If the definition of the variable in the document contains a field **Description**, this information is stored in the **Identification** of the variable during import. The name of the variable and the variable ID can be changed after import.

▸ If a variable with the same name already exist in the project, you receive an error during a new import. The variable is not overwritten or merged. This error message can also be displayed:

  ▸ if the document does not use unique names in the **XML Device Profile**

  ▸ if the device name in the document is identical with already imported variable

▸ Only variables from the **XML device profile** document that are supported by the driver are offered for import.

▸ **Frozen counters** are not explicitly present in the **XML device profile**. If however the value for **frozenCounterExists** is set to *true* for a counter (*1*), the option to also import variables for **frozen counter** is offered.

NOT IMPORTED VARIABLES

The following variables are not imported and must be created manually:

▸ Device attributes

▸ Control variables for classpoll, classless reading and cold restart.

▸ Binary inputs double

Note the correct **Net address** when creating variables manually

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

Name from import	Type	Offset	Description
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
LineStateOnHoldPendTransfer	*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 state of Runtime for driver simulation.
ConnectionStates	*STRING*	61	Internal connection status of the driver to the PLC.

Name from import	Type	Offset	Description
			Connection statuses: ▸ *0:* Connection OK ▸ *1:* Connection failure ▸ *2:* Connection simulated Formating: **<Net address>:<Connection status>;...;...;** 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).

Name from import	Type	Offset	Description
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.
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 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

Name from import	Type	Offset	Description
			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.
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:

Parameter	Description
Blockwrite	Not supported.
Redundancy	if the outsation supports several masters
RDA	not supported **Sequence Of Events** is supported if activated in the Outstation.
Real time stamping	Yes, if the variation in the outstation is configured

Parameter	Description
	accordingly.
Browsing	Online and Offline.
Polling	Polling for event classes, in configurable cycles or controlled manually.
Spontaneous	Yes. Only value changes are transferred.
Number of PLCs	One driver can connect to any number of outstations. For serial communications, several outstations can be configured for a serial interface (bus). Mixed operation *serial/TCP* is also possible with a driver.

ERROR FILE

The driver supports central logging in the Diagnosis Server.

COMMUNICATION DETAILS

Driver statistics variables are not set by the DNP3_NG driver. Communication monitoring is possible using the INVALID status bit.

EXTENDED ERROR FILE

The driver supports extended logging in to the Diagnosis Server. Configuration is performed via the Diagnosis Viewer.

INTEGRITY POLL

After the driver is started, an Integrity Poll is automatically sent by the driver to the outstation (**Read Request Group** *60*, **Variation** *2,3,4* and *0*, **Range** *all*). The values received are are available in Runtime accordingly. In this case, ensure that the outstation sends a variation that is different to a normal value change as a response to the integrity poll. (with/without time stamp, with/without flags.)

An integrity poll can also be instigated explicitly in Runtime by means of a *class poll* type control variable. If the driver is stopped and started using the driver command function, this also triggers an integrity poll.

INTERNAL INDICATIONS (IIN)

The outstation can inform the master of its status via **internal indication** status bits.

The DNP3_NG driver evaluates **internal indication** bits as follows:

▸ *IIN 1.1 CLASS_1_EVENTS:*
The outstation sets this bit if the event buffer for class 1 contains other DNP objects that are not included in the current answer. The DNP3_NG master reacts to this bit in that a read query for group 60, variation 2,3,4 is immediately sent to the outstation. If the outstation frequently sets this status bit, this can lead to a higher read cycle that that defined in the driver configuration.

▸ *IIN 1.2 CLASS_2_EVENTS:*
The outstation sets this bit if the event buffer for class 2 events contains further DNP objects that are not included in the current response. The DNP3_NG master reacts to this bit in that a read query for group 60, variation 2,3,4 is immediately sent to the outstation. If the outstation frequently sets this status bit, this can lead to a higher read cycle that that defined in the driver configuration.

▸ *IIN 1.3 CLASS_3_EVENTS:*
The outstation sets this bit if the event buffer for class 3 contains other DNP objects that are not included in the current answer. The DNP3_NG master reacts to this bit in that a read query for group 60, variation 2,3,4 is immediately sent to the outstation. If the outstation frequently sets this status bit, this can lead to a higher read cycle that that defined in the driver configuration.

▸ *IIN 1.4 NEED_TIME:*
The outstation sets this bit if a time synchronization is demanded by the master. The DNP3_NG master reacts immediately and responds to the outstation depending on the selected time synchronization option in the driver configuration.

