zenon driver manual
ADP

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 ADP

With the **ADP - alternative data points** driver, a zenon variable can receive values from different sources. Source variables can be either different variables of one driver or variables of different drivers.

Note that the **ADP driver** can only be created once per project. The driver can be found in the driver group COPA-DATA.

Variables of the **ADP driver** are configured for display in zenon Runtime. The parameters for these variables with the *alternative variables* driver object type are set up in the zenon Editor in the **Alternative variable** variable properties group.

▶ A list of source variables is configured in the process. These variables can be configured for different drivers.

▶ With the help of a selector variable you can select the variable which will be linked.

▶ Variables support module **Command Processing**.

In doing so, the **Alternative variable** is available for **ADP driver** variables.

💡 **Information**

Variables of the **ADP driver** of **driver object type** *alternative variable* are not counted for licensing and do not use up any I/Os.

---

3 ADP - data sheet

General:	
Driver file name	ADP.exe
Driver name	Alternative Data Points
PLC types	- Internal
PLC manufacturer	zenon system driver; COPA-DATA

Driver supports:	
Protocol	intern
Addressing: Address-based	Name based
Addressing: Name-based	--
Spontaneous communication	--
Polling communication	--
Online browsing	--
Offline browsing	--
Real-time capable	--

Driver supports:		
		6
Blockwrite	--	
Modem capable	--	
RDA numerical	--	
RDA String	--	
Hysteresis	--	
extended API	--	
Supports status bit **WR-SUC**	--	
alternative IP address	--	

Requirements:	
Hardware PC	--
Software PC	--
Hardware PLC	--
Software PLC	--
Requires v-dll	--

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

4 Driver history

Date	Build number	Change
09.01.2018	45085	Created driver documentation

5 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.

5.1 Creating a driver

In the **Create driver** dialog, you create a list of the new drivers that you want to create.
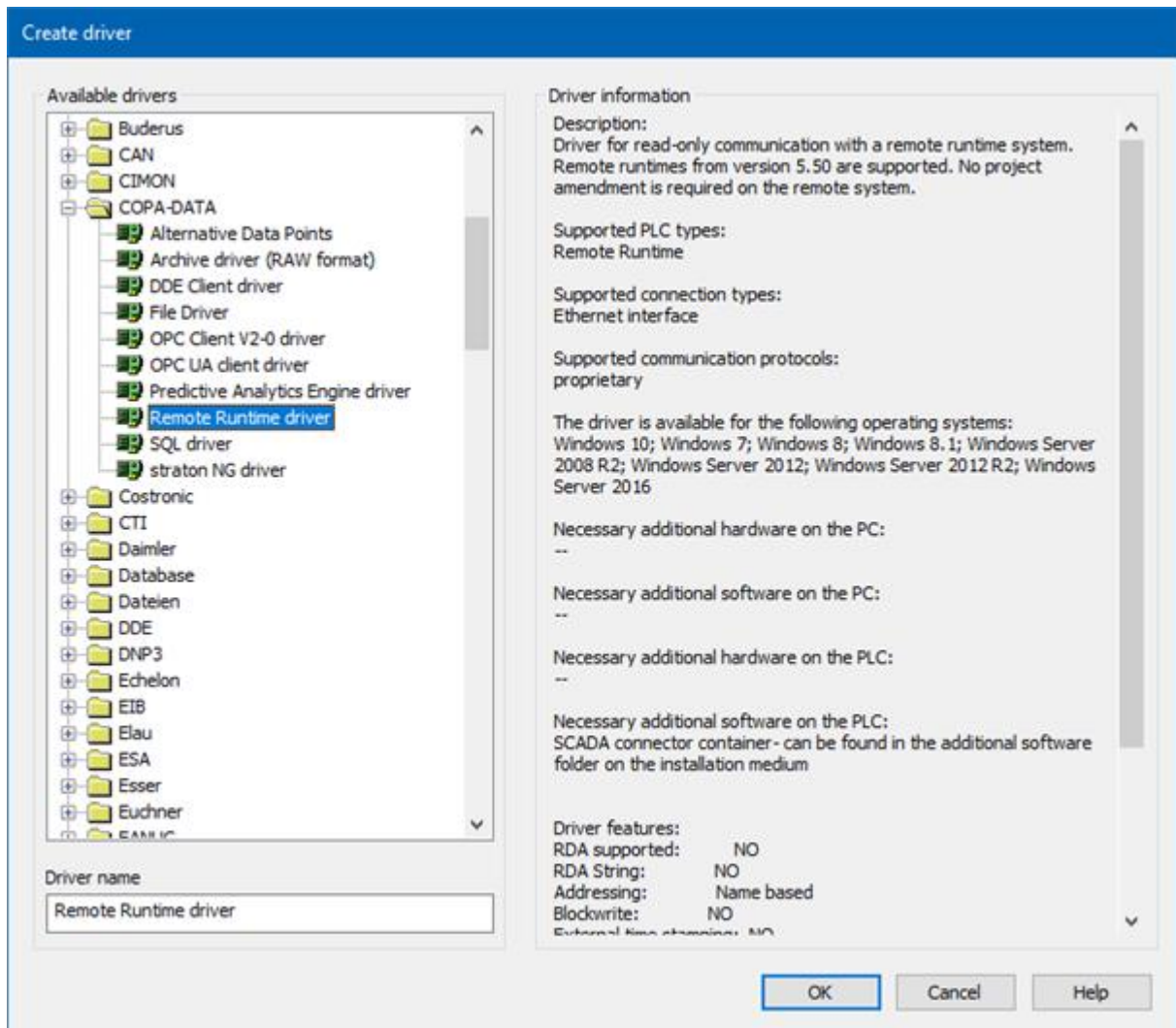
Parameter	Description
Available drivers	List of all available drivers.
	The display is in a tree structure: *[+]* expands the folder structure and shows the drivers contained therein. [-] reduces the folder structure
	Default: *No selection*
Driver name	Unique **Identification** of the driver.
	Default: *empty* The input field is pre-filled with the pre-defined

