COPADATA
do it your way

zenon driver manual

S7TCP32

v.8.00

COPA·DATA
do it your way

Contents

1. Welcome to COPA-DATA help

ZENON VIDEO-TUTORIALS

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

GENERAL HELP

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

PROJECT SUPPORT

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

LICENSES AND MODULES

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

2. S7TCP32

Driver for S7 TCP/IP connection using standard network card without additional software.

The driver also supports Simatic PDiag error messages.

3. S7TCP32 - Data sheet

General:	
Driver file name	S7TCP32.exe
Driver name	S7 TCP-IP
PLC types	Siemens S7 200, 300, 400 and S71200, S7 1500 or VIPA 200V, 300V, 300S and 500S
PLC manufacturer	Inat; Siemens; Vipa; Process-Informatik;

Driver supports:	
Protocol	TCP/IP - RFC1006;
Addressing: Address-based	X
Addressing: Name-based	--
Spontaneous communication	X
Polling communication	X
Online browsing	--
Offline browsing	X
Real-time capable	--
Blockwrite	X
Modem capable	--
Serial logging	--
RDA numerical	X
RDA String	--
Hysteresis	X
extended API	X
Supports status bit **WR-SUC**	X
alternative IP address	X

Requirements:	
Hardware PC	Standard network card
Software PC	No additional Siemens communication software necessary
Hardware PLC	Siemens: CP 343-1 or. CP 443-1; INAT: S7 Ethernet Adapter; Vipa: CP 443; Process-Informatik: S7-LAN Adapter (MPI on TCP/IP Gateway); Helmholz NetLink PRO;Simatic CPU S7 31x PN/DP (Ethernet onboard)
Software PLC	--
Requires v-dll	--

Platforms:	
Operating systems	Windows CE 6.0, Embedded Compact 7; Windows 7, 8, 8.1, 10, Server 2008R2, Server 2012, Server 2012R2, Server 2016;
CE platforms	x86; ARM;

4. Driver history

Date	Driver version	Change
7/7/2008	5900	Created driver documentation
4/14/2015		Addressing by means of host name or IP address

DRIVER VERSIONING

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

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

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

5. Requirements

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

5.1 PC

This driver supports a connection via the standard network card of the PC. Make sure that the PLC and the PC are in the same network range and that the subnet masks are set accordingly on both devices.

5.2 PLC

The driver uses S7 communication via the TCP/IP transport protocol.

An Ethernet interface with ISO protocol RFC 1006 is required at the PLC (for so-called open IE communication). The communication interface should support these communication services (S7 communication and TCP/IP).
Here are some examples for this:

- ► Siemens: S7   CP 343-1, CP 443-1, CP243-1, S7 CPU 31x PN/DP

- ► Vipa: CP443; Speed7 PLC

- ► Helmholz NetLink PRO

- ► INAT: S7 Ethernet Adapter;'ECHOLINK' Serial Ethernet converter

- ► Process informatics: S7–LAN Adapter (MPI to TCP/IP converter)

TIA SETTINGS FOR COMMUNICATION WITH S7 1511

The following settings are necessary for communication with an S7 1511 PLC:

TIA:

▸   *PLC -> General -> Protection*:
    Allow access by remote partner via PUT/GET communication: `active`

▸   *PLC -> Data block -> General -> Attributes*:
    Optimized block access: `Inactive`

zenon Driver, TCP/IP connection (on page 19) tab:

▸   **Remote TSAP**: `02.01`

The controller expects direct addresses, no **Symbolic address**.

6.   Configuration

In this chapter you will learn how to use the driver in a project and which settings you can change.
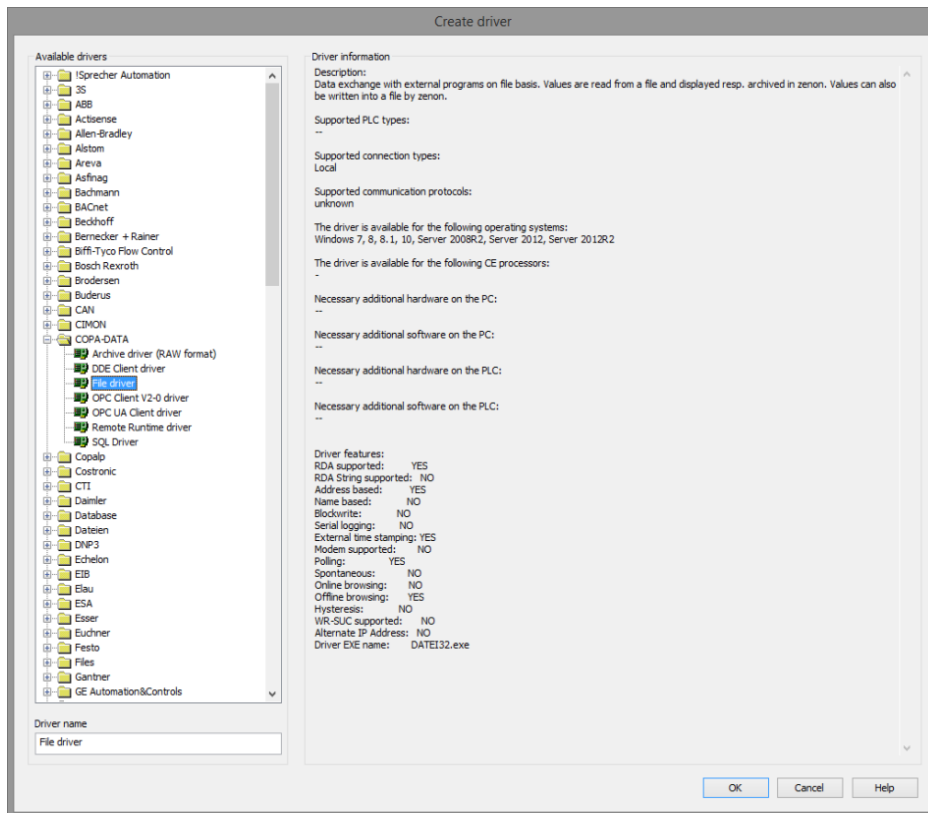
> 💡   **Information**
>
> *Find out more about further settings for zenon variables in the chapter Variables (main.chm::/15247.htm) of the online manual.*

6.1 Creating a driver

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

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

CLOSE DIALOG

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

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

CREATE NEW DRIVER

In order to create a new driver:

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

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

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

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

- The **Driver name** is part of the file name.
Therefore it may only contain characters which are supported by the operating system.
Invalid characters are replaced by an underscore (_).

- Attention: This name cannot be changed later on.

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

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

DRIVER NAME **DIALOG ALREADY EXISTS**

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

ZENON PROJECT

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

▸ **Intern**

▸ **MathDr32**

▸ **SysDrv**

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

6.2   Settings in the driver dialog

You can change the following settings of the driver:

6.2.1   General

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

Option	Description
Mode	Allows to switch between hardware mode and simulation mode ▸ `Hardware:` A connection to the control is established. ▸ Simulation - static: No communication between to the control is established, the values are simulated by the driver. In this modus the values remain constant or the variables keep the values which were set by zenon Logic. Each variable has its own memory area. E.g. two variables of the type marker with offset 79 can have different values in the Runtime and do not influence each other. Exception: The simulator driver. ▸ `Simulation - counting:` No communication between to the control is established, the values are simulated by the driver. In this modus the driver increments the values within a value range automatically. ▸ `Simulation - programmed:` No communication is established to the PLC. The values are calculated by a freely programmable simulation project. The simulation project is created with the help of the zenon Logic Workbench and runs in a zenon Logic Runtime which is integrated in the driver. For details see chapter Driver simulation (main.chm::/25206.htm).
Keep update list in the memory	Variables which were requested once are still requested from the control even if they are currently not needed. This has the advantage that e.g. multiple screen switches after the screen was opened for the first time are executed faster because the variables need not be requested again. The disadvantage is a higher load for the communication to the control.
Output can be written	▸ `Active:` Outputs can be written. ▸ `Inactive:` Writing of outputs is prevented. Note: Not available for every driver.
Variable image remanent	This option saves and restores the current value, time stamp and the states of a data point. Fundamental requirement: The variable must have a valid value and time stamp. The variable image is saved in mode hardware if:

	▶ one of the states S_MERKER_1(0) up to S_MERKER8(7), REVISION(9), AUS(20) or ERSATZWERT(27) is active The variable image is always saved if: ▶ the variable is of the object type **Driver variable** ▶ the driver runs in simulation mode. (not programmed simulation) The following states are not restored at the start of the Runtime: ▶ `SELECT(8)` ▶ `WR-ACK(40)` ▶ `WR-SUC(41)` The mode **Simulation - programmed** at the driver start is not a criterion in order to restore the remanent variable image.

Stop on Standby Server	Setting for redundancy at drivers which allow only one communication connection. For this the driver is stopped at the Standby Server and only started at the upgrade.
	Attention: If this option is active, the gapless archiving is no longer guaranteed.
	▸ `Active:` Sets the driver at the not-process-leading Server automatically in a stop-like state. In contrast to stopping via driver command, the variable does not receive status **switched off** (**statusverarbeitung.chm::/24150.htm**) but an empty value. This prevents that at the upgrade to the Server irrelevant values are created in the AML, CEL and Historian.
	Default: `Inactive`
	Note: Not available if the CE terminal serves as a data server. You can find further information in the zenon Operator manual in the CE terminal as a data server chapter.
Global Update time	Setting for the global update times in milliseconds:
	▸ `Active:` The set **Global update time** is used for all variables in the project. The priority set at the variables is not used.
	▸ `Inactive:` The set priorities are used for the individual variables.
	Exceptions: Spontaneous drivers ignore this option. They generally use the shortest possible update time. For details, see the **Spontaneous driver update time** section.
Priority	The polling times for the individual priority classes are set here. All variables with the according priority are polled in the set time.
	The variables are allocated separately in the settings of the variable properties. The communication of the individual variables can be graded according to importance or required topicality using the priority classes. Thus the communication load is distributed better.
	Attention: Priority classes are not supported by each driver, e.g. spontaneously communicating zenon drivers.

CLOSE DIALOG

Option	Description
OK	Applies all changes in all tabs and closes the dialog.

Cancel	Discards all changes in all tabs and closes the dialog.
Help	Opens online help.

UPDATE TIME FOR SPONTANEOUS DRIVERS

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

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

6.2.2   S7-TCP

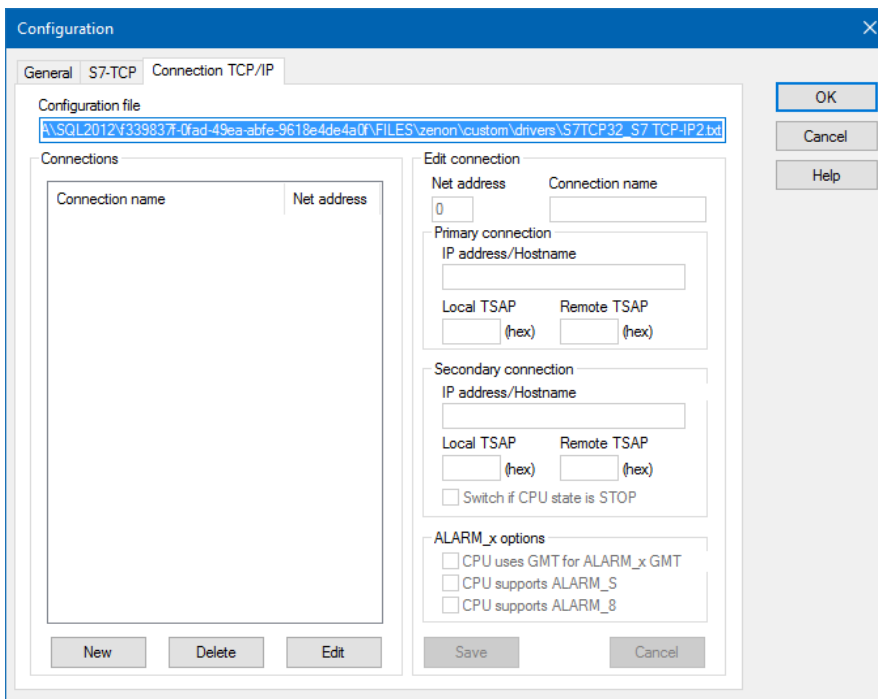| Parameter	Description
Configuration file	The configuration file must be in the current project directory. The file name can be freely defined.
	Default: `S7TCP32_S7 TCP-IP driver.txt`
	Note: The proposed name of the file consists of the driver names and their description.
	A change in this input field also has effects on the list of connections and its configuration in the TCP/IP connection tab.
Delay after connection termination (ms)	If the connection fails, the driver will take the set amount of time between re-attempts to establish communication.
	Default: `20000 ms`
Char string without header	*The type of String variables in the PLC* *S7 strings with or without header information:*
	▸ `Inactive`: STRING
	▸ `Active`: ARRAY (CHAR)
	Default: `Inactive`

6.2.3   Connection TCP/IP

Parameter	Description
Configuration file	The configuration file must be in the current project directory. The file name can be freely defined. Default: `S7TCP32_S7 TCP-IP driver.txt` Note: The proposed name of the file consists of the driver names and their description. A change in this input field also has effects on the list of connections and its configuration in the TCP/IP connection tab.