▸ *IIN 1.7 DEVICE_RESTART:*
Is set by the outstation in the event of a restart. If a cold restart is sent to the outstation by the DNP3_NG master, the outstation sets this bit. It is reset by the DNP3_NG master in this process.

▸ *IIN 2.2 PARAMETER_ERROR:*
Is set by the outstation, if, for example, an object that is not present in the outstation with this DNP3 index is explicitly read. In this case, the DNP3_NG sets the INVALID status bit for the variable in Runtime.

If the **NO_AUTO_IIN** option is set to active in the driver configuration for the station, the driver ignores the **Internal Indication** flags *IIN1.1*, *IIN1.2* and *IIN1.3*. The driver does not send an automatic read query for group 60, variation 2, 3, 4. This option can be activated in order to circumvent compatibility problems if the outstation of one of these flags is not reset in time, which leads to the driver only sending event polls.

In general, the outstation sets these flags in order to command the master to read again, because other data is available and this could therefore possibly avoid a buffer overflow in the outstation. Note this if you activate this option for compatibility reasons, and ensure that the driver polls the outstation for events at regular intervals.

DNP3 OBJECT FLAGS MAPPING

The DNP object flags **ONLINE** and **COMM_LOST** are currently evaluated by the DNP3_NG driver. With **ONLINE** = *false* or **COMM_LOST** = *true*, the *INVALID* bit of the variable is set in the Runtime. Note that the **Variation** configured in the outstation determines whether objects are sent with or without flags for an object group.

TIME SYNCHRONIZATION

The DNP3_NG driver supports time synchronization of outstations with the time of the master station. If the outstation reports a need for time synchronization by means of an internal indication flag 1.4 an, the DNP3_NG driver sends the current system time in accordance with the options for UTC/local time and the LAN time synchronization.

It is not currently possible to send a time synchronization to the outstation cyclically without a request.

Ensure that the outstation gets the time from a different source in this case (such as a GPS receiver) and as a result may possibly not send any requirement for time synchronization to the master. In this case, ensure that the Runtime computer with the DNP3_NG master station is synchronized with the same time source.

The DNP3 protocol does not allow the master to synchronize its own time with the time of the outstation.

SELECT AND CANCEL

The DNP3_NG driver immediately responds positively to a Select and Cancel with a corresponding COT. The Execute is carried out after a Select with the COT amended. Automatic Select and Execute are given preferential treatment in the process.
Furthermore, Select, Cancel and Execute have an additional status bit. As a result of this, orderly Runtime monitoring of the configured routing is possible.

DNP3 SECURE AUTHENTICATION V2

SELECT BEFORE OPERATE - DIRECT OPERATE

The DNP3_NG driver uses **Direct operate** (by default) or **Select Before Operate**, for writing *Binary Output Status* or *Analog Output* type variables. Configuration is carried out using the driver-specific variable property. **Command Mode**.

Ensure that the **Select Before Operate** property remains inactive for the variable. This property changes the behavior of the two-stage command processing and is not compatible with the DNP3 standard. If this property is active however:

  ▸   Block the command processing for further commands if *Direct operate* is set for the variable

▸ Carry out the command at the first level and also for the second level if *Auto SBO* is set for the variable

Note: With the DNP332 driver, a **Select Before Operate** can only be carried out via this option. This is however not compliant with the rules for **Select Before Operate** defined in the DNP3 standard.

DNP3 SEQUENTIAL FILE TRANSFER

DNP3 sequential file transfer Is not currently supported by the DNP3_NG driver.

HYSTERESIS

The driver supports hysteresis for spontaneous values (unsolicited responses). Hysteresis is not taken into account by:

▸ Values that are received as a response to a read request for an event class

▸ Variables that are explicitly read as classless variables

If an identical value, however with a more recent time stamp is received, then this value is sent by the driver as a new value in Runtime.

