zenon
by COPA-DATA

zenon driver manual
EUROMAP63

v.8.20

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 EUROMAP63

GENERAL

The **EUROMAP63 driver** communicates with injection molding machines using the EUROMAP63 protocol.

- The driver supports communication to several machines using configurable connections. Alternatively, several instances of the **EUROMAP63 driver** can be configured. In this case, a separate **EUROMAP63 driver** is configured for each machine.
- Communication is carried out by means of a file-based interface.

- A folder for the communication can be configured for each connection.
  There is the choice to have the driver use relative paths or absolute paths for the files.
- A timeout can be configured for each connection. Due to the slower communication, it is recommended that this timeout is configured as somewhat longer.
- Variables can be taken directly from the machine via online import into the zenon Editor configuration.

EDITOR

Variables can be exchanged in the background by means of communication with the EUROMAP63 server. The variables can then be imported into the editor from the saved local information. The import enables the creation of standard Euromap63 variables, manufacturer-specific Euromap63 variables and general machine information.

RUNTIME

The driver supports communication of variable values in the Runtime in read direction via a report job. The report job is created in such a way that the variables are updated cyclically.

In doing so, note:

- All **State** variables are communicated in a single report job at all times.
- A **SET** job is created in write direction. Not all variables can be written.

ALARMS

For communication of alarms in the Runtime, a Boolean variable and a string variable can be created for each connection in the editor per alarm number. Addressing is carried out using the **Net address** (connection in the driver configuration) and the offset (alarm number).

The alarm text of the EUROMAP63 server is applied to a string variable in the Runtime. It contains the respective alarm numbers of the EUROMAP63 server. With a dynamic limit value text or status text in a reaction matrix, this information can be transferred to the alarm message list. To do this, configure an additional binary variable and the reaction matrix ... in Runtime.

CHANGES

A string variable can be created for each connection for the communication of events (CHANGES) in the Runtime. Addressing is carried out using the **Net address**. The events in the Runtime can be transferred from the string variable by means of a dynamic status text of a string reaction matrix and logged in the chronological event list.

UPLOAD/DOWNLOAD

One command variable per connection can be created in the editor for the sending of specific commands to the EUROMAP63 server in the Runtime.

NETWORK

Runtime reads/writes from/to the local folder and via report jobs. This is executed on both the primary server and the secondary server independently of one another.

3 Driver history

Date	Build number	Change
8/30/2019	60055	Created driver documentation

4 Requirements

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

4.1 PC

The local folder in which the EUROMAP63 server - according to the implementation of the machine - reads and writes the files must be present of the target system. The target system is the computer on which the Runtime is running.

Information

The EUROMAP63 standard does not describe how the machine reads files from this folder or writes to this folder. Machines generally use SMB (in the approved folder in the network) or FTP: FTP client on the machine and FTP server on the computer with the Runtime.

4.2 PLC

The machine must comply with the EUROMAP63 standard. In addition, it must be guaranteed that the machine can read and write files in the folder on the target system. The target system is the computer on which the Runtime is running.

In practice, this communication on the machines is carried out by means of SMB or FTP.

