# zenon

by COPA-DATA

# zenon manual

## Project efficiently with the help of zenon

v.8.10

**COPA-DATA**

# 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 Project efficiently with the help of zenon

Is it the first time you are using zenon?
Here you find information on how to use zenon efficiently. Even if you already have experience with other process control systems, we recommend that you look at this chapter. You´ll learn which basic ideas are behind zenon and how it works; why and how zenon helps you work easier and faster.

## AN OVERVIEW OF ZENON

zenon consists of Editor and Runtime.
Projects are created in the Editor. Operation and observation is carried out in Runtime. Editor and

Runtime are compatible between versions from version 6.20 onwards. A project that was created with version 6.50, for example, also runs on Runtime with version 6.22 and vice versa.

zenon mainly differs from other systems by:

▸ Simple operation (on page 6): Many actions can be achieved with a few mouse clicks; assistants and wizards make work easy and fast. You can work using drag&drop with the mouse or the keyboard - as you like it.

▸ Practical object orientation (on page 8): You work consistently with objects for which the parameters can be set quickly, instead of laborious programming. You also use objects that have been defined once throughout projects, benefit from inheritance and lightning-quick changes to properties.

▸ Excellent reusability (on page 18) of objects and elements that have already been configured: Many elements, from individual variables to complete projects can be very easily and effectively reused in zenon.

▸ Integrated network (on page 37): Distributed engineering in the network and 100% redundancy are no longer a big effort. You can create networks by checking the relevant checkbox. It is in precisely this way that you network individual standalone projects into efficient redundant systems.

## TIPS AND TRICKS

You can find useful tips in the forum of www.copadata.com. The best way to learn how get the most out of zenon is to visit the COPA-DATA trainings and workshops that are attuned to your priorities and projects. Your distributor has more information for you - or drop us a mail at sales@copadata.com.

# 3  Simple operation

zenon makes projecting easier through many small amenities and thus increases your productivity. When working with zenon you´ll discover that you can solve many problems according to your preferences. For example:

## FAVORITES

You can define your own favorites in the object properties and in the function dialog. These are always offered to you at the start of the properties list and give you quick access to the properties that you frequently need.

## GENERAL KEYBOARD OPERATION IN THE EDITOR

The Editor can be easily operated using symbols and menus with the mouse, but also with full keyboard operation. The shortcut key combinations from Windows also work in zenon. For example:

- **Ctrl+C** and **Ctrl+V** for **copy** and **paste**
- **Insert key** creates new objects
- **Delete key** deletes highlighted objects
- **Moving mouse while pressing the mouse button:** Navigating in views
- **Esc key** cancels an action
- **Backspace** jumps in the directory tree to the next higher level
- **Ctrl+Z** undoes the last action, with the exception of database operations

You can read more about keyboard operation in the keyboard shortcuts chapter.

## DRAG&DROP

In the Editor, it is possible to make worksteps easier by using Drag&Drop. For example:

- Sequence for changing menu entries
- Link fonts to dynamic elements such as universal sliders, clocks, combined elements etc.
- Change sequence of structure elements for structure data types
- Change sequence within a script
- Copying or moving functions using different scripts
- Copy scripts by dragging them to the root node

## SMART ASSISTANTS AND WIZARDS FOR COMPLEX TASKS

The assistants and wizards integrated in zenon help you to create a base project in no time or to purposefully make complex adjustments. For beginners, we recommend the assistants for the universal slider, the combined element and archiving.

## TOOL TIPS FOR ELEMENTS AND PROPERTIES

If you move the mouse pointer to above a display element (such as a text button) or a property with a linking (such as a function linked to a text button), a tool tip appears. It shows information about the linked elements, like the linked function and its parameters.

## ENHANCED DRAWING POSSIBILITIES

The graphics editor offers you a wide range of functions that make the imaging of equipment easy for you and also give you many design options at the same time. From step-free zoom through to scaling by click up to element corner points and automatic distribution of elements according to rules. You can find details in chapter Screens.

## JUMPING TO LINKED ELEMENTS

With all graphic elements, with functions, variables and also with other elements, you have the option to jump directly to the linked elements. For example, you can go from a display element directly to a linked element with a mouse click, from the variable directly to the data type used, and from there even further to the unit used. If necessary, you can also jump from all these steps back to the initial element easily with the click of a mouse.

# 4  Object-orientation

COPA-DATA sees the topic of automatization object-oriented and pragmatically. That´s how zenon works; furthermore, it can be largely automatized itself. You don´t need a programming language for zenon. You do not have to program one single line of code. Instead, you assign objects with properties using the mouse. That works not only with individual objects but also centrally for a whole project and across objects, too.

If you´re now worried that you can´t individualize zenon and are totally dependent on the object properties, the good news is: Of course you can individually adapt the design of zenon. To do so, use preferably VBA or the VSTA (.NET) development environment. Additionally, zenon offers many interfaces to communicate with hardware and software and boasts zenon Logic with an integrated SCADA logic.

## THREE IMPORTANT BASIC PRINCIPLES THAT HELP YOU WITH ZENON

Three central principles make projecting with zenon easy, reliable and easy to maintain:

1.  Parameterizing instead of programming (on page 9):

    Easily set the required parameters instead of programming or adapting scripts.