CONNECTIONS

Configuration of the connections.

Parameter	Description
Connection list	List with all configured connections. Displays the connection names with the corresponding Net addresses. The connection parameters are displayed when the connection name is selected.
New	Creates a new connection. The connection can be configured in the Edit connection section.
Delete	The selected connection will be deleted from the list without requesting confirmation.
Edit	Unlocks the configuration of the selected connection in the **edit connection** area.

EDIT CONNECTION

Settings for a selected connection. If there is no connection selected in the connection list, this area is grayed out.

Parameter	Description
Net address	Corresponds to the **Net address** property in variable configuration.
	Default: `0`
	A unique net address must be issued foe each connection. The uniqueness is validated by clicking on the **Save** button.
Connection name	Freely definable name.
	Default: `Default name`
Primary connection	Configuration of the primary connection.
	When Runtime is started, the driver first attempts to establish a connection to the PLC using this address. If this connection fails, a substitute connection - if configured - is established using the secondary connection.
	Note: The entry is checked by clicking on the **Save** button.
IP address/host name	Addressing of the primary connection to the PLC via IP address or host name.
	Depends on the settings of the S7 TCP Runtime.
	Communication is performed via port 102.
Local TSAP	TSAP for this station. It consists of two groups (bytes). Each group is built from two hexadecimal characters, and the two groups are separated by a blank or a dot.
	▸ First group: Can contain a device identification
	▸ Second group: is always 00
	Default: `01.00`
	Recommended setting: `01.00`
	Example: 01.00 = PD communicates directly with the connected SIMATIC components
Remote TSAP	TSAP for the partner station (S7 CPU). It consists of two groups (bytes). Each group is built from two hexadecimal characters, and the two groups are separated by a blank or a dot.
	▸ First group: Contains a device identification, for which resources are reserved in the SIMATIC-S7. Possible device identifications: 01 = PD 02 = OM (Operating & Monitoring) 03 = Other
	▸ Second group: Contains the addressing of the SIMATIC station, with which communication should be established. Divided into: (Bit 7...5) = Rack (subsystem)

(Bit 4...0) = CPU slot

Attention: Not the communication processor slot, but the CPU on which the PLC program also runs. Usually: Slot 2.

Default: `02.02`

Sample configuration:

OS communicates via the SIMATIC with the assembly group in rack 2, slot 3.
Help rule for rack/slot group:
Left character = rack * 2
Right character = slot

Special case:

If the device connected to the net is addressed directly, the group contains 00. The remote TSAP can be read directly in the Hardware Manager. (avoids misinterpretations due to the writing on the device itself.)

Communication with S7 200:

In order to be able to also use the S7TCP driver for the S7 200 PLCs (via CP243), a configured connection must be created in the PLC. (this is possible using "MicroWIN" configuration software). The TSAP settings in the driver must then be selected according to this connection.

For CPUs of the company Vipa:
Speed 7 CPU 315 2AG10:

RemoteTSAP: `02.02`

Communication with series 1200 and 1500 S7:

RemoteTSAP: `02.01`

Note the TIA settings (on page 8) too!

Secondary connection	Alternative connection parameters if primary configuration does not work.
IP address/host name	Addressing of the secondary connection to the PLC by means of IP address or host name.
	If this field has been completed, the driver attempts to connect to this address after each failed attempt to establish a connection. The connection to this alternative address remains until Runtime is restarted or the secondary address can no longer be reached.
	Example: For a network with redundancy with two communication processors in one controller and two network cards in the zenon computer.
	Note: The entry is checked by clicking on the **Save** button.
Local TSAP	Alternative TSAP local.
	Configuration the same as for the primary connection.
Remote TSAP	Alternative TSAP remote.
	Configuration the same as for the primary connection.
Switch if CPU status is STOP	Checkbox for a reaction to CPU `STOP` status:
	▶ `Active:` As soon as the CPU has the status `STOP`, a switch to the alternative connection is made.
	▶ `Inactive:` With the CPU STOP status, there is no switch to an alternative connection.
	Default: `Inactive`
Alarm_x options	Settings for `Alarm_S` and `ALARM_8`
CPU uses, for ALARM_x GMT	Checkbox for a possible activation of the time in GMT:
	▶ `Active:` The transferred time of the `ALARM_S` or `ALARM_8` object is considered as GMT. Activate this option if the PLC sends the time stamp of the ALARM-S/8 message GMT format.
	▶ Inactive: The transferred time of the `ALARM_S` or `ALARM_8` object is not additionally converted.
	Default: `Inactive`
CPU supports ALARM_S	Checkbox for a possible activation of `ALARM_S`:
	▶ `Active:` CPU supports ALARM_S.
	▶ `Inactive:` no support of ALARM_S.

	Default: `active`
	For more details see chapter Driver-specific functions (on page 50).
CPU supports ALARM_8	Checkbox for a possible activation of `ALARM_S`:
	▶ `Active:` CPU supports `ALARM_8`.
	▶ `Inactive:` no support of `ALARM_8`.
	Default: `Inactive`
	For more details see chapter Driver-specific functions (on page 50).
Save	Saves the configuration of the connection.
Cancel	Discards all changes to the selected connection. No changes are saved. The **Edit connection** area is deactivated.

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.

CREATE NEW CONNECTION

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

EDIT CONNECTION

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

DELETE CONNECTION

1. select the connection in the connection list

2. click on the button **Delete**

3. the connection will be removed from the list

SHOW CONNECTION DETAILS

Highlight the desired connection in the connection list.

Validation of connection configuration

Inputs for the **IP address/host name** are validated when a connection is configured. The validation is carried out by clicking on the **Save** button.

Validation rules:

▶ The properties of the primary connection must not be empty.
Error message: `Primary IP address must not be empty!`

▶ The following characters are not permitted for input: `{}|&~![]()\"';=#`
Error message: `The input for [Primary/Secondary] IP address contains invalid characters!`

▶ If there are only figures and period(s) present in the entry, a valid IP address is expected for validation.

▶ The following is applicable for the input of IP addresses:

- The format of the IP address entered must correspond to the standard:
There must be precisely 4 numerical fields included in the entry.
These fields must be separated by a period.
Error message: `The input for [Primary/Secondary] IP address contains an invalid number of address fields!`

- No field value can be greater that 255
Error message: `The input for [Primary/Secondary] IP address contains an invalid address field!`

VALIDATION - ERROR DIALOG

Note: This dialog is only available in English.

The buttons are displayed in the system language of the computer.

Missing addressing:
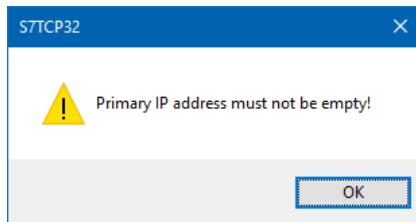
Invalid characters
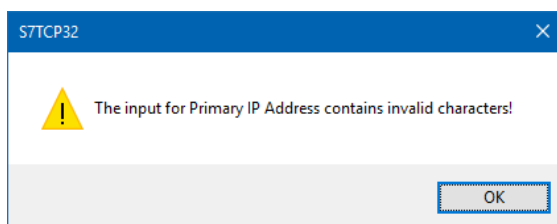
Invalid IP address format
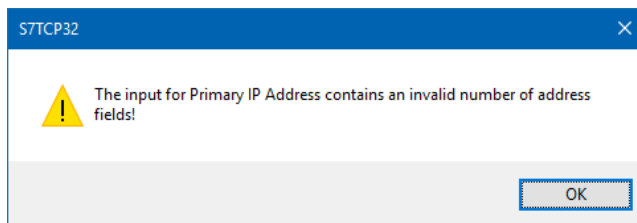
IP ADDRESS higher than 255
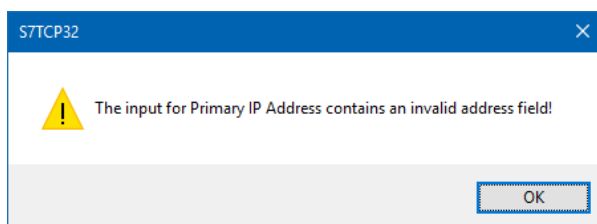
Configuration of Helmholz NETLinkPRO adapter

To create a connection for a **Helmholz NETLinkPRO** adapter:

1. Click on the **New** button in the configuration dialog in the **TCP/IP connection** tab

2. Enter the **Network address** and **Name**

3. Enter, under `Remote IP`, the IP address of the **NETLinkPRO**

4. Enter `01.00` for **Local TSAP**

5. Enter `01.00` for **Local TSAP**

6. Click on **Save**

7. Configure further properties as required

8. click on **OK**
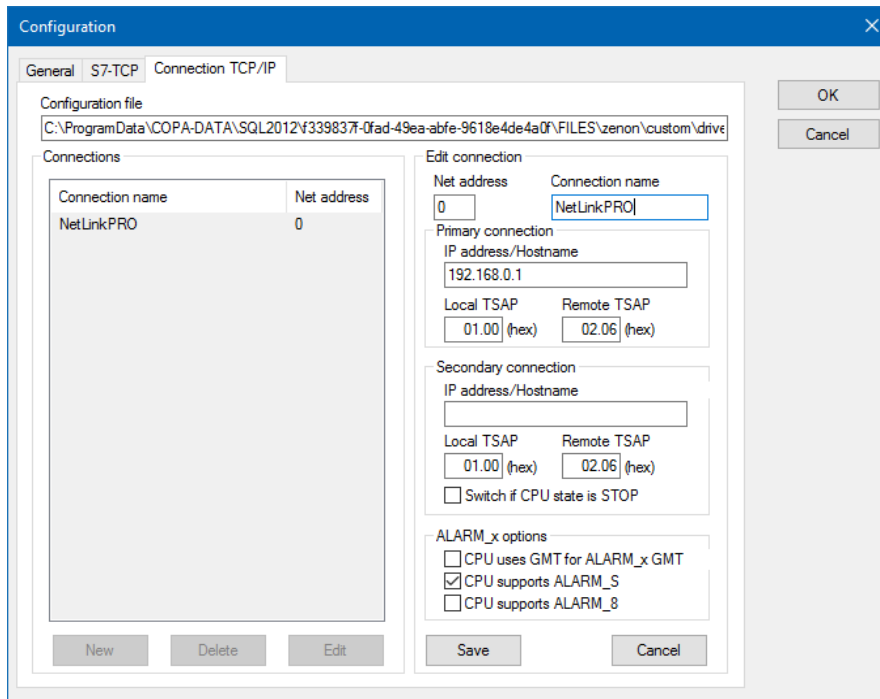
💡 Information

*You can find a detailed description of the input fields in the **TCP/IP connection** (on page 19) chapter.*

7. Creating variables

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

7.1 Creating variables in the Editor

Variables can be created:

▸ as simple variables

▸ in arrays (main.chm::/15262.htm)

▶ as structure variables (main.chm::/15278.htm)