5 Configuration

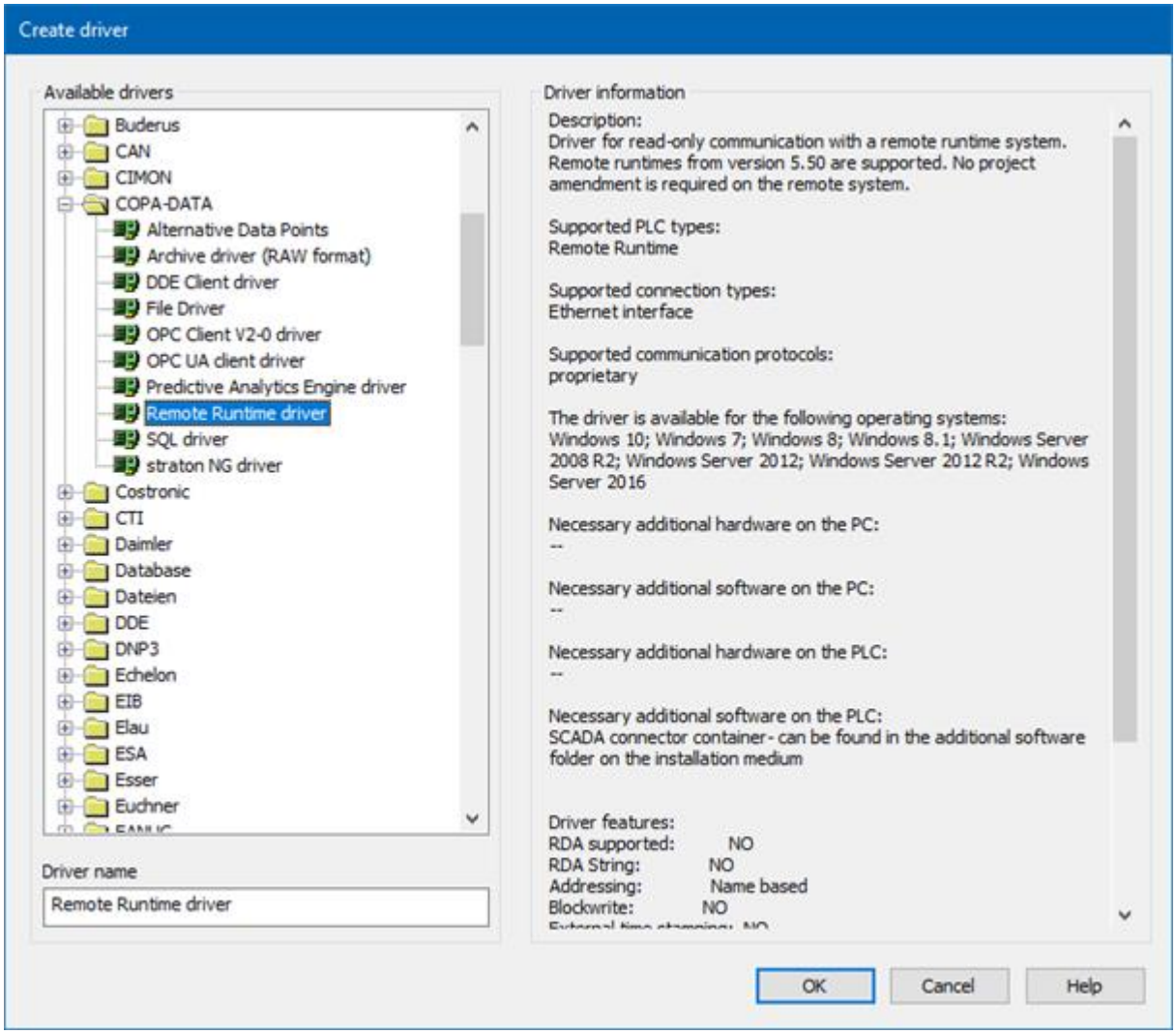
In this chapter you will learn how to use the driver in a project and which settings you can change.

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

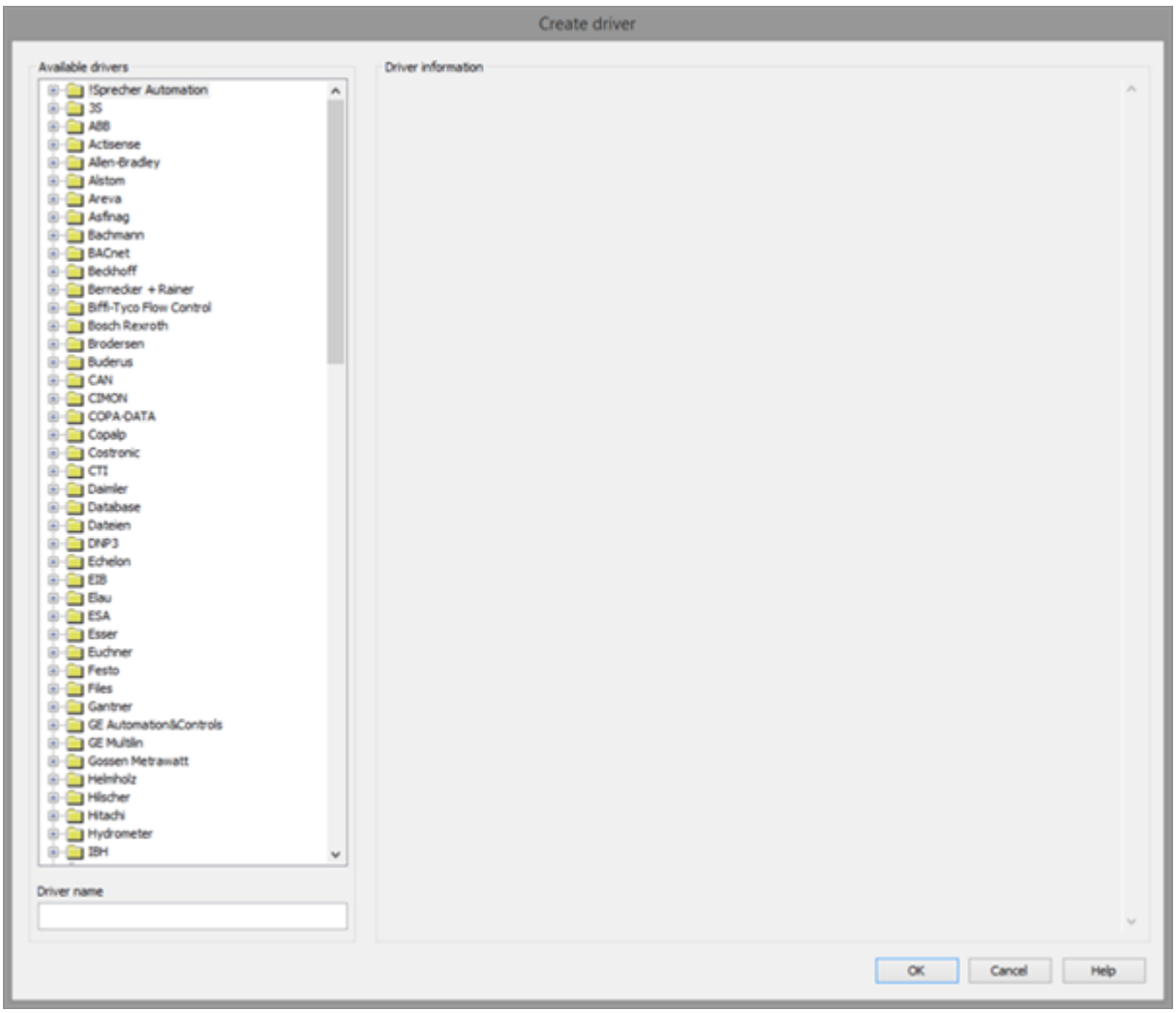
In order to create a new driver:

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**:

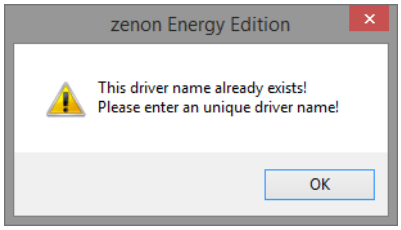
   - The **Driver name** must be unique.
   If a driver is used more than once in a project, a new name has to be given each time. This is evaluated by clicking on the **OK** button. If the driver is already present in the project, this is shown with a warning dialog.
   - The **Driver name** is part of 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**

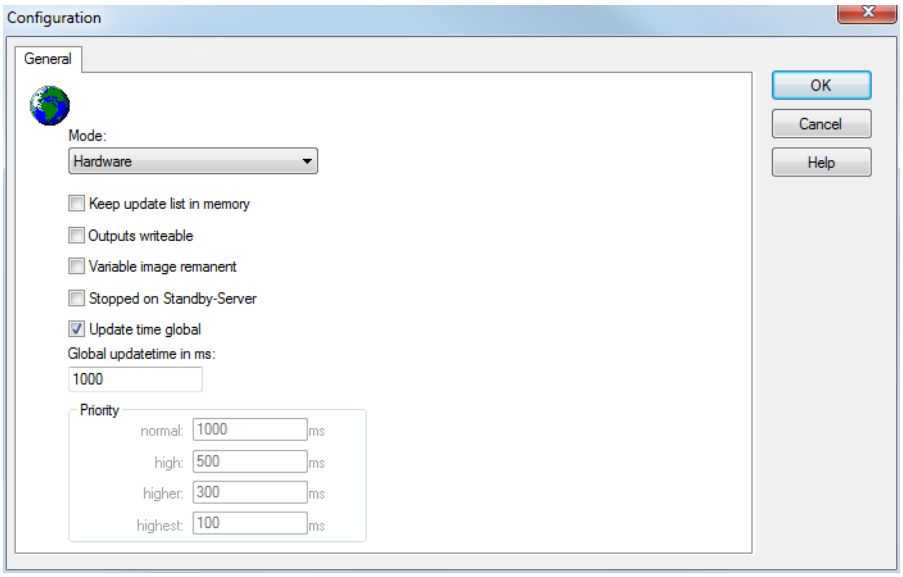
> **Information**
>
> Only the required drivers need to be present in a zenon project. Drivers can be added at a later time if required.

5.2 Settings in the driver dialog

You can change the following settings of the driver:

5.2.1 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.