Parameter	Description
	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].
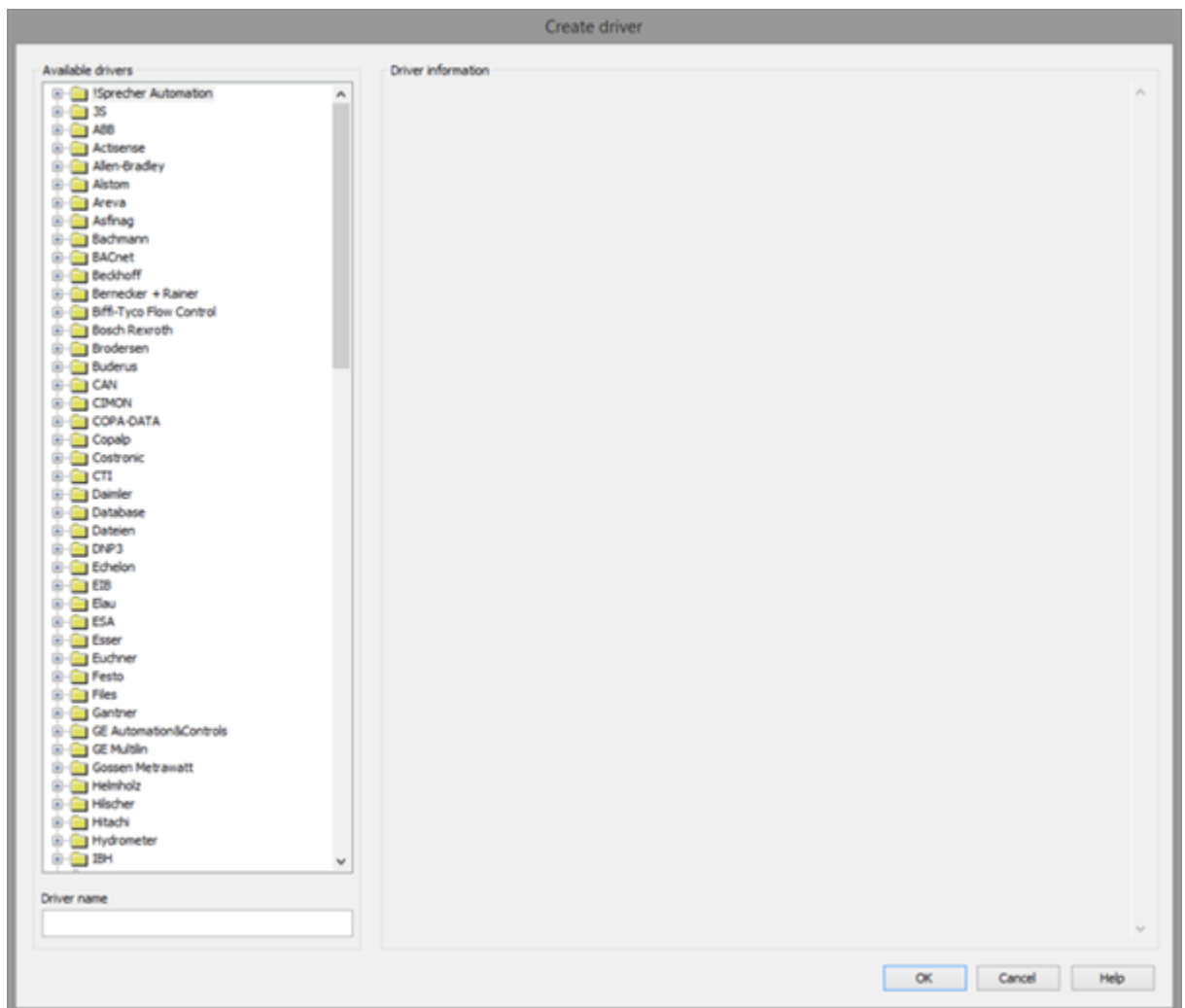
CREATE NEW DRIVER

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.
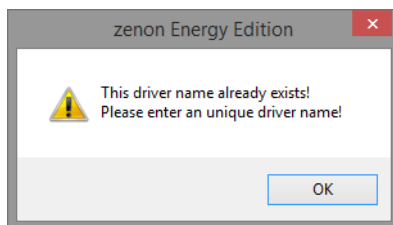
ZENON PROJECT

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.