VARIABLE DIALOG

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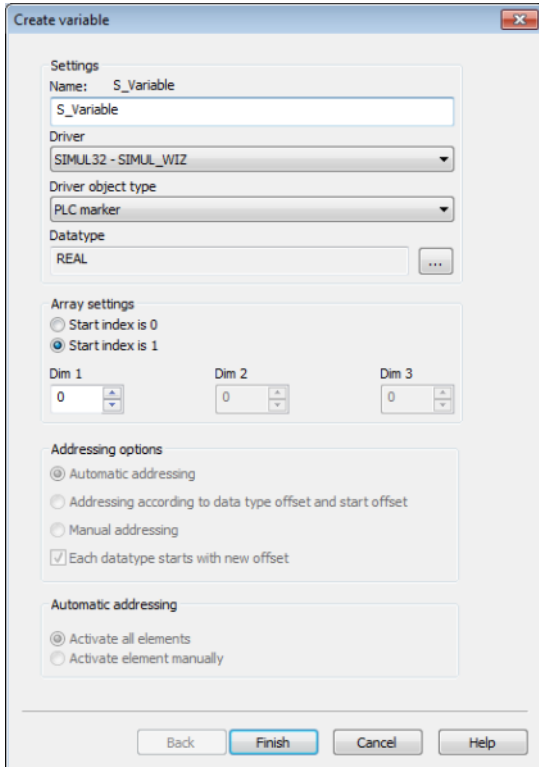
1. Select the **New variable** command in the **Variables** node in the context menu
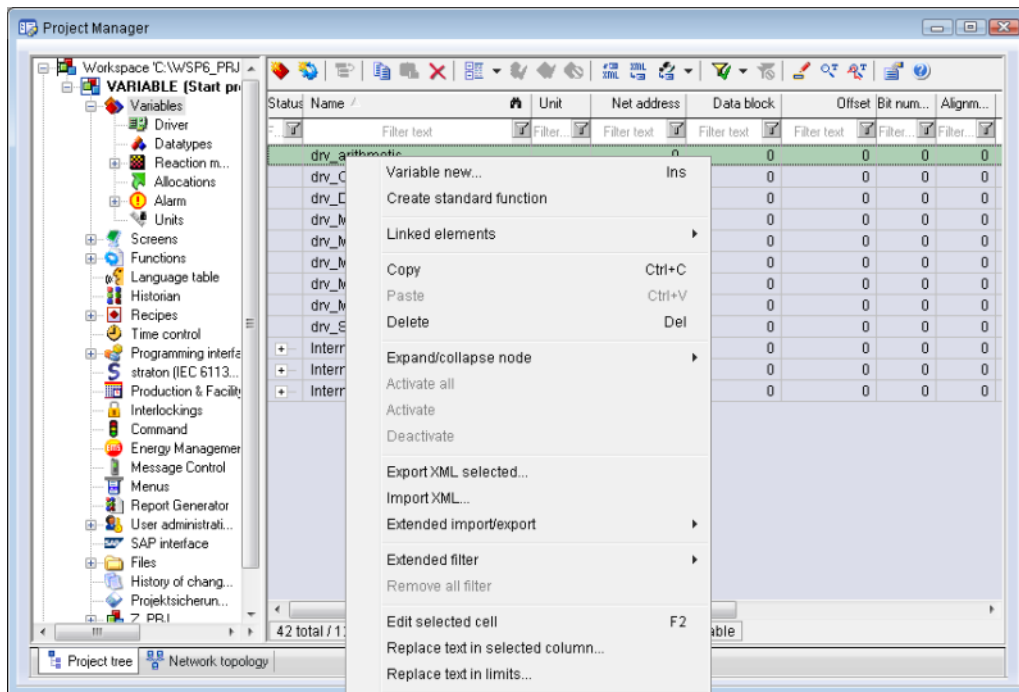
The dialog for configuring variables is opened

2. Configure the variable

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

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

Data Type	Select the desired data type. Click on the ... button to open the selection dialog.
Array settings	Expanded settings for array variables. You can find details in the Arrays chapter.
Addressing options	Expanded settings for arrays and structure variables. You can find details in the respective section.
Automatic 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.

INHERITANCE FROM DATA TYPE

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

▶ derived from the datatype

▶ Automatically adapted if the data type is changed

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

7.2   Addressing

The address resolution of the driver is BYTE-based, therefore SINT. When addressing INT and UINT variables, address in steps of two, for DINT and UDINT in steps of four. You can address single bits by adding the bit number.

SETTINGS FOR THE UNIQUE ADDRESSING OF VARIABLES

Property	Description
Name	Freely definable name.
	Attention: For every zenon project the name must be unambiguous.
Identification	Freely definable identification.
	E.g. for Resources label, comments, ...
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	For variables of object type `Extended data block`, enter the datablock number here.
	Adjustable from `0` to `4294967295`.
	You can take the exact maximum area for data blocks from the manual of the PLC.
Offset	Offset of variables. Equal to the memory address of the variable in the PLC. Adjustable from `0` to `4294967295`. Most S7 controllers support a maximum offset of 65535. You can look up the exact maximum range for each data block in the manual of the PLC.
Alignment	not used for this driver
Bit number	Number of the bit within the configured offset.
	Possible entries: `0` to `65535`. Working range [0..7]
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.
Driver connection/Priority	Setting the priority class. The variable of the priority class is thus assigned as it was configured in the driver dialog in the **General** tab. The priority classes are only used if the **global update time** is deactivated.
	If the **global update time** option is activated and the priority classes are used, there is an error entry in the log file of the system. The driver uses the highest possible priority.

EXAMPLE

- For addressing double word (DINT/UDINT) variables:
The connection was configured with net address 2. Two double words in data block 33 in a row starting from offset 20.

Addressing double word 1:

Net address:	2
Data block:	33
Offset:	20
Driver object type:	Ext. Data block
Data type:	UDINT (DINT)

Addressing double word 2:

Net address:	2
Data block:	33
Offset:	24
Driver object type:	Ext. Data block
Data type:	UDINT (DINT)

- For addressing bit (BOOL) variables:
The connection was configured with bus address 2. One marker bit with offset 79, the fourth bit.

Net address:	2
Data block:	not used
Offset:	79
Bit number:	3 (bits are counted from 0)
Driver object type:	SPS marker
Data type:	BOOL

7.3 Driver objects and datatypes

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

7.3.1 Driver objects

The following object types are available in this driver:

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

Driver Object Type	Channel type	Read	Write	Supported data types	Comment
ALARM_S Associated value	65	X	--	`BOOL, SINT, USINT, INT, UINT, DINT, UDINT`	Variables of the type ALARM_S associated value contain the associated values received with a message, if they exist. Also here the offset contains the S7 message number. The addressing of the associated value in the associated value record is realized via the bit number. If e.g. 3 associated values of the type byte are received, the first one has bit number 0, the second bit number 8 and the third bit number 16. If it is 3 words, the bit numbers are 0, 16 and 32.
ALARM_S Message	9	X	--	`BOOL, UDINT`	With the datatype ALARM_S bit, spontaneous realtime-stamped alarm messages of the S7 can be received (e.g. from PDiag). This variable only has a value, when the S7 sends an ALARM_S telegram. No initial image or similar thing is read. The offset here is the S7 message number. A variable of type ALARM_S message doubleword always contains the last received message number. The offset is not used here and has to be 0.
Output	11	X	X	`BOOL, SINT, USINT, INT, UINT`	If you want outputs to be written, you have to activate this in the general section of the driver settings.
Input	10	X	--	`BOOL, SINT, USINT, INT, UINT`	
Extended data block	34	X	X	`BOOL, DATE_AND_TIME, INT,`	STRING: max. 210 characters DATE_AND_TIME: 8 bytes time*

				SINT, USINT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL, STRING, WSTRING,	
S5Time data block	97	X	X	REAL	Only times in seconds will be read and written. Attention: 32 bits of data will be read, but in the S7, S5Time has only 16 bits; you should therefore make sure that there are still 16 bits left after the last S5Time object in a data block.
PLC marker	8	X	X	BOOL, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, LREAL	
Counter	4	X	X	UDINT	
Time	5	X	--	REAL	
Communication details	35		X	BOOL, SINT, USINT, INT, UINT, DINT, UDINT, REAL, STRING	Variables for the static analysis of the communication; is transferred between driver and Runtime (not to the PLC). 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 45) chapter.

Key:

X: supported

--: not supported

*)The data type `DATE_AND_TIME` in detail:

For each offset the following structure is read or written:

Byte	Contents	Value	Possible value range	BCD code
0	Year	0 - 99	1990 - 1999 2000 - 2089	90h - 99h 00h - 89h
1	Month	1 - 12	1 - 12	01h – 12h
2	Day	1 - 31	1 - 31	01h – 31h
3	Hour	0 - 23	0 - 23	00h - 23h
4	Minute	0 - 59	0 - 59	00h - 59h
5	Second	0 - 59	0 - 59	00h - 59h
6	Millisecond HT	0 - 990 ms Bit 4 - 7 Hundreds, 0 - 3 Tens	0 - 999	00h - 999h
7	Milliseconds O, Data	0 - 9 Bit 4 - 7 millisec. Ones, 0 - 15 Bit 0 - 3 Data		

7.3.2   Mapping of the data types

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

MAPPING OF THE DATA TYPES FROM THE PLC TO ZENON DATA TYPES

PLC	zenon	Data type
BOOL	BOOL	8
SINT	SINT	10
BYTE	USINT	9
INT	INT	1
WORD	UINT	2
DINT	DINT	3
DWORD	UDINT	4
DATE_AND_TIME	DATE_AND_TIME	20
REAL	REAL	5
STRING	STRING	12

The channel type or the data type is used in the driver for mapping the correct object types or data types. This information is also used for the "Extended variable import/export" via DBF files.

Data type: The property **Data type** is the internal numerical name of the data type. It is also used for the extended DBF import/export of the variables.

7.4   Creating variables by importing

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

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

7.4.1   XML import

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

▶ `Import:`
The element is imported as a new element.

▶ `Overwrite:`
The element is imported and overwrites a pre-existing element.

▶ `Do not import:`
The element is not imported.

Note: The actions and their durations are shown in a progress bar during import.

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 cannot be taken over. The handover of data from newer to older versions is not supported.

▶ Consistency

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

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

▶ Structure data types

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

👍 **Hint**

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

7.4.2 DBF Import/Export

Data can be exported to and imported from dBase.

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

IMPORT DBF FILE

To start the import:

1. right-click on the variable list

2. in the drop-down list of **Extended export/import...** select the **Import dBase** command

3. follow the import assistant

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

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

EXPORT DBF FILE

To start the export:

1. right-click on the variable list

2. in the drop-down list of **Extended export/import...** select the **Export dBase...** command

3. follow the export assistant

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

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

> 💡 **Information**
>
> *dBase does not support structures or arrays (complex variables) at export.*

FILE STRUCTURE OF THE DBASE EXPORT FILE

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

> ⚠️ **Attention**
>
> dBase does not support structures or arrays (complex variables) at export.
>
> DBF files must:
>
> ▸ conform with their name to the 8.3 DOS format (8 alphanumeric characters for name, 3 characters for extension, no space)
>
> ▸ Be stored close to the root directory   (Root)

STRUCTURE

Identification	Type	Field size	Comment
KANALNAME	Char	128	Variable name. The length can be limited using the **MAX_LAENGE** entry in the **project.ini** file.
KANAL_R	C	128	The original name of a variable that is to be replaced by the new name entered under "VARIABLENNAME" (variable name) (field/column must be entered manually). The length can be limited using the **MAX_LAENGE** entry in the **project.ini** file.
KANAL_D	Log	1	The variable is deleted with the 1 entry (field/column has to be created by hand).
TAGNR	C	128	Identification. The length can be limited using the **MAX_LAENGE** entry in the **project.ini** file.
EINHEIT	C	11	Technical unit
DATENART	C	3	Data type (e.g. bit, byte, word, ...) corresponds to the data type.
KANALTYP	C	3	Memory area in the PLC (e.g. marker area, data area, ...) corresponds to the driver object type.
HWKANAL	Num	3	Net address
BAUSTEIN	N	3	Datablock address (only for variables from the data area of the PLC)
ADRESSE	N	5	Offset
BITADR	N	2	For bit variables: bit address For byte variables: 0=lower, 8=higher byte For string variables: Length of string (max. 63 characters)
ARRAYSIZE	N	16	Number of variables in the array for index variables ATTENTION: Only the first variable is fully available. All others are only available for VBA or the Recipegroup Manager