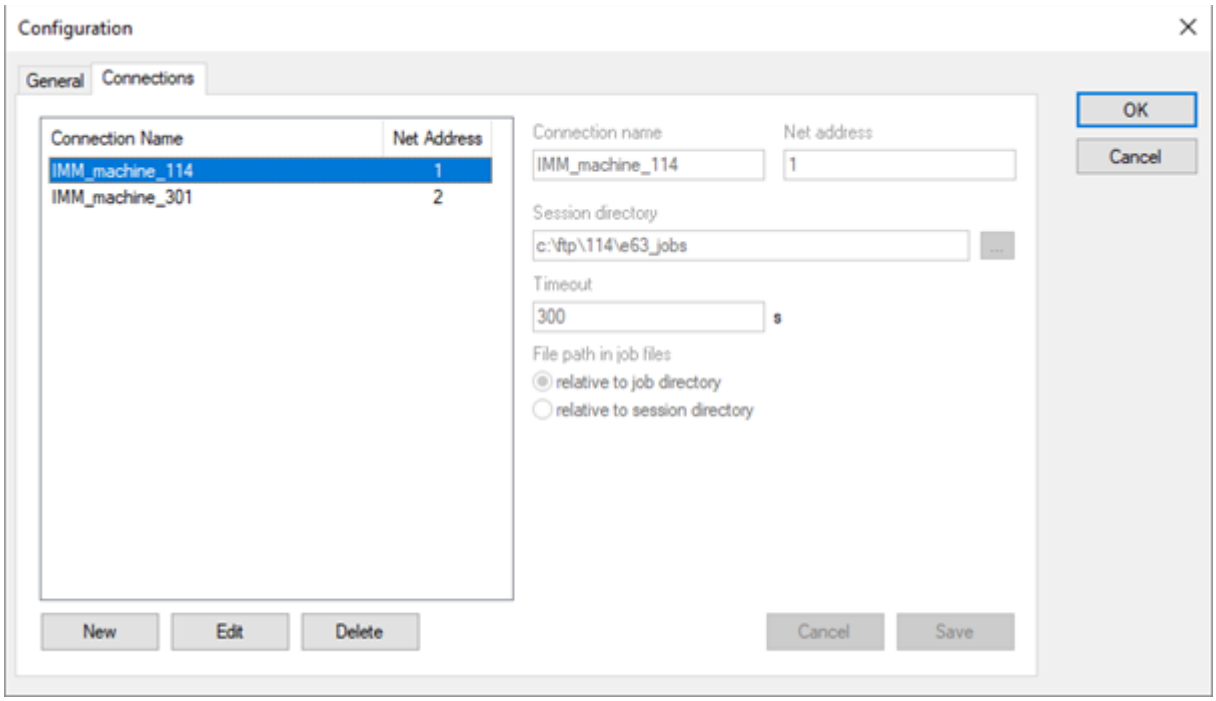
UPDATE TIME FOR SPONTANEOUS DRIVERS

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**.

5.2.2 Connections

You configure one or more connections to EUROMAP63-compatible machines in the **Connections** tab. The driver communicates by means of files in a local folder. In doing so, files for queries to the machine are created by the driver and written to the configurable storage location. In addition, the driver checks the directory for files that are created or stored by the machine.



Parameter	Description
Connections	List of configured connections. The freely-selectable names and the net address are shown. Select a connection and click on the Edit button to get to the fields of the connection settings for the configuration.
New	Creates a new connection. The connection can be configured in the corresponding input fields.
Edit	Unlocks the configuration of the selected connection or subconnection in the configuration area. Not active if there is no connection or subconnection selected in the connection list.
Delete	Deletes a selected connection from the connection list.

Parameter	Description
	Attention: The connection is deleted without requesting confirmation. Variables that have already been configured remain in zenon, but are no longer supplied with valid values. Not active if no connection is selected in the connection list.
Connection name	Path to the EUROMAP63 session directory of the machine.
Net address	Time that is waited for a response to connection queries as well as for a response for commands and for the results for commands Default: *0* The recommended value very much depends on the machine, the method with which the EUROMAP63 server communicates with the host computer and the number of variables that have been created for a machine. If a very large amount of variables have been configured for a machine, it can take up to 5 minutes until the first response for a report is received. Set the value higher if the variables in the Runtime are initially invalid.
Session directory	Path to the EUROMAP63 session directory of the machine.
Timeout	Time that is waited for a response to connection queries as well as for a response for commands and for the results for commands Default: *0* The recommended value very much depends on the machine, the method with which the EUROMAP63 server communicates with the host computer and the number of variables that have been created for a machine. If a very large amount of variables have been configured for a machine, it can take up to 5 minutes until the first response for a report is received. Set the value higher if the variables in the Runtime are initially invalid.
File path in job files	This option determines whether the driver uses the path in JOB files relative to the session directory or relative to the JOB directory. For KraussMaffei machines: "*relative to job directory*" For Engel machines: "*relative to session directory*"

Parameter	Description
Save	Saves the configuration of the connection. The current configuration is validated by clicking on the button. Errors detected are displayed accordingly.
Cancel	Discards all changes to the selected connection. No changes are saved.

CLOSE DIALOG

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

CREATE NEW CONNECTION

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

EDIT CONNECTION

1. Select the connection in the connection list.
2. Click on the **Edit** button.
3. Change the connection parameters.
4. Close by clicking on the **Save** button.

DELETE CONNECTION

1. Select the connection in the connection list.
2. Click on the **Delete** button.
3. The connection will be removed from the list