5.2 Settings in the driver dialog

You can change the following settings of the driver:

6 Creating variables

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

6.1 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:
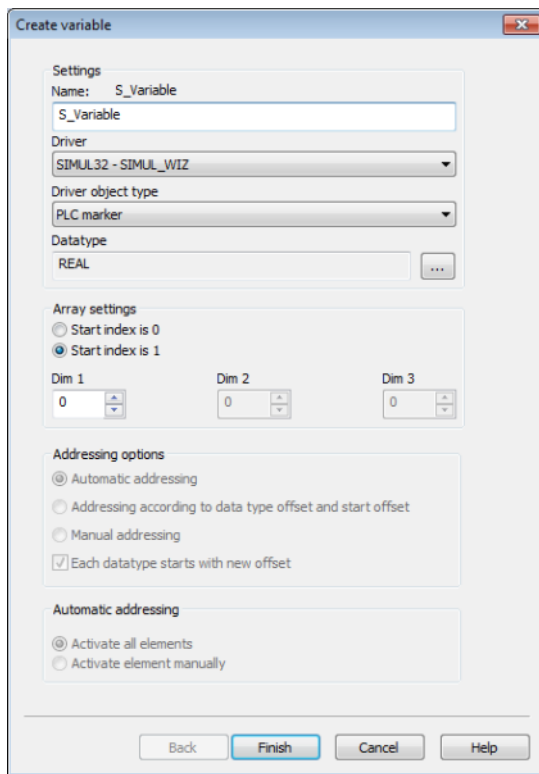
1.   Select the **New variable** command in the **Variables** node in the context menu
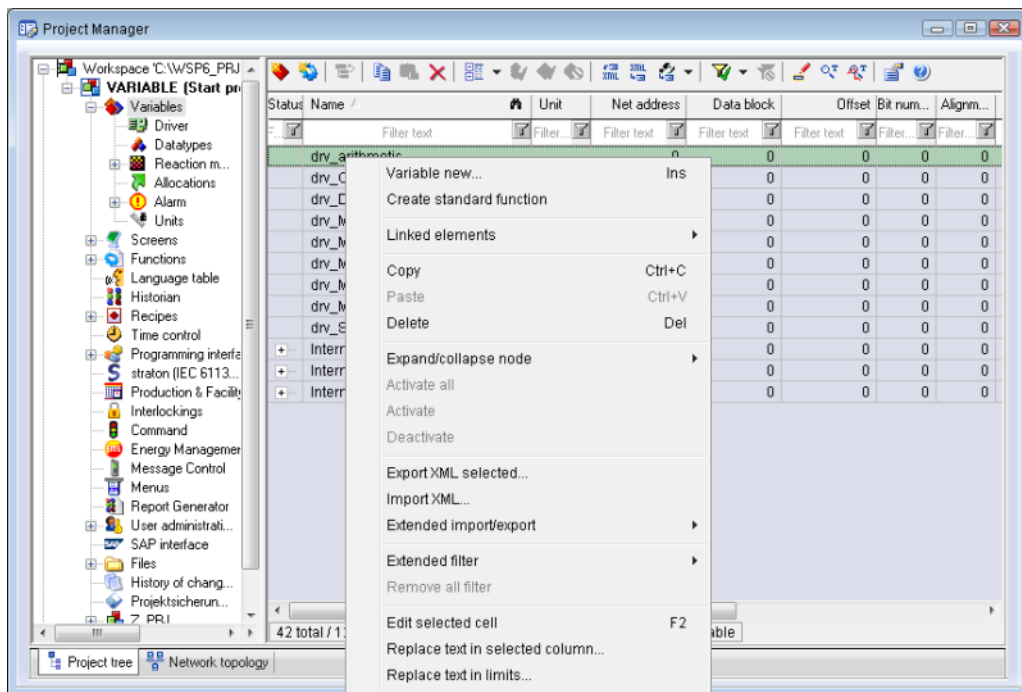
The dialog for configuring variables is opened

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.

6.2   Addressing

Group/Property	Description
General	
Name	Freely definable name.
	Attention: For every zenon project the name must be unambiguous.
Identification	Freely assignable identification, e.g. for resources label, comment ...
Addressing	
Net address	not used for this driver
Data block	not used for this driver
Offset	not used for this driver
Alignment	not used for this driver
Bit number	not used for this driver
String length	not used for this driver
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 Type**	Data type of the variable. Is selected during the creation of the variable; the type can be changed here.
	Attention: If you change the data type later, all other properties of the variable must be checked and adjusted, if necessary.

6.3   Define source variables

The configuring takes place directly in the properties of the variable of **driver object type** *alternative variable*. These properties can be found in property group **Alternative variable** of the variable.

The configuration dialog of the source variable is called via property **Source variable**.