LES_SCHR	L	1	Write-Read-Authorization 0: Not allowed to set value. 1: Allowed to set value.
MIT_ZEIT	R	1	time stamp in zenon (only if supported by the driver)
OBJEKT	N	2	Driver-specific ID number of the primitive object comprises TREIBER-OBJEKTTYP and DATENTYP
SIGMIN	Float	16	Non-linearized signal - minimum (signal resolution)
SIGMAX	F	16	Non-linearized signal - maximum (signal resolution)
ANZMIN	F	16	Technical value - minimum (measuring range)
ANZMAX	F	16	Technical value - maximum (measuring range)
ANZKOMMA	N	1	Number of decimal places for the display of the values (measuring range)
UPDATERATE	F	19	Update rate for mathematics variables (in sec, one decimal possible) not used for all other variables
MEMTIEFE	N	7	Only for compatibility reasons
HDRATE	F	19	HD update rate for historical values (in sec, one decimal possible)
HDTIEFE	N	7	HD entry depth for historical values (number)
NACHSORT	R	1	HD data as postsorted values
DRRATE	F	19	Updating to the output (for zenon DDE server, in [s], one decimal possible)
HYST_PLUS	F	16	Positive hysteresis, from measuring range
HYST_MINUS	F	16	Negative hysteresis, from measuring range
PRIOR	N	16	Priority of the variable
REAMATRIZE	C	32	Allocated reaction matrix
ERSATZWERT	F	16	Substitute value, from measuring range
SOLLMIN	F	16	Minimum for set value actions, from measuring range
SOLLMAX	F	16	Maximum for set value actions, from measuring range
VOMSTANDBY	R	1	Get value from standby server; the value of the variable is not requested from the server but from the Standby Server in redundant networks
RESOURCE	C	128	Resources label. Free string for export and display in lists. The length can be limited using the MAX_LAENGE entry in **project.ini**.
ADJWVBA	R	1	Non-linear value adaption: 0: Non-linear value adaption is used

			1: Non-linear value adaption is not used
ADJZENON	C	128	Linked VBA macro for reading the variable value for non-linear value adjustment.
ADJWVBA	C	128	ed VBA macro for writing the variable value for non-linear value adjustment.
ZWREMA	N	16	Linked counter REMA.
MAXGRAD	N	16	Gradient overflow for counter REMA.

⚠ **Attention**

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

LIMIT VALUE DEFINITION

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

Identification	Type	Field size	Comment
AKTIV1	R	1	Limit value active (per limit value available)
GRENZWERT1	F	20	technical value or ID number of a linked variable for a dynamic limit value (see VARIABLEx) (if VARIABLEx is 1 and here it is −1, the existing variable linkage is not overwritten)
SCHWWERT1	F	16	Threshold value for limit value
HYSTERESE1	F	14	Is not used
BLINKEN1	R	1	Set blink attribute
BTB1	R	1	Logging in CEL
ALARM1	R	1	Alarm
DRUCKEN1	R	1	Printer output (for CEL or Alarm)
QUITTIER1	R	1	Must be acknowledged
LOESCHE1	R	1	Must be deleted
VARIABLE1	R	1	Dyn. limit value linking the limit is defined by an absolute value (see field GRENZWERTx).
FUNC1	R	1	Functions linking
ASK_FUNC1	R	1	Execution via Alarm Message List
FUNC_NR1	N	10	ID number of the linked function (if "-1" is entered here, the existing function is not overwritten during import)
A_GRUPPE1	N	10	Alarm/Event Group
A_KLASSE1	N	10	Alarm/Event Class
MIN_MAX1	C	3	Minimum, Maximum
FARBE1	N	10	Color as Windows coding
GRENZTXT1	C	66	Limit value text
A_DELAY1	N	10	Time delay
INVISIBLE1	R	1	Invisible

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

7.5 Communication details (Driver variables)

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

- ▶ Information
- ▶ Configuration
- ▶ Statistics and
- ▶ Error message

The definitions of the variables implemented in the driver kit are available in the import file **drvvar.dbf** (on the installation medium in the `\Predefined\Variables` folder) and can be imported from there.

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 only for pure polling drivers
>
> ▷ Connection-related information such as ErrorMSG only for drivers that only edit one connection at a a time

INFORMATION

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

| ConnectionStates | STRING | 61 | Internal connection status of the driver to the PLC.

Connection statuses:

`0:` Connection OK

`1:` Connection failure

`2:` Connection simulated

Formating:

`<Netzadresse>:<Verbindungszustand>;…;…;`

A connection is only known after a variable has first signed in. In order for a connection to be contained in a string, a variable of this connection must be signed in once.

The status of a connection is only updated if a variable of the connection is signed in. Otherwise there is no communication with the corresponding controller. |

CONFIGURATION

Name from import	Type	Offset	Description
ReconnectInRead	BOOL	27	If TRUE, the modem is automatically reconnected for reading
ApplyCom	BOOL	36	Apply changes in the settings of the serial interface. Writing to this variable immediately results in the method SrvDrvVarApplyCom being called (which currently has no further function).
ApplyModem	BOOL	37	Apply changes in the settings of the modem. Writing this variable immediately calls the method SrvDrvVarApplyModem. This closes the current connection and opens a new one according to the settings **PhoneNumberSet** and **ModemHwAdrSet**.
PhoneNumberSet	STRING	38	Telephone number, that should be used
ModemHwAdrSet	DINT	39	Hardware address for the telephone number

GlobalUpdate	UDINT	3	Update time in milliseconds (ms).
BGlobalUpdaten	BOOL	4	TRUE, if update time is global
TreiberSimul	BOOL	5	TRUE, if driver in sin simulation mode
TreiberProzab	BOOL	6	TRUE, if the variables update list should be kept in the memory
ModemActive	BOOL	7	TRUE, if the modem is active for the driver
Device	STRING	8	Name of the serial interface or name of the modem
ComPort	UINT	9	Number of the serial interface.
Baudrate	UDINT	10	Baud rate of the serial interface.
Parity	SINT	11	Parity of the serial interface
ByteSize	USINT	14	Number of bits per character of the serial interface Value = 0 if the driver cannot establish any serial connection.
StopBit	USINT	13	Number of stop bits of the serial interface.
Autoconnect	BOOL	16	TRUE, if the modem connection should be established automatically for reading/writing
PhoneNumber	STRING	17	Current telephone number
ModemHwAdr	DINT	21	Hardware address of current telephone number
RxIdleTime	UINT	18	Modem is disconnected, if no data transfer occurs for this time in seconds (s)
WriteTimeout	UDINT	19	Maximum write duration for a modem connection in milliseconds (ms).
RingCountSet	UDINT	20	Number of ringing tones before a call is accepted
ReCallIdleTime	UINT	53	Waiting time between calls in seconds (s).
ConnectTimeout	UINT	54	Time in seconds (s) to establish a connection.

STATISTICS

Name from import	Type	Offset	Description
MaxWriteTime	UDINT	31	The longest time in milliseconds (ms) that is required for writing.
MinWriteTime	UDINT	32	The shortest time in milliseconds (ms) that is required for writing.
MaxBlkReadTime	UDINT	40	Longest time in milliseconds (ms) that is required to read a data block.
MinBlkReadTime	UDINT	41	Shortest time in milliseconds (ms) that is required to read a data block.
WriteErrorCount	UDINT	33	Number of writing errors
ReadSucceedCount	UDINT	35	Number of successful reading attempts
MaxCycleTime	UDINT	22	Longest time in milliseconds (ms) required to read all requested data.
MinCycleTime	UDINT	23	Shortest time in milliseconds (ms) required to read all requested data.
WriteCount	UDINT	26	Number of writing attempts
ReadErrorCount	UDINT	34	Number of reading errors
MaxUpdateTimeNormal	UDINT	56	Time since the last update of the priority group **Normal** in milliseconds (ms).
MaxUpdateTimeHigher	UDINT	57	Time since the last update of the priority group **Higher** in milliseconds (ms).
MaxUpdateTimeHigh	UDINT	58	Time since the last update of the priority group **High** in milliseconds (ms).
MaxUpdateTimeHighest	UDINT	59	Time since the last update of the priority group **Highest** in milliseconds (ms).
PokeFinish	BOOL	55	Goes to 1 for a query, if all current pokes were executed

ERROR MESSAGE

Name from import	Type	Offset	Description

ErrorTimeDW	UDINT	2	Time (in seconds since 1.1.1970), when the last error occurred.
ErrorTimeS	STRING	2	Time (in seconds since 1.1.1970), when the last error occurred.
RdErrPrimObj	UDINT	42	Number of the PrimObject, when the last reading error occurred.
RdErrStationsName	STRING	43	Name of the station, when the last reading error occurred.
RdErrBlockCount	UINT	44	Number of blocks to read when the last reading error occurred.
RdErrHwAdresse	DINT	45	Hardware address when the last reading error occurred.
RdErrDatablockNo	UDINT	46	Block number when the last reading error occurred.
RdErrMarkerNo	UDINT	47	Marker number when the last reading error occurred.
RdErrSize	UDINT	48	Block size when the last reading error occurred.
DrvError	USINT	25	Error message as number
DrvErrorMsg	STRING	30	Error message as text
ErrorFile	STRING	15	Name of error log file

8. Driver-specific functions

The driver supports the following functions:

INAT BOARD (OPTIONAL)

This card can replace an original Siemens CP-443-1/TCP.
The requests are handled via RFC 1006. The definition of the send/receive orders is done in the INAT configurator.
On the zenon side, you have to enter the same own and remote TSAPs have as on the according board.
Several simultaneous and parallel connections are possible.
If several independent zenon servers communicate with one card, a separate connection for each server has to be defined on the INAT-CP.
The card can handle S5 and S7 protocols simultaneously.
The configuration on the PLC is realized by creating a configuration block in OB 100 or 101 (for restart),

i.e. synchronizing the CPU with the board.

In a cyclically called block the block NET_ALL is called (like in an S5 Send-All/Receive-All), which handles the data communication with the CPU.

F&S (FISCHER & SCHMIDT)

In order to be able to use the datatypes for F&S, they must be activated explicitly. This activation is carried out with the following entry in the project.ini file:

[S7TCP]
FS=1

ALARM-S

With the datatype ALARM_S message bit, spontaneous realtime-stamped alarm messages of the S7 can be received (e.g. from PDiag). This variable only has a value, when the S7 sends an ALARM_S telegram. No initial image or similar thing is read. The offset here is the S7 message number.

A variable of type ALARM_S message doubleword always contains the last received message number. The offset is not used here and has to be 0.

Variables of the type 'ALARM_S associated value' contain the associated values received with a message, if they exist. Also here the offset contains the S7 message number. The addressing of the associated value in the associated value record is realized with the bit number. If e.g. 3 associated values of the type byte are received, the first one has bit number 0, the second bit number 8 and the third bit number 16. If it is 3 words, the bit numbers are 0, 16 and 32. On transmitting PDiag associated values always a doubleword is sent. In the doubleword the low byte and the high byte are permuted; this has to be cared of in the addressing of the bit number.

EXAMPLES:

▶ Associated value = bit: offset = message number / bit number = 24

▶ associated value = byte offset = message number / bit number = 24

▶ associated value = word offset = message number / bit number = 16

▶ associated value = doubleword offset = message number / bit number = 0

ALARM-8

Configuration is similar to ALARM_S.

Differences:

▶ The index of the message at the ALARM_8 block must also be given in the database.

▶ For ALARM_8, 8 bit messages are transferred for each message number. The 10 associated values are packed and processed consecutively as Dump.
Maximum size of all associated values together per message: 32 bytes

▸ The initial stack, which is read after the connection is established, contains in contrast to ALARM_S neither associated values nor time stamps.

LIMITATIONS

In projects with the S7 TCP driver, note that S7 PLCs only have limited communication resources. This becomes noticeable with the S7 300 types, that particularly for the smaller models allow only a few (6-12) simultaneous communications.

Example:
S7 312C only allows 6 connections, where as a default 4 are reserved, so that only 2 simultaneous connections are possible.
All reserved connections with the exception of the PD communication and an OP communication can be released, so for a S7 312C a maximum of 4 free connections is available.
S7 414-2 allows 32 connections, 4 reserved as a default, and 28 additional possible.