6 Creating variables

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

6.1 Creating variables in the Editor

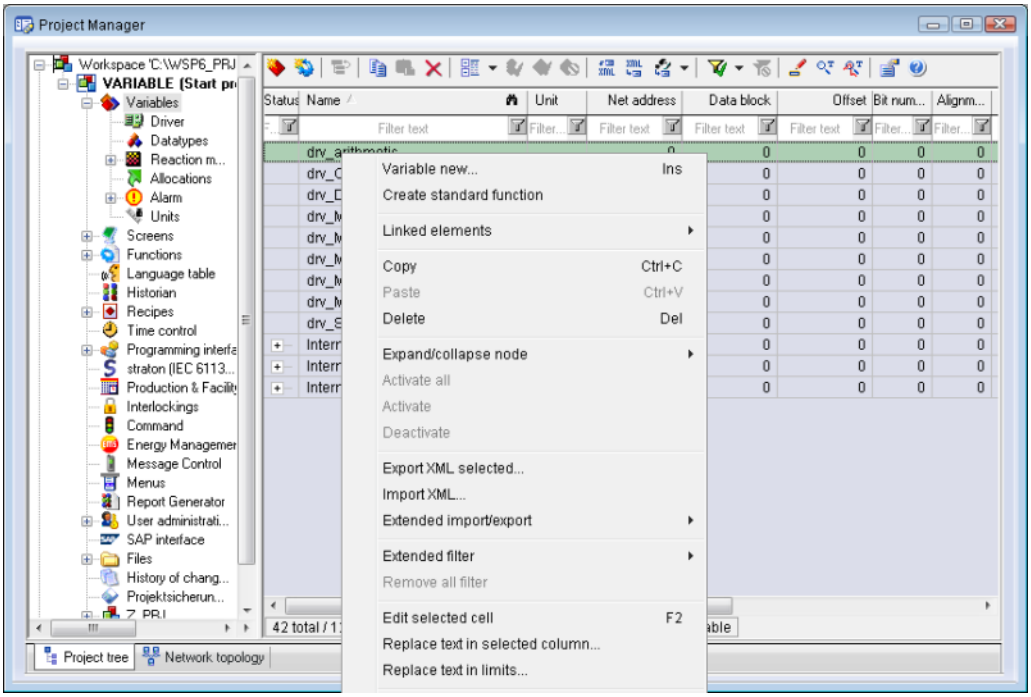
Variables can be created:

- as simple variables
- in arrays
- as structure variables

VARIABLE DIALOG

To create a new variable, regardless of which type:

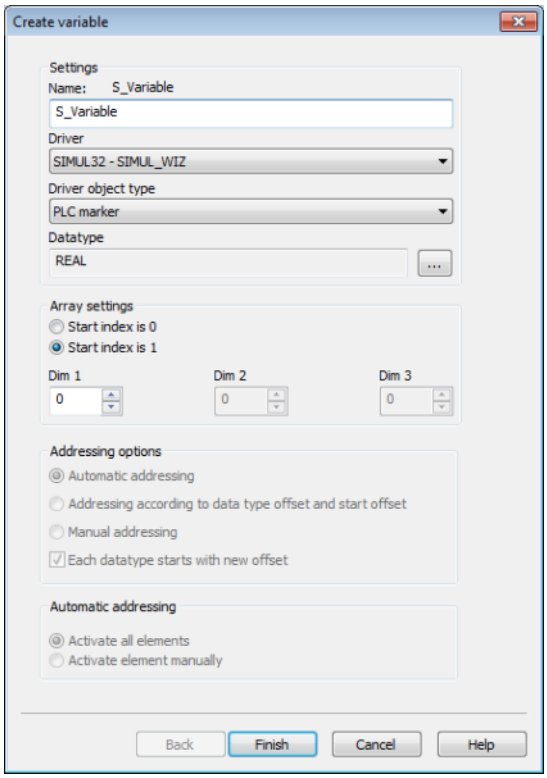
1. Select the **New variable** command in the **Variables** node in the context menu



   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	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.
Data block	not used for this driver
Offset	not used for this driver
Symbolic address	Is used for the addressing and contains the identification for the "*token parameter*". The value is normally set by the online import mechanism in the editor. "*token parameters*" that are defined in the EUROMAP63 standard do not contain a prefixed "@" character. Machine-specific *token parameters* contain a prefixed "@" character. This field is not required for the driver object types "*Changes*" and "*Command*". For the "*Alarm*" driver object type, this field must contain the manufacturer-specific alarm number or alarm identification. Two variables with the same addressing must be configured for an alarm: ▸ A *BOOL* variable as a trigger for the alarm. ▸ A *STRING* variable that contains the alarm text.
Alignment	not used for this driver

Group/Property	Description
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 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 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.

6.3.1 Driver objects

The following object types are available in this driver:

Driver Object Type	Channel type	Read	Write	Supported data types	Description
State	66	X	X	*BOOL*, *DINT*, *UDINT*, *REAL*, *LREAL*, *STRING*	Variables for "*token parameter*". Values are updated in the Runtime after receipt of "*report .DAT*" files. A machine only creates files if it is running. Variables with "*Set*" in the identification can also be written in accordance with the *EUROMAP63* standard. A "*SET Job*" is used for this.

Driver Object Type	Channel type	Read	Write	Supported data types	Description
Machine Information	65	X		*STRING*	Variables for information about a machine's hardware and software. In the Runtime, the values are updated to the *GETINFO* command after receipt of a result.
Alarm	68	X		*BOOL*, *STRING*	Variables for the communication of alarms that are generated by the machine. Addressing is via the machine-specific alarm number. This number is numeric. The values are updated in the Runtime after receipt of the *alarm .DAT* file. ▸ If an alarm is received, the *STRING* variable is initially assigned to the received alarm text. ▸ The *BOOL* variable is then set according to the alarm status. ▸ An alarm with the alarm text from the machine can be created by means of the STRING reaction matrix and dynamic limit value texts.
Changes	69	X		*STRING*	Variable for the communication of changes that are generated by the machine. Only one variable per connection can be created. The value is

Driver Object Type	Channel type	Read	Write	Supported data types	Description
					updated in the Runtime after receipt of the *changes .DAT* file. The changes to the chronological event list can be logged with the text from the machine by means of a STRING reaction matrix and dynamic limit value texts.
Command	67	X	X	*STRING*	Variable to trigger *UPLOAD* and *DOWNLOAD* commands. The response via the command is mapped to this variable.
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). **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 34) 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.