Parameter	Description
[Source variables list]	List of variables which are assigned to the ADP variable as source variable. ▶ *Priority* Priority of the variable This consecutive number is assigned automatically. The lower the number, the higher the priority. If a variable is shifted, its priority number is also adjusted. ▶ *Selector index* Consecutive number of the source variable. This number is assigned automatically and cannot be changed. The number corresponds to the order in which a variable is inserted as a source variable into this dialog. ▶ *Name* Configured name of the source variable. This corresponds to variable property **Name** of the current configuration in the zenon Editor. ▶ *...* Opens the variable selection dialog again **Note:** Is only available if a variable is selected in the list.
Insert...	Opens the variable selection dialog. For selection all already configured variables of all drivers of the active zenon project are offered. Multiple selection is possible. The new source variable is inserted directly under the selected source variable in the list. If no source variable is selected, the new source variable is positioned at the end of the list.
Remove	The selected source variable will be deleted from the list without requesting confirmation. Only active if an already configured source variable is selected in the list.

Parameter	Description
Arrow symbol up	Moves the selected source variable up one entry in the list. Only active if an already configured source variable is selected in the list.
Arrow symbol - on top	Moves the selected source variable to the top of the list. Only active if an already configured source variable is selected in the list.
Arrow symbol - on bottom	Moves the selected source variable to the bottom of the list. Only active if an already configured source variable is selected in the list.
Arrow symbol down	Moves the selected source variable down one entry in the list. Only active if an already configured source variable is selected in the list.

⚠**Attention**

If the same source variable is configured in several ADP variables, it is not guaranteed that each of these ADP variables will be supplied with a value in the Runtime.

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.

6.3.1 Engineering in the Editor - Example

The following example shows how configuration for ADP with two drivers is carried out.

Carry out the following steps in the zenon Editor:

▶ Configure two different drivers.
For example:

    ▶ **IEC 61850**

    ▶ **Modbus Energy**

▶ Configure corresponding variables for these two drivers.
Per example:

    ▶ for the IEC 61850 driver
IEC_61850_Value1

    ▶ for the Modbus Energy driver
Modbus_Energy_Value1

Take care that the variables are configured in the same way for both drivers (e.g. link command variables only to command variables, same parameters for select before operate, ...)

▶ Create a new variable *SourceVariableSelector*, e.g. for the **Internal driver**.

▶ Create a new variable *ADP_State*, e.g. for the **Internal driver**.

▶ Create a new variable for **driver alternative data points**.with driver object type **Alternative variable**.

    ▶ Name this variable *Display_Value1*.

▶ Configure the ADP variable:

    ▶ Switch to the **Alternative variable** variable property group.

    ▶ In the **Source variable** property, click on the **...** button.
The **Define source variable** (on page 15) dialog is opened.

    ▶ In this dialog click on the **Add...**button.
The variable selection dialog is opened.

    ▶ Select the variable *IEC_61850_Value1*.

    ▶ Again click the **Add...** button.
Now select variable *Modbus_Energy_Value1*.

    ▶ Confirm the dialog with **OK**.

▶ Link the ADP variable with a corresponding selector variable.

▶ For this click the **...** button at property **Selector** of the ADP variable.
   The variable selection dialog is opened.

▶ Select the variable *SourceVariableSelector*.

▶ Link the ADP variable with a corresponding status variable.

▶ For this click the **...** button at property **State variable** of the ADP variable.
   The variable selection dialog is opened.

▶ Select the variable *ADP_State*.

If an ADP variable has already been created and linked with a **Selector**, every newly created ADP variable will also be linked with this **Selector**. This applies until the next start of the zenon Editor. If **Selector** is no longer available, the link for the newly created ADP variable remains empty. If a **Selector** in a different project has been linked with an ADP variable, no automatic link will be established.

👍 **Hint**

In addition, **Selector** and **State variable** can be linked with drag&drop. In order to do this open the variable list in an own window.

RUNTIME

For this example the following is true for display in the Runtime:

▶ Variable *Display_Value1* displays the value of the variable which is currently considered in the ongoing process.

▶ The value of variable *SourceVariableSelector* controls which variable is processed in the Runtime.

▶ The value of variable *ADP_State* shows which configured source variable is used in the currently ongoing process.

Examples:
SourceVariableSelector with value 1

▶ The variable *Display_Value1* shows the value of the variable *IEC_61850_Value1*

▶ The value of the variable *ADP_State = 1*.

SourceVariableSelector with value 2

▶ The variable *Display_Value1* shows the value of the variable *Modbus_Energy_Value1*

▶ The value of the variable *ADP_State = 2*.

6.3.2 Automatic switching

The following is applicable for the switch:

▶ Value 1-n switches to the alternative variable. The automatism that is used at value 0 and described below, is not executed.

▶ If the value is *0* or no switch variable has been configured, the first source variable entry is used.

   ▶ Automatic switching is always active if no switch variable has been configured or the switch variable has the value *0*. If the value is zero, the state of the variable is irrelevant.

   ▶ If the current connection is interrupted, the driver automatically switches to the next available source variable. Several source variables are available in the order 1-n.

   ▶ Checking the connection always involves checking all the source variables. The driver always switches to the variable which has the highest configured priority and does not have the state *invalid* (Ibit) or *off* (off bit).