Access to S7-200 via CP243, in S7 200 only DB 1 exists.

PROJECT.INI ENTRIES

[S7TCP]	
BLOCKWRITE=1	Activates blockwrite

8.1   Configuration file

The configuration file must be in the current project directory. The file name can be freely defined.

Save path:
`C:\ProgramData\COPA-DATA\SQL2012\54af8312-0a04-46c9-ba32-16093eefc323\`
`FILES\zenon\custom\drivers\[driver name]_[driver identification].txt`

FILE STRUCTURE

Entry	Description
[FETCH_HWn]	Entry of a connection configuration. A [FETCH_HW] entry for each configured connedtion will be created. n represents the net address of the connection.
VERB_NAME=	Configured name of the connection.
FREMD_IP=	IP address or host name of the primary connection
FREMDE_IP2=	IP address or host name of the secondary connection. Empty if no secondary connection is configured.
EIGENE_TSAP=	Configuration for the **Local TSAP** property for the primary connection.
FREMDE_TSAP=	Configuration for the **Remote TSAP** for the primary connection.
EIGENE_TSAP2=	Configuration for the **Local TSAP** property for the secondary connection.
FREMDE_TSAP2=	Configuration for the **Remote TSAP** property for the secondary connection.
USES_GMT=	Configuration of the **CPU uses, for ALARM_x GMT** property 0: Not activated 1: activated
USES_ALARM_S=	Configuration of the **CPU supports ALARM_S** property 0: Not activated 1: activated
USES_ALARM_8=	Configuration of the **CPU supports ALARM_8** property 0: Not activated 1: activated
SWITCHONSTOP=0	Configuration of the Switch if CPU status is STOP property: 0: Not activated 1: activated

9. Driver command function

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

▸ Start

▸ Stop

▸ Shift a certain driver mode

▸ Instigate certain actions

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

> 💡 **Information**
>
> *This chapter describes standard functions that are valid for most zenon drivers. However, 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.*

CONFIGURATION OF THE FUNCTION

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

1. Create a new function in the zenon Editor.

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 Runtime. Heed the notices in the **Driver command function in the network** section.

DRIVER COMMAND DIALOG
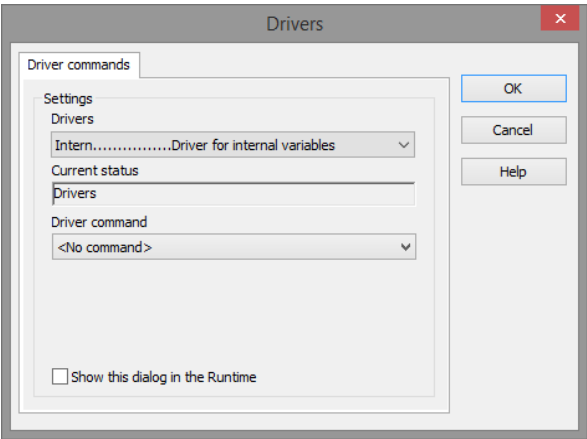
Option	Description
Drivers	Selection of the driver from the drop-down list. It contains all drivers loaded in the project.
Current status	Fixed entry which has no function in the current version.
Driver command	Drop-down list for the selection of the command:
<No command>	No command is sent. A command that already exists can thus be removed from a configured function.
Start driver (online mode)	Driver is reinitialized and started.
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	Enter driver-specific commands. Opens input field in order to enter a command.
Activate driver write set value	Write set value to a driver is allowed.
Deactivate driver write set value	Write set value to a driver is prohibited.
Establish connection with modem	Establish connection (for modem drivers) Opens the input fields for the hardware address and for the telephone number.
Disconnect from modem	Terminate connection (for modem drivers)
Driver in counting simulation mode	Driver is set into counting simulation mode. All values are initialized with 0 and incremented in the set update time by 1 each time up to the maximum value and then start at 0 again.
Driver in static simulation mode	Driver is set into counting simulation mode. All values are initialized with 0.
Driver in programmed simulation mode	Driver is set into counting 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.
Show this dialog in the Runtime	The dialog is shown in Runtime so that changes can be made.

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, which then executes the desired action on its driver. In addition, the Server sends the same driver command to the project standby. The standby also carries out the action on its driver.

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

10. Error analysis

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

10.1 Analysis tool

All zenon modules such as Editor, Runtime, drivers, etc. write messages to a joint log file. To display them correctly and clearly, use the Diagnosis Viewer (main.chm::/12464.htm) program that was also installed with zenon. You can find it under *Start/All programs/zenon/Tools 8.00 -> Diagviewer.*

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

%ProgramData%\COPA-DATA\LOG.

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

In the Diagnosis Viewer you can also:

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

Note:

1. The Diagnosis Viewer displays all entries in UTC (coordinated world time) and not in local time.

2. The Diagnosis Viewer does not display all columns of a LOG file per default. To display more columns activate property **Add all columns with entry** in the context menu of the column header.

3. If you only use **Error-Logging**, the problem description is in the column **Error text**. For other diagnosis level the description is in the column **General text**.

4. For communication problems many drivers also log error numbers which the PLC assigns to them. They are displayed in **Error text** or **Error code** or **Driver error parameter** (**1** and **2**). Hints on the meaning of error codes can be found in the driver documentation and the protocol/PLC description.

5. At the end of your test set back the diagnosis level from **Debug** or **Deep Debug**. At **Debug** and **Deep Debug** there are a great deal of data for logging which are saved to the hard drive and which can influence your system performance. They are still logged even after you close the Diagnosis Viewer.

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

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

10.2   Error numbers

Example of a log entry:

Error Read - HW:0 Kennung:132 DB:10 OFF:599 Count:4 Error:wrong length

Error Read

Read error

HW:0

Net address of the PLC (according to driver configuration)

Identification 132

(Internal Siemens ID)

·   129   Input

·   130   Output

·   131   Marker

· 132 Data block

DB:10

Number of the data block as defined in the variable configuration

OFF:599

Offset as defined in variable configuration

Count 4

The block size to be read, usually in bytes. The driver optimizes this size automatically. Attention: Hint: Size must also be readable from the PLC, i.e. it must exist !! (in the case of Offset 599 and Count 4, the data block must be configured until Offset 603)

Error:wrong length

Error source - if known

Remote Error Code (e.g.: 0xA) Para1   Para2

Error codes not disclosed by Siemens !

ERROR CODES IN THE API

The following is a list of possible error codes returned by the WSAGetLastError call, along with their ex-tended explanations. Errors are listed in alphabetical order by error macro. Some error codes defined in Winsock2.h are not returned from any function-these are not included in this topic.

Error (Code)	Meaning	Description
WSAEACCES (10013)	Permission denied.	An attempt was made to access a socket in a way forbidden by its access permissions. An example is using a broadcast address for sendto without broadcast permission being set using set-sockopt(SO_BROADCAST). Another possible reason for the WSAEACCES error is that when the bind function is called (on Windows NT 4 SP4 or later), another application, service, or kernel mode driver is bound to the same address with exclusive access. Such exclusive access is a new feature of Windows NT 4 SP4 and later, and is imple-mented by using the SO_EXCLUSIVEADDRUSE option.
WSAEADDRINUSE (10048)	Address already in use.	Typically, only one usage of each socket address (protocol/IP address/port) is permitted. This error oc-curs if an application attempts to bind a socket to an IP address/port that has already been used for an existing socket, or a socket that was not closed properly, or one that is still in the process of closing. For server applications that need to bind multiple sockets to the same port number, consider using set-sockopt(SO_REUSEADDR). Client applications usually need not call bind at all-connect chooses an unused port automatically. When bind is called with a wildcard address (involving ADDR_ANY), a WSAEADDRINUSE error could be delayed until the specific address is committed. This could happen with a call to another function later, including connect, listen, WSAConnect, or WSAJoinLeaf.
WSAEADDRNOTAVAIL (10049)	Cannot assign requested address.	The requested address is not valid in its context. This normally results from an attempt to bind to an address that is not valid for the local machine. This can also result from connect, sendto, WSAConnect, WSAJoinLeaf, or WSASendTo when the remote address or port is not valid for a remote machine (for example, address or port 0).
WSAEAFNOSUPPORT (10047)	Address family not supported by protocol family.	An address incompatible with the requested protocol was used. All sockets are created with an associ-ated address family (that is, AF_INET for Internet Protocols) and a generic protocol type (that is, SOCK_STREAM). This error is returned if an incorrect protocol is explicitly requested in the socket call, or if an address of the wrong family is used for a socket, for example, in sendto.
WSAEALREADY (10037)	Operation already in progress.	An operation was attempted on a nonblocking socket with an operation already in progress-that is, calling connect a second time on a nonblocking socket that is already connecting, or canceling an asynchronous request (WSAAsyncGetXbyY) that has already been canceled or completed.
WSAECONNABORTE	Software caused	An established connection was aborted by the software in your

D **(10053)**	connection abort.	host machine, possibly due to a data transmission time-out or protocol error.
WSAECONNREFUSED **(10061)**	Connection refused.	No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host-that is, one with no server application running.
WSAECONNRESET **(10054)**	Connection reset by peer.	An existing connection was forcibly closed by the remote host. This normally results if the peer application on the remote host is suddenly stopped, the host is rebooted, or the remote host uses a hard close (see setsockopt for more information on the SO_LINGER option on the remote socket.) This error may also result if a connection was broken due to keep-alive activity detecting a failure while one or more operations are in progress. Operations that were in progress fail with WSAENETRESET. Subsequent operations fail with WSAECONNRESET.
WSAEDESTADDRREQ **(10039)**	Destination address required.	A required address was omitted from an operation on a socket. For example, this error is returned if sendto is called with the remote address of ADDR_ANY.
WSAEFAULT **(10014)**	Bad address.	The system detected an invalid pointer address in attempting to use a pointer argument of a call. This error occurs if an application passes an invalid pointer value, or if the length of the buffer is too small. For instance, if the length of an argument, which is a SOCKADDR structure, is smaller than the sizeof(SOCKADDR).
WSAEHOSTDOWN **(10064)**	Host is down.	A socket operation failed because the destination host is down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.
WSAEHOSTUNREACH **(10065)**	No route to host.	A socket operation was attempted to an unreachable host. See WSAENETUNREACH.
WSAEINPROGRESS **(10036)**	Operation now in progress.	A blocking operation is currently executing. Windows Sockets only allows a single blocking operation-per- task or thread-to be outstanding, and if any other function call is made (whether or not it references that or any other socket) the function fails with the WSAEINPROGRESS error.
WSAEINTR **(10004)**	Interrupted function call.	A blocking operation was interrupted by a call to WSACancelBlockingCall.
WSAEINVAL	Invalid argument.	Some invalid argument was supplied (for example, specifying an invalid level to the setsockopt function). In some instances, it also refers to the current state of the socket-for instance,

(10022)		calling accept on a socket that is not listening.
WSAEISCONN (10056)	Socket is already connected.	A connect request was made on an already-connected socket. Some implementations also return this error if sendto is called on a connected SOCK_DGRAM socket (for SOCK_STREAM sockets, the to pa-rameter in sendto is ignored) although other implementations treat this as a legal occurrence.
WSAEMFILE (10024)	Too many open files.	Too many open sockets. Each implementation may have a maximum number of socket handles avail-able, either globally, per process, or per thread.
WSAEMSGSIZE (10040)	Message too long.	A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram was smaller than the datagram itself.
WSAENETDOWN (10050)	Network is down.	A socket operation encountered a dead network. This could indicate a serious failure of the network sys-tem (that is, the protocol stack that the Windows Sockets DLL runs over), the network interface, or the local network itself.
WSAENETRESET (10052)	Network dropped connection on reset.	The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress. It can also be returned by setsockopt if an attempt is made to set SO_KEEPALIVE on a con-nection that has already failed.
WSAENETUNREACH (10051)	Network is unreachable.	A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host.
WSAENOBUFS (10055)	No buffer space available.	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.
WSAENOPROTOOPT (10042)	Bad protocol option.	An unknown, invalid or unsupported option or level was specified in a getsockopt or setsockopt call.
WSAENOTCONN (10057)	Socket is not connected.	A request to send or receive data was disallowed because the socket is not connected and (when send-ing on a datagram socket using sendto) no address was supplied. Any other type of operation might also return this error-for example, setsockopt setting SO_KEEPALIVE if the connection has been reset.
WSAENOTSOCK (10038)	Socket operation on nonsocket.	An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for select, a member of an fd_set was not valid.
WSAEOPNOTSUPP (10045)	Operation not supported.	The attempted operation is not supported for the type of object referenced. Usually this occurs when a socket descriptor to a socket that cannot support this operation is trying to accept a connection on a datagram socket.