6.3.2 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
numeric	BOOL	8
-	USINT	9
-	SINT	10
-	UINT	2
-	INT	1
numeric	UDINT	4
numeric	DINT	3
-	ULINT	27
-	LINT	26
numeric	REAL	5
numeric	LREAL	6
vstring	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.

6.4 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.

6.4.1 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.

6.4.2 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

Identification	Type	Field size	Comment
			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 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

Identification	Type	Field size	Comment
			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
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.

Identification	Type	Field size	Comment
			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.
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

Identification	Type	Field size	Comment
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 (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.

6.4.3 Online import

The **EUROMAP63 driver** supports the import of variables from the machine.

1. Configure a valid connection in the driver configuration (on page 16).
2. Read the data from the machine in the background. You can select one or more connections from the driver configuration.

   In this step, the driver in the editor communicates with the machine and uses the commands *GETID* and *GETINFO*. If communication is successful, the driver saves the information for each connection in a *.plccache* file. This file is a component of the project.

3. Use the online variable import functionality to import variables from the *.plccache* file. As long as the driver configuration and the machine configuration do not change, you can repeatedly import variables from the *.plccache* file and do not need to communicate with the machine again to do this. The *.plccache* file is saved as part of the project.

6.5 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.

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

Name from import	Type	Offset	Description
			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 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

Name from import	Type	Offset	Description
			PhoneNumberSet and **ModemHwAdrSet**.
PhoneNumberSet	*STRING*	38	Telephone number, that should be used
ModemHwAdrSet	*DINT*	39	Hardware address for the telephone number
GlobalUpdate	*UDINT*	3	Update time in milliseconds (ms).
BGlobalUpdaten	*BOOL*	4	TRUE, if update time is global
TreiberSimul	*BOOL*	5	TRUE, if driver in sin simulation mode
TreiberProzab	*BOOL*	6	TRUE, if the variables update list should be kept in the memory
ModemActive	*BOOL*	7	TRUE, if the modem is active for the driver
Device	*STRING*	8	Name of the serial interface or name of the modem
ComPort	*UINT*	9	Number of the serial interface.
Baudrate	*UDINT*	10	Baud rate of the serial interface.
Parity	*SINT*	11	Parity of the serial interface
ByteSize	*USINT*	14	Number of bits per character of the serial interface Value = *0* if the driver cannot establish any serial connection.
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

Name from import	Type	Offset	Description
			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).

Name from import	Type	Offset	Description
PokeFinish	*BOOL*	55	Goes to *1* for a query, if all current pokes were executed

ERROR MESSAGE

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

ALARMS

The **EUROMAP63 Treiber** supports communication of alarms.

- The manufacturer-specific alarm detection is used for addressing: firstly for a BOOL variable for the alarm status and secondly for a STRING variable for the alarm text received from the machine.
- If an alarm is received from the machine, the driver first writes the alarm text to the string variable and then the alarm status to the BOOL variable. An alarm can be generated from a limit value for the BOOL variable with dynamic limit value text from the corresponding STRING variable. In doing so, the alarm text from the machine is applied. Alternatively, a reaction matrix with dynamic limit value text can also be used.
- If only the alarm status is to be displayed in the alarm list, a BOOL variable with static limit value text is sufficient.
- If only the alarm text is to be displayed in the chronological event list, this can be implemented with a STRING variable in combination with a STRING reaction matrix with dynamic text.
- The driver determines the current alarms when the Runtime is started. The pending alarms are then sent to the Runtime. **Attention:** Historic alarms that have been recorded whilst the Runtime was not running are not transferred.

CHANGES

Only one variable per connection of the "**Changes**" driver object type can be created. In the Runtime, the value is initialized with an empty string, so that values from the machine are detected by a STRING reaction matrix accordingly and can be logged.

SET

The writing of "**State**" variables is supported for some variables. The driver uses the **SET** command for this. In general, the direct value change of a token parameter is logged on the machine and also communicated as a change via the **Changes** variable.

UPLOAD AND DOWNLOAD

The driver supports the **UPLOAD** of manufacturer-specific machine datasets to the computer with the Runtime and **DOWNLOAD** of manufacturer-specific machine datasets from the computer with the Runtime on the machine. Depending on the manufacturer or machine type, the datasets are an individual file or several files.

In order to start **UPLOAD** or **DOWNLOAD**, a "**Command**" STRING variable (string length 255 characters) is required. A single string variable with corresponding net address can be created manually for each connection. The result of the execution of the command is mirrored to the string variable.