6.4 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.

6.4.1 Driver objects

The following object types are available in this driver:

Driver Object Type	Channel type	Read	Write	Supported data types	Description
Alternative variable	33	X	X	*BOOL, DATE, DATE_AND_TIME, DINT, INT, LINT, LREAL, REAL, SINT, STRING, TIME, TOD, UDINT, UINT, ULINT, USINT, WSTRING*	Variable with configured allocation which existing variable is used in the process (principle of a multiplexer).
Communication details	35	X	X	*BOOL, SINT, USINT, INT, UINT, DINT,*	Variables for the static analysis of the communication; Values

Driver Object Type	Channel type	Read	Write	Supported data types	Description
				UDINT, REAL, STRING	are transferred between driver and Runtime (not to the PLC). **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 29) chapter.

Key:

X: supported

--: not supported

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.

6.4.2 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

PLC	zenon	Data type
	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

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.

6.5   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.

6.5.1 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.

6.5.2 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

Identification	Type	Field size	Comment
			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 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

Identification	Type	Field size	Comment
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
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.

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

⚠Attention

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

LIMIT VALUE DEFINITION

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

Identification	Type	Field size	Comment
AKTIV1	R	1	Limit value active (per limit value available)
GRENZWERT1	F	20	technical value or ID number of a linked variable for a dynamic limit value (see VARIABLEx) (if VARIABLEx is *1* and here it is *-1*, the existing variable linkage is not overwritten)
SCHWWERT1	F	16	Threshold value for limit value
HYSTERESE1	F	14	Is not used
BLINKEN1	R	1	Set blink attribute
BTB1	R	1	Logging in CEL
ALARM1	R	1	Alarm

Identification	Type	Field size	Comment
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 (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.

6.6   Communication details (Driver variables)

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

▸   Information

▸   Configuration

▸   Statistics and

▸   Error message

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

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

💡 **Information**

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

For example:

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

▸ Driver variables for the polling cycle are only available for pure polling drivers.

▸ Connection-related information such as **ErrorMSG** is only supported for drivers that only edit one connection at a a time.

INFORMATION

Name from import	Type	Offset	Description
MainVersion	*UINT*	0	Main version number of the driver.
SubVersion	*UINT*	1	Sub version number of the driver.
BuildVersion	*UINT*	29	Build version number of the driver.
RTMajor	*UINT*	49	zenon main version number
RTMinor	*UINT*	50	zenon sub version number
RTSp	*UINT*	51	zenon Service Pack number
RTBuild	*UINT*	52	zenon build number
LineStateIdle	*BOOL*	24.0	TRUE, if the modem connection is idle
LineStateOffering	*BOOL*	24.1	TRUE, if a call is received
LineStateAccepted	*BOOL*	24.2	The call is accepted
LineStateDialtone	*BOOL*	24.3	Dialtone recognized
LineStateDialing	*BOOL*	24.4	Dialing active

Name from import	Type	Offset	Description
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. Connection statuses: ▸ *0:* Connection OK ▸ *1:* Connection failure ▸ *2:* Connection simulated Formating: **<Net address>:<Connection status>;...;...;** A connection is only known after a variable

Name from import	Type	Offset	Description
			has first signed in. In order for a connection to be contained in a string, a variable of this connection must be signed in once.
			The status of a connection is only updated if a variable of the connection is signed in. Otherwise there is no communication with the corresponding controller.

CONFIGURATION

Name from import	Type	Offset	Description
ReconnectInRead	BOOL	27	If TRUE, the modem is automatically reconnected for reading
ApplyCom	BOOL	36	Apply changes in the settings of the serial interface. Writing to this variable immediately results in the method SrvDrvVarApplyCom being called (which currently has no further function).
ApplyModem	BOOL	37	Apply changes in the settings of the modem. Writing this variable immediately calls the method SrvDrvVarApplyModem. This closes the current connection and opens a new one according to the settings **PhoneNumberSet** and **ModemHwAdrSet**.
PhoneNumberSet	STRING	38	Telephone number, that should be used
ModemHwAdrSet	DINT	39	Hardware address for the telephone number
GlobalUpdate	UDINT	3	Update time in milliseconds (ms).
BGlobalUpdaten	BOOL	4	TRUE, if update time is global
TreiberSimul	BOOL	5	TRUE, if driver in sin simulation mode
TreiberProzab	BOOL	6	TRUE, if the variables update list should be kept in the memory
ModemActive	BOOL	7	TRUE, if the modem is active for the driver
Device	STRING	8	Name of the serial interface or name of the modem

Name from import	Type	Offset	Description
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.

Name from import	Type	Offset	Description
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 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.

Name from import	Type	Offset	Description
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

7 Driver-specific functions

The driver supports the following functions:

▸   Command Processing

7.1 Command Processing

The value of the variable linked to variable property **Selector** determines on which source variable the Command Processing is routed.

⚠Attention

Take care that the configuration of the source variables and ADP variables coincide (e.g. Same configuration for Standby Server, authorization level, ...)