WSAEPFNOSUPPOR T **(10046)**	Protocol family not supported.	The protocol family has not been configured into the system or no implementation for it exists. This mes-sage has a slightly different meaning from WSAEAFNOSUPPORT. However, it is interchangeable in most cases, and all Windows Sockets functions that return one of these messages also specify WSAEAFNOSUPPORT.
WSAEPROCLIM **(10067)**	Too many processes.	A Windows Sockets implementation may have a limit on the number of applications that can use it simul-taneously. WSAStartup may fail with this error if the limit has been reached.
WSAEPROTONOSUP PORT **(10043)**	Protocol not supported.	The requested protocol has not been configured into the system, or no implementation for it exists. For example, a socket call requests a SOCK_DGRAM socket, but specifies a stream protocol.
WSAEPROTOTYPE **(10041)**	Protocol wrong type for socket.	A protocol was specified in the socket function call that does not support the semantics of the socket type requested. For example, the ARPA Internet UDP protocol cannot be specified with a socket type of SOCK_STREAM.
WSAESHUTDOWN **(10058)**	Cannot send after socket shutdown.	A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown call. By calling shutdown a partial close of a socket is requested, which is a signal that sending or receiving, or both have been discontinued.
WSAESOCKTNOSUP PORT **(10044)**	Socket type not supported.	The support for the specified socket type does not exist in this address family. For example, the optional type SOCK_RAW might be selected in a socket call, and the implementation does not support SOCK_RAW sockets at all.
WSAETIMEDOUT **(10060)**	Connection timed out.	A connection attempt failed because the connected party did not properly respond after a period of time, or the established connection failed because the connected host has failed to respond.
WSATYPE_NOT_FOU ND **(10109)**	Class type not found.	The specified class was not found.
WSAEWOULDBLOCK **(10035)**	Resource temporarily unavailable.	This error is returned from operations on nonblocking sockets that cannot be completed immediately, for example recv when no data is queued to be read from the socket. It is a nonfatal error, and the operation should

		be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connect on a nonblocking SOCK_STREAM socket, since some time must elapse for the connection to be established.
WSAHOST_NOT_FOUND **(11001)**	Host not found.	No such host is known. The name is not an official host name or alias, or it cannot be found in the data-base(s) being queried. This error may also be returned for protocol and service queries, and means that the specified name could not be found in the relevant database.
WSA_INVALID_HANDLE **(OS dependent)**	Specified event object handle is invalid.	An application attempts to use an event object, but the specified handle is not valid.
WSA_INVALID_PARAMETER **(OS dependent)**	One or more parameters are invalid.	An application used a Windows Sockets function which directly maps to a Win32 function. The Win32 function is indicating a problem with one or more parameters.
WSAINVALIDPROCTABLE **(OS dependent)**	Invalid procedure table from service provider.	A service provider returned a bogus procedure table to Ws2_32.dll. (Usually caused by one or more of the function pointers being null.)
WSAINVALIDPROVIDER **(OS dependent)**	Invalid service provider version number.	A service provider returned a version number other than 2.0.
WSA_IO_INCOMPLETE **(OS dependent)**	Overlapped I/O event object not in signaled state.	The application has tried to determine the status of an overlapped operation which is not yet completed. Applications that use WSAGetOverlappedResult (with the fWait flag set to FALSE) in a polling mode to determine when an overlapped operation has completed, get this error code until the operation is com-plete.
WSA_IO_PENDING **(OS dependent)**	Overlapped operations will complete later.	The application has initiated an overlapped operation that cannot be completed immediately. A comple-tion indication will be given later when the operation has been completed.
WSA_NOT_ENOUGH_MEMORY **(OS dependent)**	Insufficient memory available.	An application used a Windows Sockets function that directly maps to a Win32 function. The Win32 func-tion is indicating a lack of required memory resources.
WSANOTINITIALISE	Successful	Either the application has not called WSAStartup or

D (10093)	WSAStartup not yet performed.	WSAStartup failed. The application may be access-ing a socket that the current active task does not own (that is, trying to share a socket between tasks), or WSACleanup has been called too many times.
WSANO_DATA (11004)	Valid name, no data record of requested type.	The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for. The usual example for this is a host name-to-address translation attempt (using gethostbyname or WSAAsyncGetHostByName) which uses the DNS (Domain Name Server). An MX record is returned but no A record-indicating the host itself exists, but is not directly reachable.
WSANO_RECOVERY (11003)	This is a nonrecoverable error.	This indicates some sort of nonrecoverable error occurred during a database lookup. This may be be-cause the database files (for example, BSD-compatible HOSTS, SERVICES, or PROTOCOLS files) could not be found, or a DNS request was returned by the server with a severe error.
WSAPROVIDERFAIL EDINIT (OS dependent)	Unable to initialize a service provider.	Either a service provider's DLL could not be loaded (LoadLibrary failed) or the provider's WSPStartup/NSPStartup function failed.
WSASYSCALLFAILU RE (OS dependent)	System call failure.	Returned when a system call that should never fail does. For example, if a call to WaitForMultipleObjects fails or one of the registry functions fails trying to manipulate the protocol/name space catalogs.
WSASYSNOTREADY (10091)	Network subsystem is unavailable.	This error is returned by WSAStartup if the Windows Sockets implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable. Users should check: That the appropriate Windows Sockets DLL file is in the current path. That they are not trying to use more than one Windows Sockets implementation simultaneously. If there is more than one Winsock DLL on your system, be sure the first one in the path is appropriate for the network subsystem currently loaded. The Windows Sockets implementation documentation to be sure all necessary components are currently installed

		and configured correctly.
WSATRY_AGAIN **(11002)**	Nonauthoritative host not found.	This is usually a temporary error during host name resolution and means that the local server did not receive a response from an authoritative server. A retry at some time later may be successful.
WSAVERNOTSUPPORTED **(10092)**	Winsock.dll version out of range.	The current Windows Sockets implementation does not support the Windows Sockets specification ver-sion requested by the application. Check that no old Windows Sockets DLL files are being accessed.
WSAEDISCON **(10101)**	Graceful shutdown in progress.	Returned by WSARecv and WSARecvFrom to indicate that the remote party has initiated a graceful shut-down sequence.
WSA_OPERATION_ABORTED **(OS dependent)**	Overlapped operation aborted.	An overlapped operation was canceled due to the closure of the socket, or the execution of the SIO_FLUSH command in WSAIoctl.

10.3   Check list

Problem	Diagnostics	Reason
Values can be read or written by the controller.	The controller can be contacted by 'pinging'?	▸ The controller is not connected to the power supply or the network. ▸ The PC is not connected to the network. ▸ The controller is connected but is in a different subnetwork and the network gateway is not entered in the controller or the subnetmask is not set correctly. ▸ Is the firewall activated? Port 102 is used for communication; add it to the exceptions.
	The controller can be contacted by 'pinging'?	▸ Have the communication parameters been set correctly? - Remote TSAP for example 02.02 if the S7 CPU is in the second slot. or 02.01 in the first slot, such as with 1200 and 1500 series controllers. ▸ The network address in the addressing of the variable does not correspond to the network

		address of the connection in the driver.
		▶ The driver configuration file was not transferred to the target computer.
Certain values cannot be read or written by the controller.	Has an analysis with the Diagnosis Viewer been carried out to see which errors have occurred? See Analysis tool (on page 57) chapter.	▶ See the following chapter: Error numbers (on page 58) ▶ Are the used datablocks defined correctly in the PLC? ▶ Are the variables correctly addressed? ▶ Is the 'Write outputs' checkbox set? (if outputs are to be written to the output terminals)
Incorrect values are displayed.	Has an analysis with the Diagnosis Viewer been carried out to see which errors have occurred? See Analysis tool (on page 57) chapter.	▶ Are the variables correctly addressed? ▶ Are the correct data types used? ▶ Is the value calculation correct?

11. Example: spontaneous communication ALARM_S, ALARM_8 and ALARM_8P

In this example, you find out how you can use spontaneous communication with ALARM_S, ALARM_8 and ALARM-8P.

ALARM_8 AND ALARM_8P

In the driver configuration, it is possible to activate `ALARM_S messages` and `ALARM_8 message` for each connection. The `ALARM_8 messages` and the associated values are to be configured in the same way as **ALARM_S**. Except that, in addition, the index of the message on the **ALARM_8** block is to be given in the database, because 8 bit messages are transferred per message number with **ALARM_8**. The 10 associated values are packed and processed consecutively as Dump. The maximum size of all associated values together per message is set at 32 bytes. With **ALARM_8** messages, the initial Stack that is read after the connection has been established has, in contrast to **ALARM_S**, neither associated values nor a time stamp.

GENERAL INFORMATION ON ALARM_S

Variable object types:

▶ `ALARM_S associated value`

Variables of the type `ALARM_S associated value` contain the associated values received with a message, if they exist. Also here the offset contains the S7 message number. The addressing of the associated value in the associated value record is realized via the bit number. If e.g. 3 associated values of the type `Byte` are received, the first one has bit number `0`, the second bit number `8` and the third bit number `16`. If it is 3 words, the bit numbers are `0`, `16` and `32`.

▶ `Message ALARM_S`

With the datatype `ALARM_S bit`, spontaneous realtime-stamped alarm messages of the S7 can be received, e.g. from PDiag. This variable only has a value, when the S7 sends an **ALARM_S** telegram. No initial image or similar thing is read. The offset here is the S7 message number. A variable of type `ALARM_S message` doubleword always contains the last received message number. The offset is not used here and has to be `0`.

11.1 Configuration of driver and variables for Alarm_8 messages

For the receipt of **Alarm_8** messages, the following parameters must be set in the **TCP/IP connection** tab in the **ALARM_x options** section:
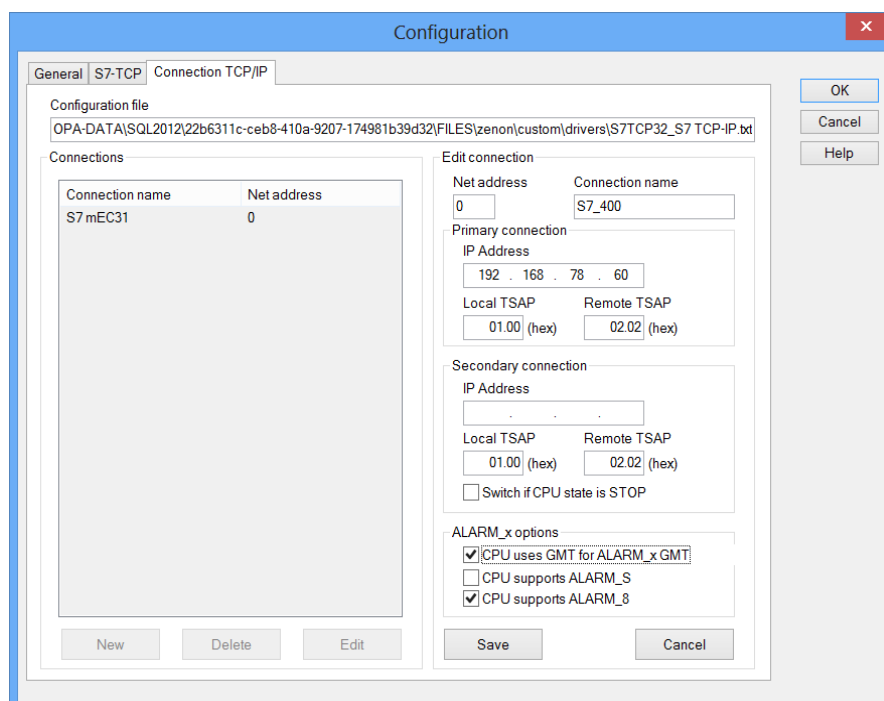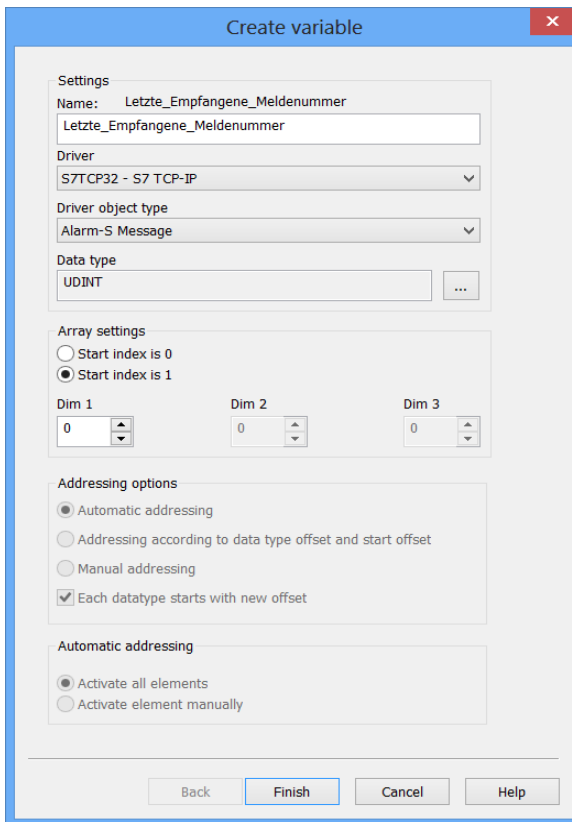
Note: This dialog is only available in English.