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!
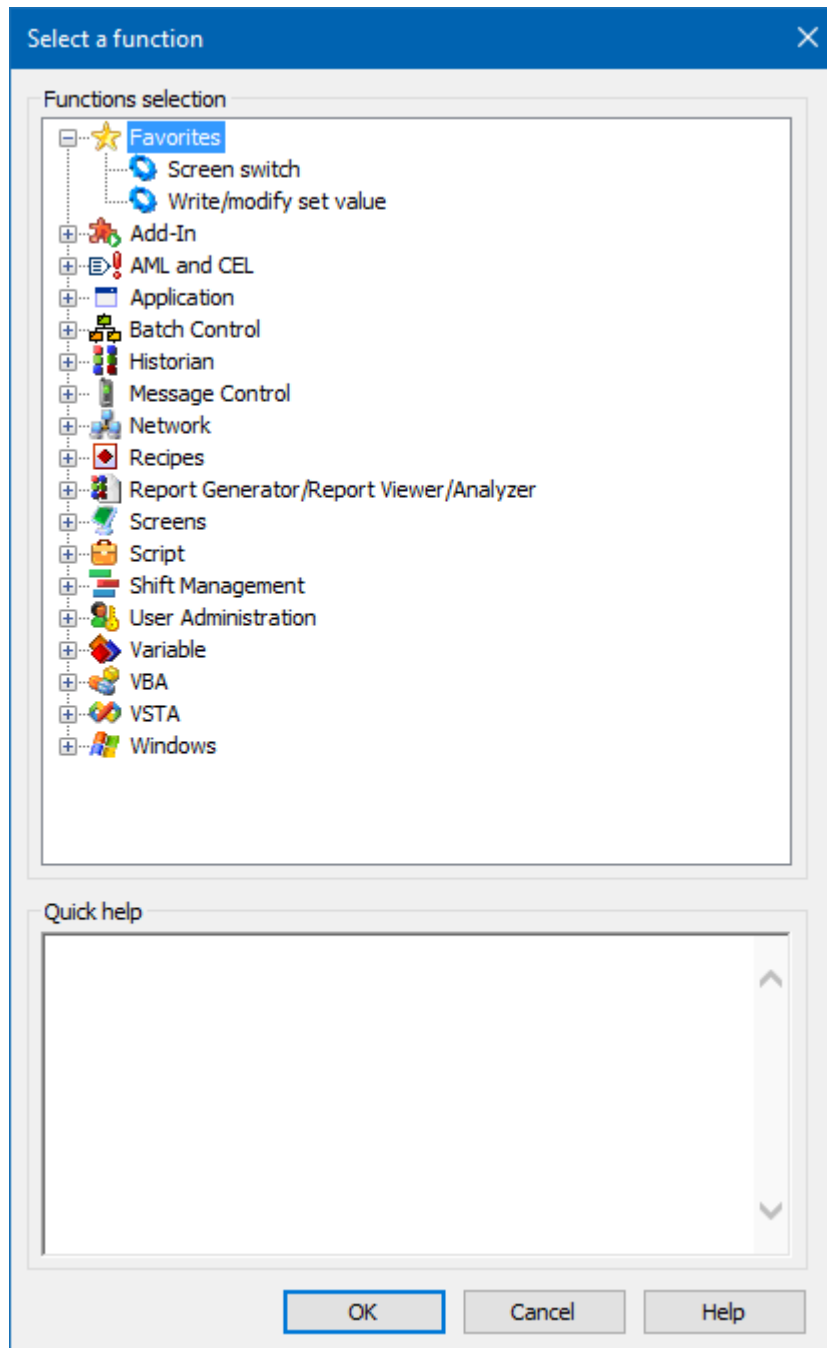
CONFIGURATION OF THE FUNCTION

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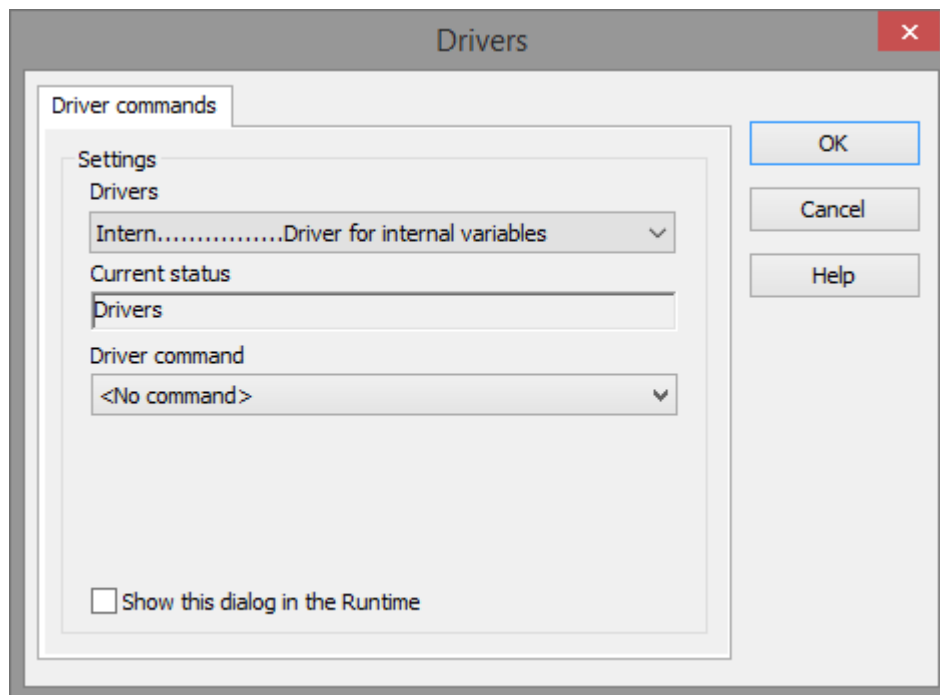To configure the function:

1. Create a new function in the zenon Editor.

   The dialog for selecting a function is opened
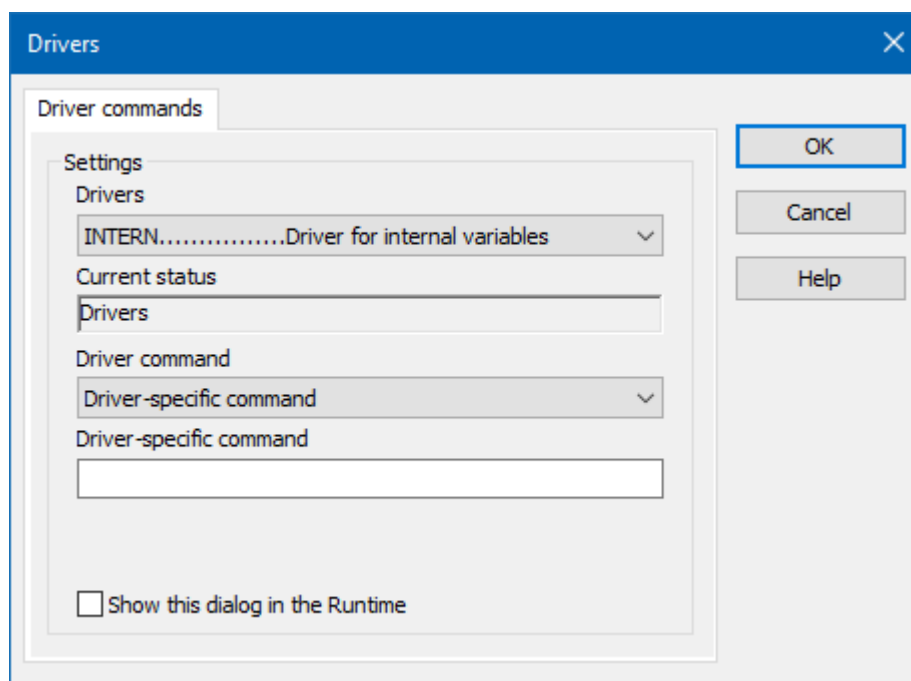

2. Navigate to the node **Variable.**

3. Select the **Driver commands** entry.

The dialog for configuration is opened

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

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. no function in the current version.
Driver command	no function in the current version. 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. **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.

Driver command	Description
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 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.
Driver - activate set setpoint value	Write set value to a driver is possible.
Driver - deactivate set setpoint value	Write set value to a driver is prohibited.
Establish connecton 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.

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 program that was also installed with zenon. You can find it under **Start/All programs/zenon/Tools 8.20 -> 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 manual.

10.2 Driver monitoring

Runtime monitors the availability of the driver by means of a watchdog. If a driver is no longer available, the *INVALID* status bit is also set for all checked-in variables.

Possible causes for a triggering of the watchdog:

▶ The driver process is no longer running.

Check whether the driver EXE file is still running in the Task Manager.

▶ Operating system is busy with processes that have a higher priority.

Check the configuration of your system to see whether there is sufficient memory and CPU power. In this case, the driver only resets the *INVALID* status bit if there is a value change on the connected party. Static values retain the *INVALID* status bit until the next time the Runtime or the driver is started.

CONFIGURATION OF WATCHDOG

For the monitoring of communication in the Runtime, the connection to the driver is checked in a fixed, prescribed time period of 60 seconds. This process is repeated several times. If, within 5 attempts (= within 5 minutes), no valid connection to the driver is detected, the *INVALID* bit is set for the checked-in (*advised*) variables. In addition, the *INVALID* bit is also set when new variables are advised. The *INVALID* bit will no longer be reset.