In doing so, the following applies:

The value is routed to the corresponding source variable.

Exceptions:

▶ If the value of the variable linked to property **Selector** is *0*, the routing is carried out on source variable *#1*.
If no variable is linked, it also equals value 0.

▶ If the value of the variable linked to property **Selector** is *smaller than 0* or if the value is larger than the number of configured source variables, no routing is carried out. The variable contains the sate *INVALID*. In addition an entry in the LOG file is created.

▶ If the value of the variable linked to property **Selector** equals a not configured source variable number (linked source variable was deleted or not linked), no routing is carried out. The variable contains the sate *INVALID*. In addition an entry in the LOG file is created.

If property **Read from Standby Server only** is activated at the ADP variable, the action is routed onward to the source variable selected at the Standby Server.

⚠**Attention**

If property **Read from Standby Server only** is activated at the ADP variable, it displays the value of the selected variable at the Standby Server (independent of the configuration of the selected variable).

If an action (e.g. **Write set value** or a Command Processing action) is carried out, the configuration of the selected variable is considered.

The same is true the other way round:
Is property **Read from Standby Server only** inactive at the ADP variable but active at the selected variable, the value from the server (and not from the Standby Server) is displayed.

8 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!
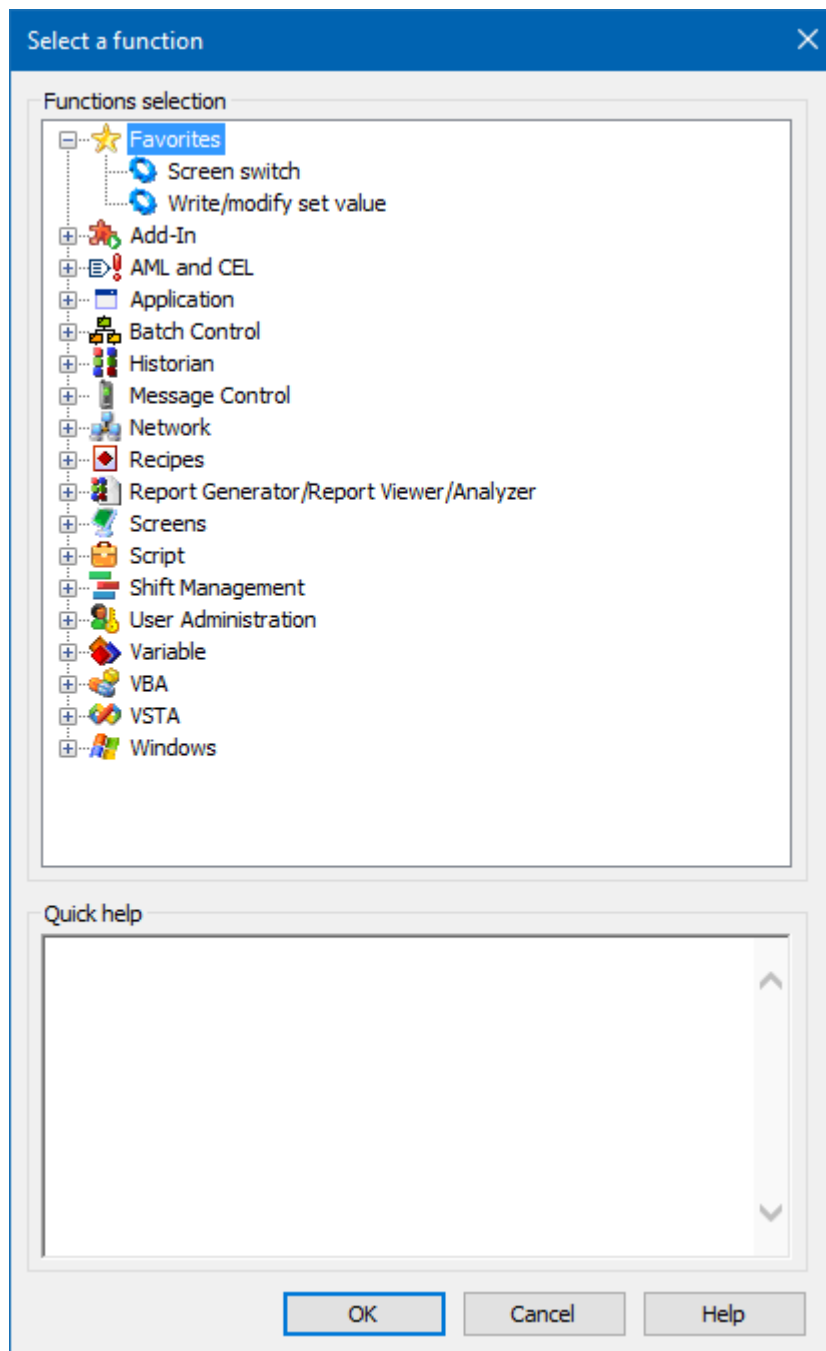
CONFIGURATION OF THE FUNCTION