- Load the current dataset from the machine:

  UPLOAD <Target name for the dataset on the local computer> ACTIVE

  Example: *UPLOAD "MACHINE1" ACTIVE*
- Load an archived dataset from the machine:

  UPLOAD <Target name for the dataset on the local computer> <Name of the archived data set>

  Example: *UPLOAD "MACHINE1_ARCHIVE1" "ARCHIVE1"*
- Load a local data set into the current data set of the machine:

  DOWNLOAD <Name of the data set on the local computer> ACTIVE

  Example: *DOWNLOAD "ARCHIVE1" ACTIVE*
- Load a local data set into an archived data set on the machine:

  DOWNLOAD <Name of the data set on the local computer> <Target name of the archived data set>

  Example: *DOWNLOAD "ARCHIVE1" "NEWDATASET1"*

A *DOWNLOAD* is under certain circumstances only possible if the machine is in *semi-automatic* or *manual operation* state.

The progress and the result of **UPLOAD** and **DOWNLOAD** commands is mapped to the **Command** driver object type variables:

- *0* - command successful
- *1* - command is executed
- *2* - a command is already being executed
- *3* - machine error when executing the command
- *4* - invalid command
- *5* - command variable with invalid net address
- *6* - invalid parameter or invalid number of parameters

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.

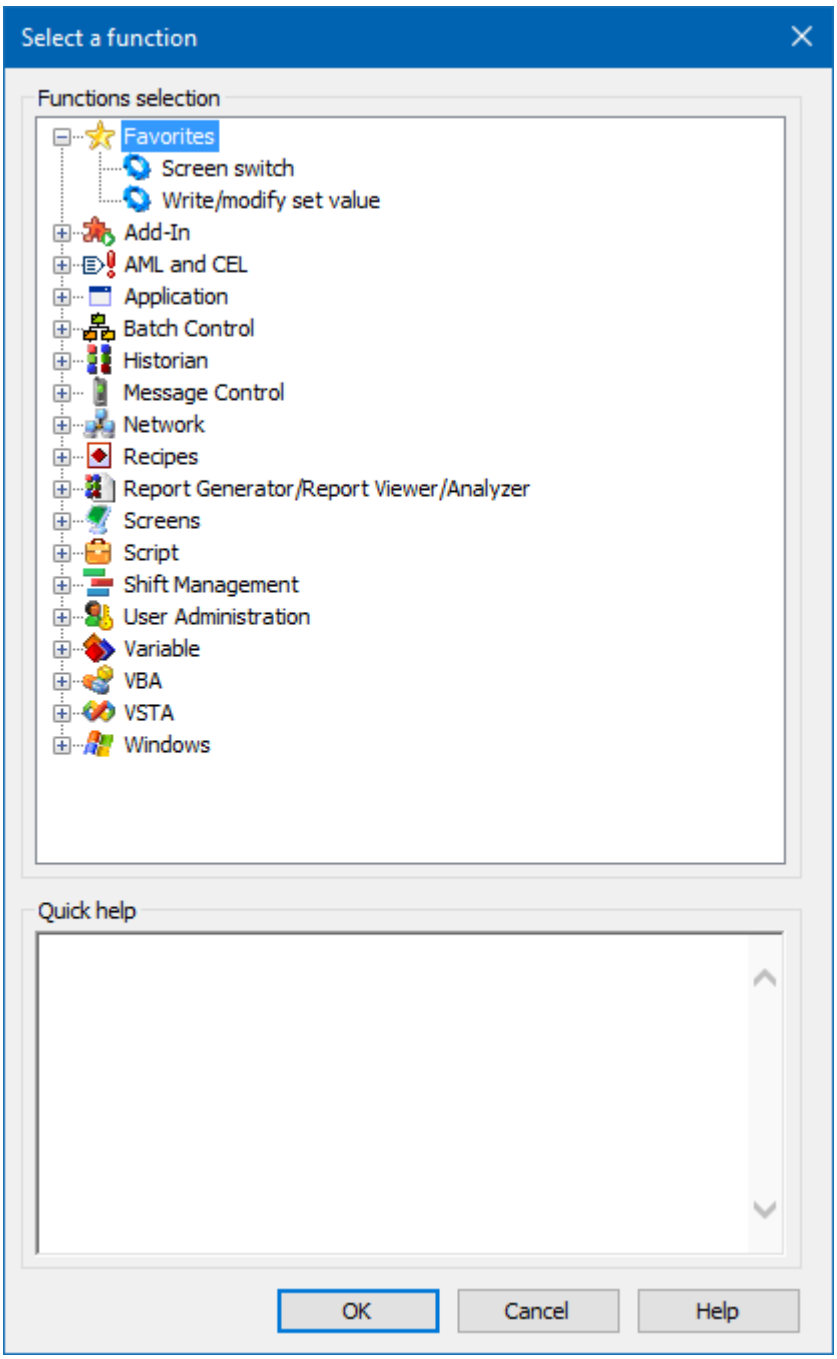
> **⚠Attention**
>
> The zenon **Driver commands** function is not identical to driver commands that can be executed in the Runtime with Energy drivers!