Corresponding LOG entries are created for this.

LOG ENTRY

An error message is logged in the LOG when the watchdog is triggered:

Parameter	Description
Communication with driver:<drvExe>/<drvDesc>(id:<drvId>) timed out. No communication for <time> ms.	No communication with driver within the given time. ▸ *<time>*: Time (in milliseconds) ▸ *<drvDesc>*: Driver name ▸ *<drvExe>*: Driver EXE name ▸ *<drvId>*: Driver ID in the zenon project
Communication with %s timed out. Invalid-Bit will be set.	Communication to the *%s* driver could not be established after 5 attempts within 60 seconds. The *INVALID* bit is set for the variable.
Communication with %s timed out. Timeout happened %d times	Communication to the *%s* driver could not be established after *%d* times within 60 seconds.

10.3 Check list

Questions and hints for fault isolation:

GENERAL TROUBLESHOOTING

▸ Is the PLC connected to the power supply?

▸ Analysis with the Diagnosis Viewers (on page 69):
   -> Which messages are displayed?

▸ Are the participants available in the **TCP/IP** network?

▸ Can the PLC be reached via the *Ping* command?

Ping: **Open command line -> ping <IP address > (e.g.: ping 192.168.0.100) -> Press the Enter key**.

Do you receive an answer with a time or a timeout?

▸ Can the PLC be reached at the respective port via *TELNET*?

Telnet: **Command line: enter: telent <IP address port number>   (e. g. telnet 192,168,0,100 20000) -> press the enter key**.

If the monitor turns black and the cursor blinks, a connection could be established.

▶ Analysis by using a network monitoring program (Sniffer, e.g. Wireshark, Microsoft Network Monitor / Microsoft Message Analyzer)

▶ Are you using the correct cable which is recommended by the manufacturer for the connection between the PLC and the PC?

▶ Did you select the right COM port?

▶ Do the communication parameters match (Baud rate, parity, start/stop bits,…)?

▶ Is the COM port blocked by another application?

▶ Did you configure the Net address in the address properties of the variable correctly?

    ▶ Does the addressing match with the configuration in the driver dialog?

    ▶ Does the net address match the address of the target station?

▶ Did you use the right object type for the variable?
Example: Driver variables based on driver object type **Communication details** are purely statistics variables. They do not communicate with the PLC.
You can find detailed information on this in the Communication details (Driver variables) (on page 55) chapter.

▶ Does the offset addressing of the variable match the one in the PLC?

SOME VARIABLES REPORT INVALID.

▶ INVALID bits always refer to a net address.

▶ At least one variable of the net address is faulty.

▶ Class poll variables are INVALID after starting the driver as long as the integritypoll was not finished.

VALUES ARE NOT DISPLAYED, NUMERIC VALUES REMAIN EMPTY

Under circumstances, no answer can be received from the PLC for a reading-request.

Driver is not working. Check the:

▶ Installation of zenon

▶ the driver installation

▶ The installation of all components
-> Pay attention to error messages during the start of the Runtime.

VARIABLES ARE DISPLAYED WITH A BLUE DOT

The communication in the network is faulty:

▶ With a network project:
Is the network project also running on the server?

▶ With a stand-alone project or a network project which is also running on the server:
Deactivate the property **Read from Standby Server only** in node **Driver connection**/**Addressing**.

VALUES ARE DISPLAYED INCORRECTLY

Check the information for the calculation in node **Value calculation** of the variable properties.

Check the configuration of the outsation, if the desired variation is sent (p. e.: **Float** with decimal places).

VALUES ARE NOT DISPLAYED PROMPTLY

Check in the outstation, if events are generated for the selected values. Also check in which class they are generated. Check the polling of this event class in the driver configuration.

Check the configuration of **unsolicited** responses in the outstation.

THE TIME STAMP OF THE VARIABLE IS INCORRECT.

Check the configuration of the outstation to see whether the DNP3 objects are actually sent with a time stamp. (corresponding variation selected with time stamp.)

Check to see if the outstation uses local time or UTC and set the option in the driver configuration accordingly,

Check the time of the local computer and the time of the outstation, including settings for the time zone.

DRIVER FAILS OCCASIONALLY

Analysis with the Diagnosis Viewers (on page 69):
-> Which messages are displayed?

Check the timeout of the response time in the driver configuration, especially with outstations with many data points and a slow connection.

With *secure authentication*, check to see if the **pre-shared update keys** in the driver and the outstation are identical.