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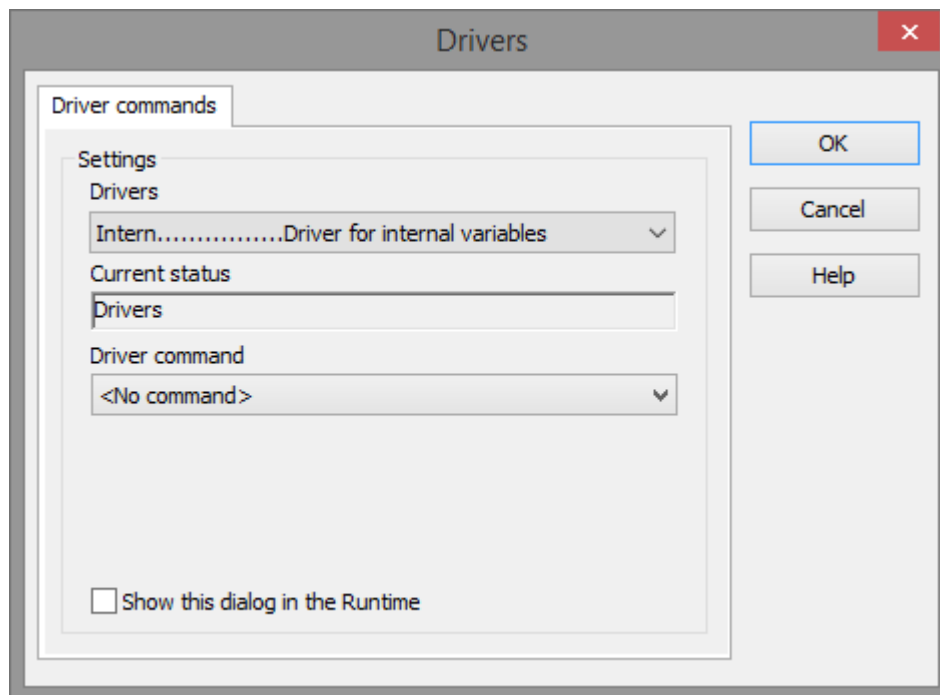To configure the function:

1. Create a new function in the zenon Editor.
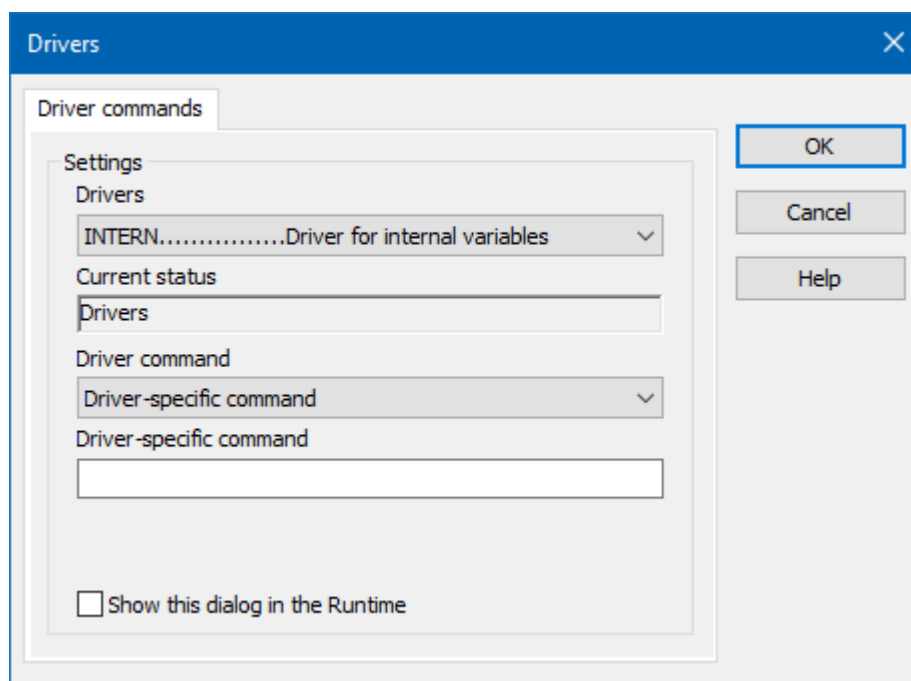
The dialog for selecting a function is opened

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.

9 Error analysis

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

9.1 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.

9.2 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.

9.3 Check list

GENERAL TROUBLESHOOTING

Check the following in the event of errors:

▸ Analysis with the **Diagnosis Viewer** (on page 42):
-> Which messages are displayed?

▸ Did you use the right object type for the variable

Example: Driver variables are purely statistics variables. They do not communicate with the PLC. (See chapter Driver variable (on page 29).)

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. You can find more information on the Diagnosis Viewer in the Diagnosis Viewer manual.

The following is required for further analysis of errors:

▶   The project backup

▶   LOG files

Send these to your support person after agreement with the customer service department.

SOME VARIABLES REPORT INVALID.

▶   Check the configuration of the source variable.

VALUES ARE NOT DISPLAYED, NUMERIC VALUES REMAIN EMPTY

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.

DRIVER FAILS OCCASIONALLY

Analysis with the **Diagnosis Viewer** (on page 42):
-> Which messages are displayed?