The driver thus sends the initialization for the receipt of **ALARM_8**.

VALIDATE CONFIGURATION

You can check to see that messages are received using an information variable:

1. Create a new variable with the `Alarm-S Message` object type.
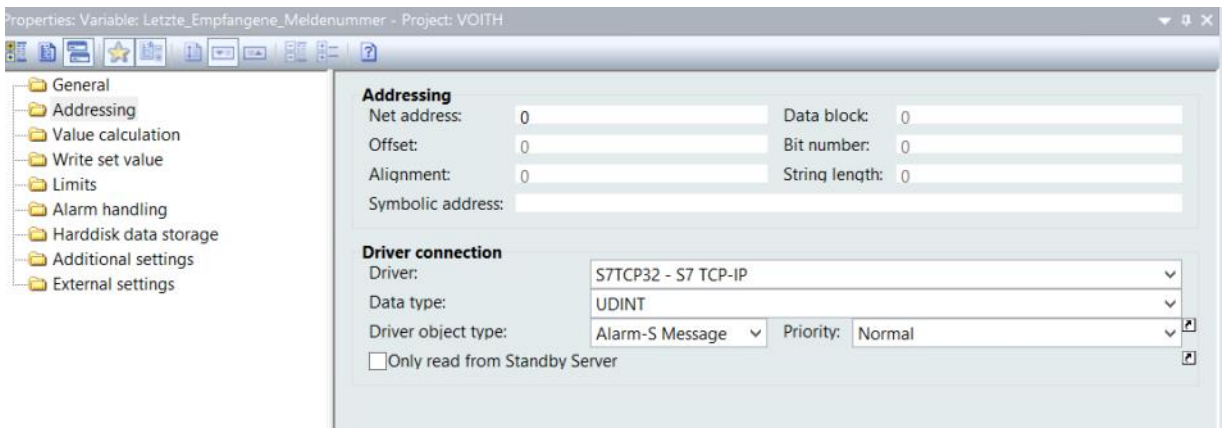
2. Enter the following address settings:
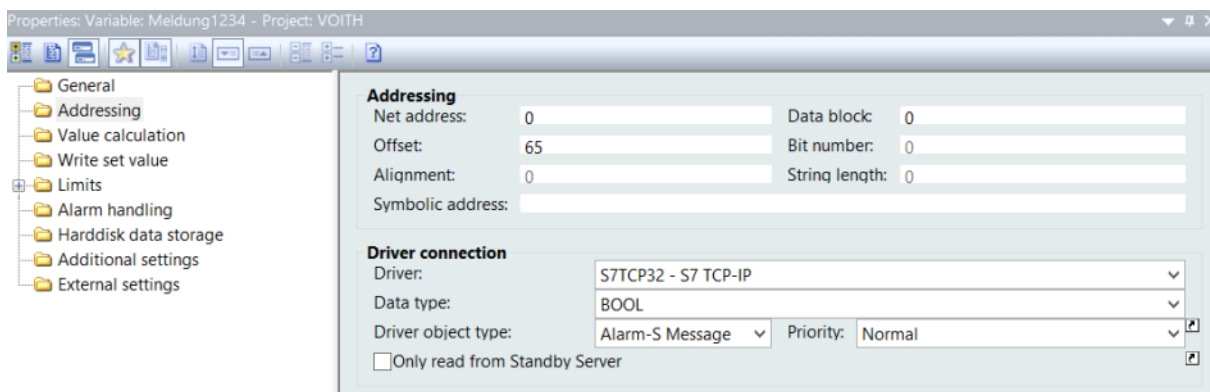
In Runtime, you then get the last-received **ALARM_S** message number in this variable.

ALARM_8 MESSAGE

The message numbers are important for the actual messages.
The message itself is configured as an `Alarm-S Message BOOL` variable. You enter the message number in the addressing in the **Offset** property:
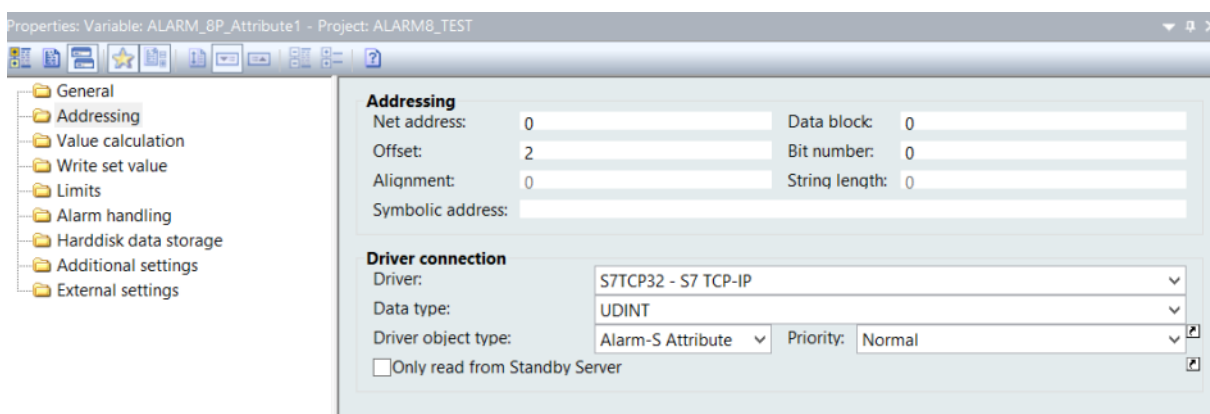
For **Alarm_8** messages, the index of the message must also be entered in the **Data block** property. This corresponds to the index of the **Alarm_8** module input to which the message is generated (`0 -7`).

ALARM_8P ASSOCIATED VALUE

Associated values can be evaluated for each alarm message. These are offered in the driver as a separate object type. Assignment is as for the message via the address parameters:

Properties to be configured:

▶ **Offset**: Message number from the S7 project configuration.

▶ **Data block**: Index of the signal from the ALARM_8P block.

▶ **Bit number**: Start address (in bits) of the ALARM_8P associated value.
   Offers a maximum of 10 associated values, depending on the data types used.
   Maximum: 32 bytes = maximum 255 bits

11.2   Example project

The following hardware is used for this example:

▶   PLC: S7_412 CPU using Firmware V 3.1.3

▶   Communication processor: CP 434

▶   PC: HP EliteBook 8560w (Core i7) with Windows 8

STEP 7 PROJECT

The Step 7 project contains the following calls for **Alarm_8-** and **Alarm_8P-**generation in OB1. The `program-defined` method was selected for issuing the **Event ID**.

ALARM_8

```
CALL  "ALARM_8" , DB108          SFB34          -- Generate Block-Related Messages without Values for 8 Signals
   EN_R     :=M400.0
   SIG_1    :=M401.0
   SIG_2    :=M401.1
   SIG_3    :=M401.2
   SIG_4    :=M401.3
   SIG_5    :=M401.4
   SIG_6    :=M401.5
   SIG_7    :=M401.6
   SIG_8    :=M401.7
   ID       :=W#16#EEEE
   EV_ID    :=DW#16#1
   SEVERITY :=
   DONE     :=M403.0
   ERROR    :=M403.1
   STATUS   :=MW404
   ACK_STATE:=MW405
```

ALARM_8P

```
CALL  "ALARM_8P" , DB208         SFB35          -- Generate Block-Related Messages with Values for 8 Signals
   EN_R     :=M410.0
   SIG_1    :=M410.0
   SIG_2    :=M410.1
   SIG_3    :=M410.2
   SIG_4    :=M410.3
   SIG_5    :=M410.4
   SIG_6    :=M410.5
   SIG_7    :=M410.6
   SIG_8    :=M410.7
   ID       :=W#16#EEEE
   EV_ID    :=DW#16#2
   SEVERITY :=
   DONE     :=M413.0
   ERROR    :=M413.1
   STATUS   :=MW414
   ACK_STATE:=MW415
   SD_1     :=MD420
   SD_2     :=MD424
   SD_3     :=MD428
   SD_4     :=MD432
   SD_5     :=MD436
   SD_6     :=MD440
   SD_7     :=MD444
   SD_8     :=MD448
   SD_9     :=MD452
   SD_10    :=MD456
```

THE USER INTERFACE OF THE ZENON TEST PROJECT:
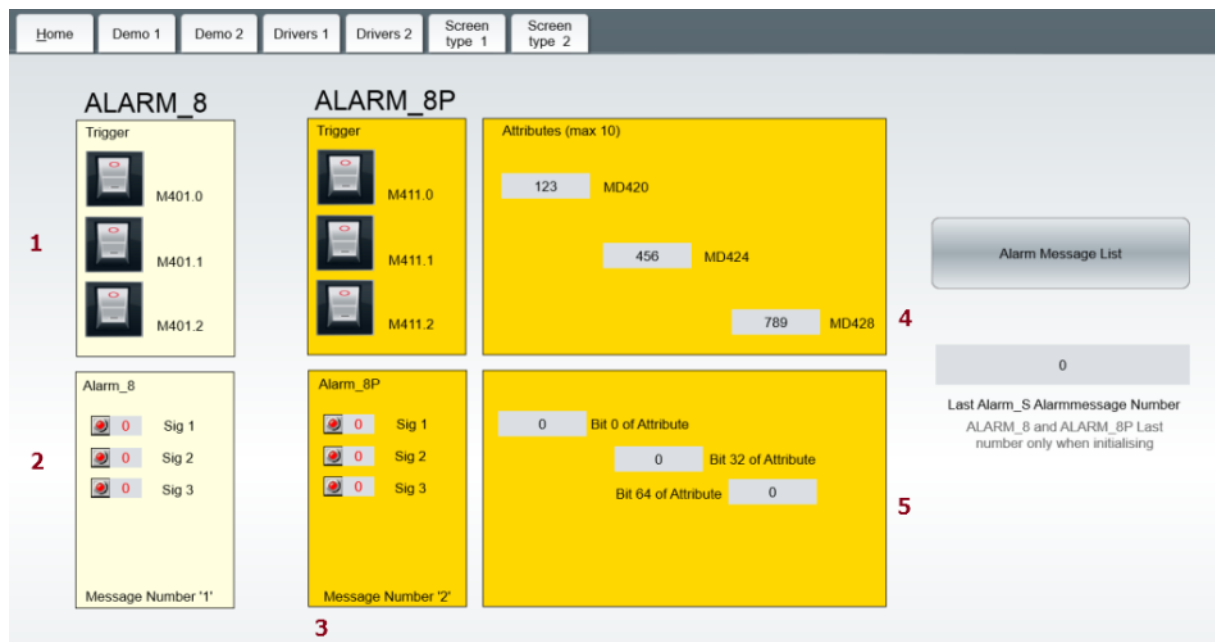
Key:

1. The trigger variables are set as PLC markers from zenon.

2. The **Alarm_8** messages are created as an **Alarm-S** message in zenon.
    In this example, the **Alarm-8** block has the message number 1 (see also the following in S7 code: `EV_ID:=DW#16#1`).