CONFIGURATION OF THE FUNCTION

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

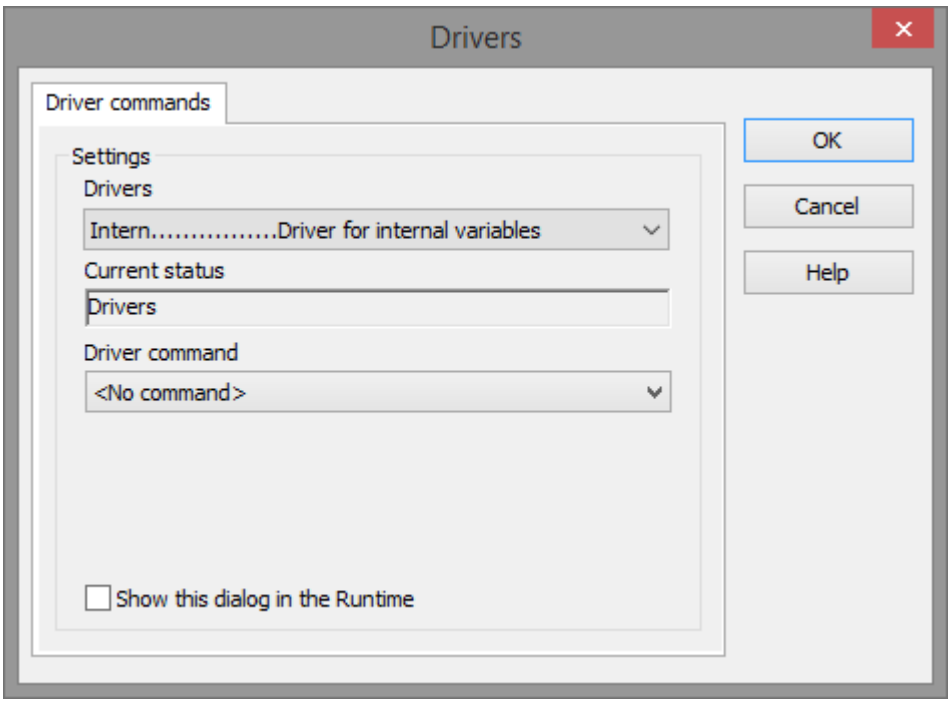
1. Create a new function in the zenon Editor.

The dialog for selecting a function is opened



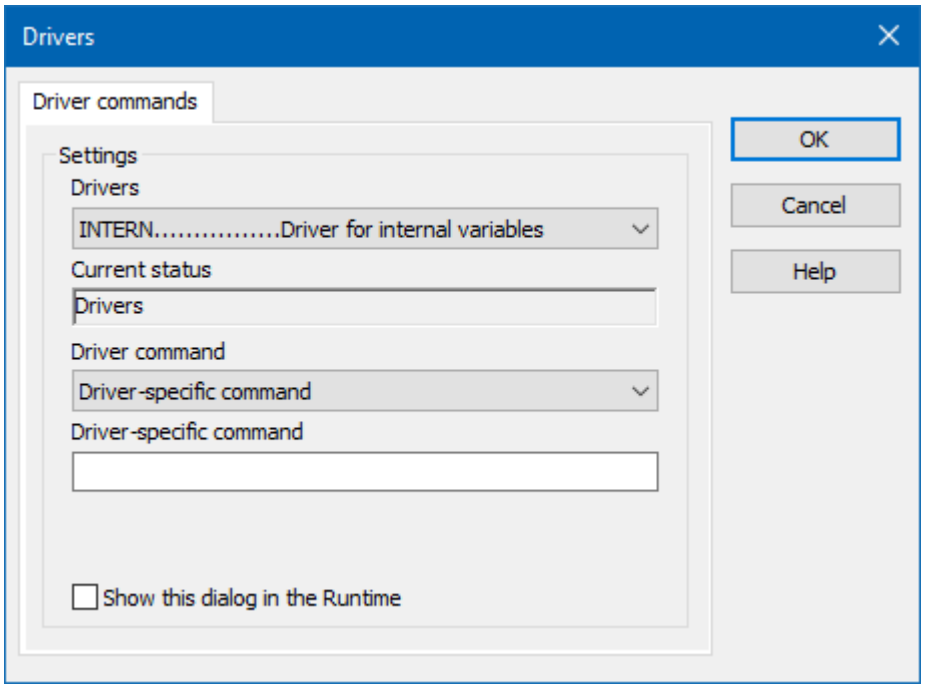
2. Navigate to the node **Variable.**
3. Select the **Driver commands** entry.

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:

- Is the PLC connected to the power supply?
- Analysis with the **Diagnosis Viewer** (on page 47):
  -> 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 Enter.**

  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 Open, telent <IP address port number> (e.g. for Modbus: telnet 192.168.0.100 502) -> Press Return key.**

If the monitor display turns black, a connection could be established.

- 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 are purely statistics variables. They do not communicate with the PLC. (See chapter Driver variable (on page 34).)
- Does the offset addressing of the variable match the one in the PLC?

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.

- INVALID bits always refer to a net address.
- At least one variable of the net address is faulty.

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 47):
-> Which messages are displayed?