9.4   LOG entries

For the **ADP driver** detailed LOG entries are created.

GENERAL

Entry	Level	Description
Advise. pv:<id>(<name>) advcnt:<count>	**DEEPDEBUG** **DRV**	*Advise* of a variable ▸ *<id>:* ID of the variable ▸ *<name>:* Variable name ▸ *<count>:* Number of how often the variable was advised
Unadvise. pv:<id>(<name>) advcnt:<count>	**DEEPDEBUG** **DRV**	*Unadvise* of a variable ▸ *<id>:* ID of the variable ▸ *<name>:* Variable name ▸ *<count>:* Number of how often the variable was unadvised
Poke. pv:<id>(<name>) value:<value> cmd:<cmd> status:<status> originator:<origin>	**DEEPDEBUG** **DRV**	A poke. ▸ *<id>:* ID of the variable ▸ *<name>:* Variable name ▸ *<cmd>:* Command ID ▸ *<value>:* Value ▸ *<status>:* Status ▸ *<origin>:* triggering computer
channel:<id>(<name>) cmd:<cmd> timeout:<time> id:<cmdid>	**DEEPDEBUG** **DRV**	Sending or receiving a *Netselect* command. ▸ *<id>:* ID of the variable ▸ *<name>:*

Entry	Level	Description
		Variable name
		▶ *<cmd>:* Netselect command (*start, stop, refresh*) or answer from the driver (*ok* or *err*)
		▶ *<time>:* Timeout for netsel
		▶ *<cmdid>:* Identification for request (at refresh)
Request value	**DEEPDEBUG** **DRV**	Direct value request.
Send requested value	**DEEPDEBUG** **DRV**	Sending an answer to *Request value*.
Transfer data point value. *pv:<id>(<name>)* *value:<value> status:<status>* *time:<time>*	**DEEPDEBUG** **DRV**	Value of the alternative data point in the Runtime.
		▶ *<id>:* ID of the variable
		▶ *<name>:* Variable name
		▶ *<value>:* Value
		▶ *<status>:* Status
		▶ *<time>:* Timestamp of value change

LOADING OR RELOADING THE RUNTIME

Entry	Level	Description
ADPObject:(<ID> <<Name>>) *Usage of deleted source with* *selector number* *<<SelNumber>>.*	**WARNINGS** **DRV**	At an ADP variable a source variable is linked which was deleted.
		▶ *<ID>* Variable ID of the ADP variable (database ID).
		▶ *<Name>*

Entry	Level	Description
		name of the ADP variable ▶ *<SelNumber>* Selector number of the source variable
ADPObject:(<ID> <<Name>>) Usage of not existing source with selector number <<SelNumber>>.	**WARNINGS** **DRV**	At an ADP variable a source variable is linked which was deleted. ▶ *<ID>* Variable ID of the ADP variable. ▶ *<Name>* name of the ADP variable ▶ *<SelNumber>* Selector number of the source variable
ADPObject:(<ID> <<Name>>) Usage of deleted selector.	**WARNINGS** **DRV**	At an ADP variable a selector variable is linked which was deleted. ▶ *<ID>* Variable ID of the ADP variable. ▶ *<Name>* name of the ADP variable
ADPObject:(<ID> <<Name>>) Usage of not existing selector.	**WARNINGS** **DRV**	At an ADP variable no selector variable is linked. ▶ *<ID>* Variable ID of the ADP variable. ▶ *<Name>* name of the ADP variable

ERROR ON ADVICE OR UNADVICE OF THE VARIABLE.

Entry	Level	Description
Activate value change notification for ADP driver failed!	**Errors** **Driver**	Internal error. The **ADP driver** does not receive a value change.
ADP advises <amount> datapoint(s). Vars:<list>	**DeepDebug** **Driver**	Advise from the **ADP driver** to the connection. ▶ *<amount>* Number of signed connections

Entry	Level	Description
		▶ *<list>* List contains the first 3 variables which were advised: *„name:<variablename>(<variableId>)"*
ADP unadvises datapoint: name:<Variablenname>(<Vari ablenId>)	**DeepDebug** **Driver**	Unadvise of a variable by the **ADP driver**.
Advise datapoints for ADP failed!	**Error** **Driver**	The **ADP driver** could not be advised due to an internal error.
Advise datapoints for ADP threading violation!	**Error** **Driver**	Internal error: Advise was called from a wrong thread
Unadvise datapoints for ADP failed!	**Error** **Driver**	The **ADP driver** could not be unadvised due to an internal error.
Unadvise datapoints for ADP threading violation!	**Error** **Driver**	Internal error: Unadvise was called from a wrong thread
EntryProcessing for ADP threading violation!	**Error** **Driver**	Internal error: Value change was called from a wrong thread
channel:<pvid>(<name>) cmd:START timeout:10000 id:0; not forwarded to driver	**DeepDebug** **Driver**	For this variable a netselect already exists in the **ADP driver**. Therefore the netselect receives a negative response. *<pvid>* ID of the ADP variable. ▶ *<name>* name of the ADP variable