3. The **ALARM_8P** messages are created in a similar way to the **Alarm_8** messages (Item 2).
    In this example, the **Alarm-8P** block has the message number 2 (see also the following in the S7 code: `EV_ID:=DW#16#1`).

4. The associated values are set from zenon using the PLC marker (MD420..MD456).

5. The associated values for **ALARM_8P** messages are created as `Alarm-S Attribute` in zenon.
    The offset is configured using the **Bit** address setting. Whereby the first associated value starts at bit 0, the second (because it is a double word) starts at bit 32 and the third starts at bit 64.
    Attention: Total length of the associated value DUMP is 32 BYTES.

11.3   Triggering an ALARM_8 message

In our example, the marker **M401.0** triggers the **ALARM_8** message:
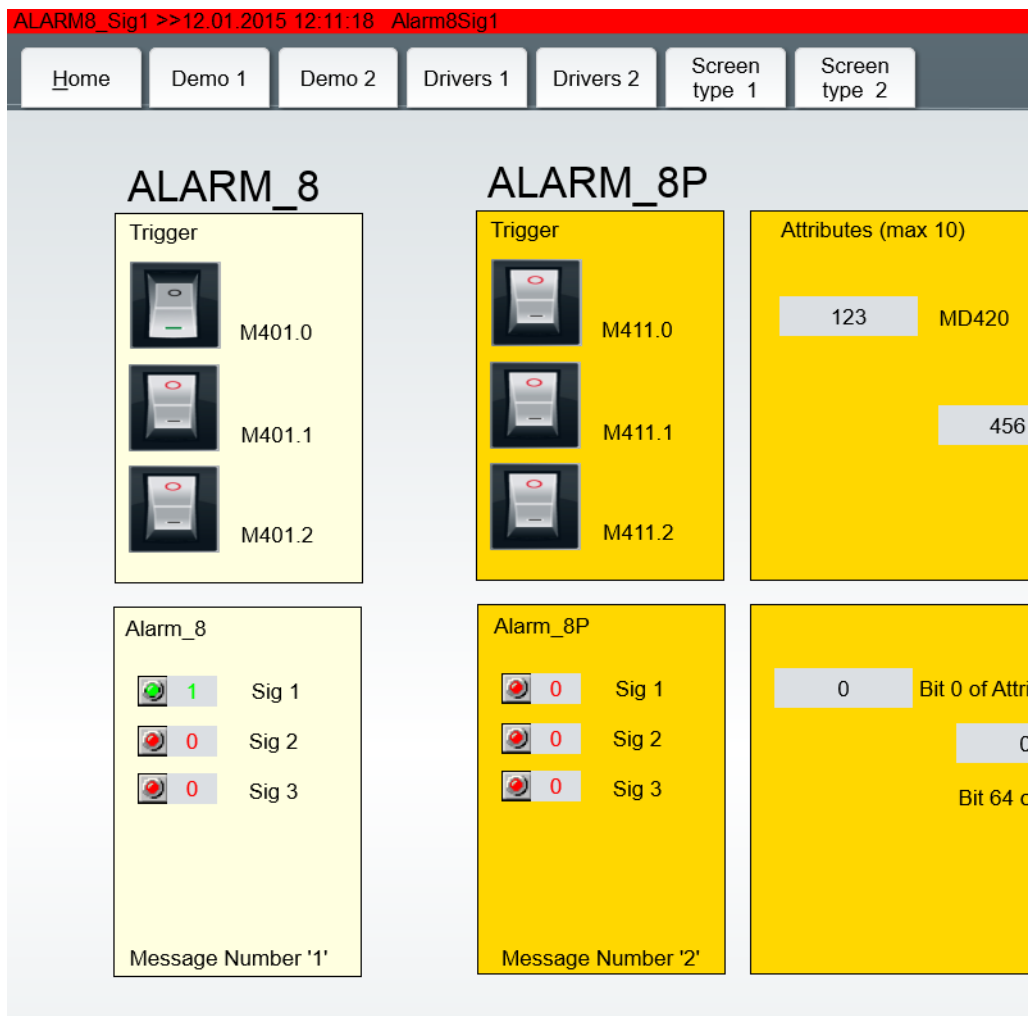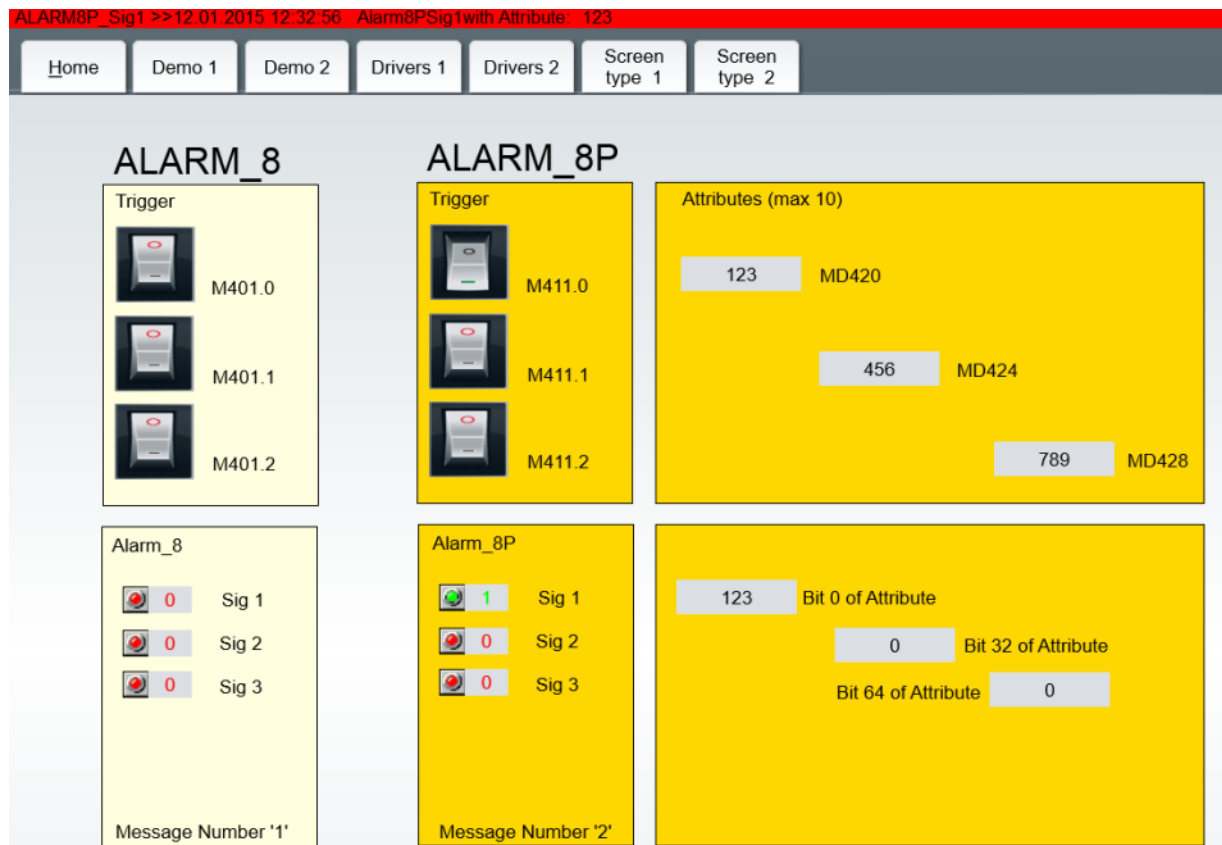
View in the Alarm Message List:

To demonstrate the time stamp, a different time was set on the PLC. The **Alarm8Trigger1** variable is a `PLC marker` that triggers the **Alarm-8** message**Signal1**.

11.4   Triggering an ALARM_8P message

In our example, the marker **M411.0** triggers the **ALARM_8P** message:

The associated value of the **Alarm-8P** block is sent with the message and displayed using the associated variables (attributes).

The associated value is inserted in the alarm text as **dynamic limit value text**.
To do this, the following text is entered in the **Limit value text** property:

```
$Alarm8PSig1with Attribute: ;%ALARM_8P_Attribute1;
```

Meaning of the control characters:

- ▶ **$**: Notice that a **dynamic limit value text will follow**.

- ▶ **;**: Separates the individual instructions.

- ▶ **%**: Key symbol to reference a variable whose value is to be inserted.

Display in the Alarm Message List

The associated value is also displayed in the alarm text.

11.5 Configuration details of the example

This example is based on the following project configuration details for **ALARM_8** and **ALARM_8P** messages and associated values for for **Offset**, **Data block** and **Bit number**:

ALARM-8 MESSAGES

> ▶ **ALARM8_Sig1**:
>
> • **Offset**: 1 (=message number from S7)
>
> • **Data block**: 0 (=index of the signal on the **ALARM_8** block)
>
> ▶ **ALARM8_Sig2**:
>
> • **Offset**: 1 (=message number from S7)
>
> • **Data block**: 1 (=index of the signal on the **ALARM_8** block)

► **ALARM8_Sig3**:

- **Offset**: 1 (=message number from S7)

- **Data block**: 2 (=index of the signal on the **ALARM_8 Block**)

ALARM-8P MESSAGE
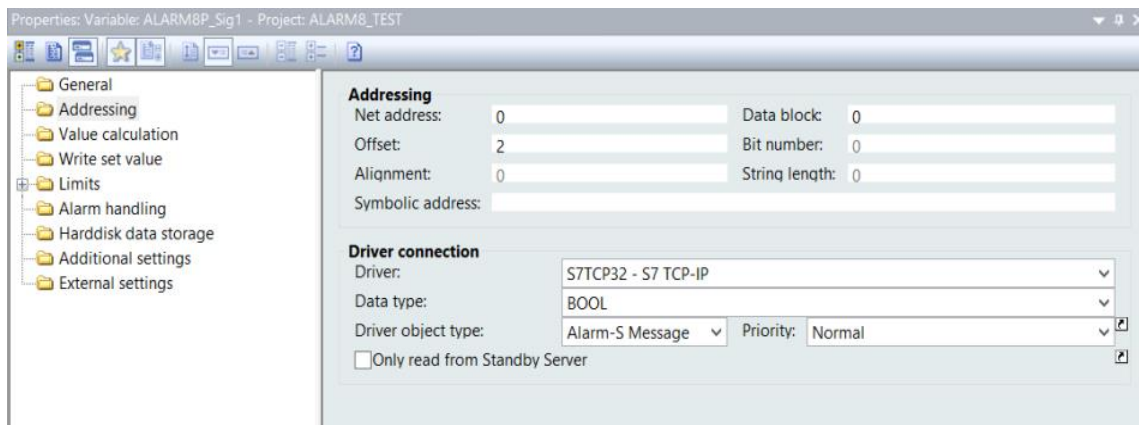
► **ALARM8P_Sig1**:

- **Offset**: 2 (=message number from S7)

- **Data block**: 0 (=index of the signal on the **ALARM_8P** block)

► **ALARM8_Sig2**:

- **Offset**: 2 (=message number from S7)

- **Data block**: 1 (=index of the signal on the **ALARM_8P** block)

► **ALARM8_Sig3**:

- **Offset**: 2 (=message number from S7)

- **Data block**: 2 (=index of the signal on the **ALARM_8P Block**)

ALARM-8P ASSOCIATED VALUE

▶ **ALARM8P_Attribute1**:

- **Offset**: 2 (=message number from S7)

- **Data block**: 0 (=index of the signal on the **ALARM_8P** block)

- **Bit number**: 0 (bit addressing from the 32-byte Dump)

▶ **ALARM8_Attribute2**:

- **Offset**: 2 (=message number from S7)

- **Data block**: 1 (=index of the signal on the **ALARM_8P** block)

- **Bit number**: 32 (bit addressing from the 32-byte Dump)

▶ **ALARM8_Attribute3**:

- **Offset**: 2 (=message number from S7)

- **Data block**: 2 (=index of the signal on the **ALARM_8P Block**)

- **Bit number**: 64 (bit addressing from the 32-byte Dump)