2.  Global / central instead of local (on page 10):

    Define objects once and reuse them over and over, also across projects, instead of writing or copying scripts over and over again.

3.  Object-oriented parameterization: (on page 11)

    Use all advantages of object-oriented thinking.

## 4.1 Parameterizing instead of programming

How do you profit if you parameterize projects instead of programming them?

You dispense with code that is prone to errors and gain flexibility, clear structures, speed and high reusability. Four items are particularly in favor of parameterization:

1. **Predefined modules save time**
   zenon contains many predefined modules. Instead of programming scripts yourself, you choose the suitable module and configure it by selecting the desired properties and parameters. That allows you to create executable projects in no time.

2. **The product from the manufacturer is already mature**
   the designer concentrates on his projects. The software is developed and tested by the manufacturer, designers don´t have to write their own code. Therefore it´s very easy to operate zenon - also for employees with no experience in software engineering.

3. **Easy maintenance and adaptation of the project**
   zenon works strictly object-oriented. Combined with the philosophy "setting parameters instead of programming", that means: simple maintenance and adaptation For changes and monitoring, you only have to verify or change the properties of the individual objects instead of reading and adapting long code lines. In the lifetime of the project that ensures

   ▶ Overview: Projects can be reproduced even some years later, even by new employees.

   ▶ Security: Error-prone scripts and programming is avoided.

   ▶ You save time: Projection, adaptation and maintenance can be realized a lot quicker.

4. **Easy adaptation for different machines**
   zenon can be adapted easily and swiftly to new or changed machines. Project adaptations can be done with just a few mouse clicks.

**ZENON ADAPTIONS**

Parameterization prevents errors in scripts from being copied again and again. It prevents tedious changes in many individual scripts if you just want to adapt little things. But that doesn´t mean you have to be inflexible. zenon, too, gives you opportunities to individually expand and adapt the system.

▶ **VBA/VSTA:**
With the integrated script language Visual Basic for Applications or C# and VB.NET, you can execute any code in the Runtime. This allows the execution of automation tasks, logical tasks and interfacing tasks like database access.

▶ **zenon Logic:**
This completely integrated control environment allows you with its SCADA logic to use all the five IEC 61131-3 languages. With it, you can complete all calculations and solve all logic problems.

▸ **PCE:**
The integrated Process Control Engine allows you to use VB-Script and Java Script for automation tasks.

▸ **ActiveX Controls:**
You can integrate standard controls like Flash Player, Acrobat Reader, etc. in zenon. Of course, the interface is disclosed so you can include your own ActiveX Controls.

▸ **WPF-Elements**
The direct integration of **Windows Presentation Foundation** format allows you to integrate graphical elements like for example a button, a pointing device and many more things in this new graphics format directly in zenon images. These elements can be freely linked with variables and functions, which will allow you to implement any graphical effect you like.

▸ **COM-interface:**
You can access zenon via the completely disclosed COM-interface from outside. For communication needs, different programming languages like C++, .NET, Delphi and so on are provided.

## 4.2 Global and central instead of local

Properties can be defined in many ways in zenon, depending on the task:

▸ directly on the object

▸ Central

▸ globally

Of course, in zenon you can adjust all settings directly at the object. But that is not always the smartest solution.

### DEFINE PROPERTIES CENTRALLY

Often it makes sense to define settings only once centrally. This way, new objects offer the desired properties from the beginning. Changes are really easy: The desired property is changed only in one place centrally; all concerned objects adopt it automatically. That means only one click instead of many lines of code or many mouse clicks.

These central settings can be found easily from any position of the editor. Every object that gets settings directly from a central location offers the option to show them directly. You simply follow a link and see immediately how these elements are linked.

But you can´t only define properties centrally for a project, you can also define them globally.

## PROJECT GLOBALLY

zenon allows you to work with more than one project at a time - and thus saves your most precious resource: Working Hours For example, it is really easy to use the same fonts in all projects. Once defined, properties like font or font size are instantly available for all projects of a work area. If you need another font for all projects, it'll take you only some seconds to change it in the global project. What if you want to provide a project with a different font? Well, then you change this font centrally in that project.

> ☀ **Information**
>
> In the global project, use a numbering system that is different from the one in the single projects. zenon always prefers the local settings if the numbers are identical.
>
> And: Use own names for font lists that you keep consistent in the global project and in the individual projects. In doing so, you ensure that all fonts are found if the language is changed.

## OVERVIEW THE RELATIONS OF OBJECTS:

Additionally, you can get an overview of your projects and object relations with:

▶ **Cross reference list:**
It shows cross references in a project and enables you to easily search for the usage of variables and functions.

▶ **Wizard for documentation**
It automatically creates complete project documentation. You can configure exactly what you want to be included, down to every single property.

# 4.3   Object-oriented parameterization

"Object-oriented parameterization" is the fundamental philosophy of zenon. This philosophy arranges work clearly and saves time for creation and management of variables.

Each variable in zenon is based on two elements:

▶ Driver (on page 12)

▶ Data type (on page 13)

## 4.3.1 Drivers

Drivers are not directly integrated in zenon, they are implemented via a driver object type. This determines which area on the controller is to be addressed.

### DRIVER OBJECT TYPE

There are many different driver object types, for example standard PLC markers, database blocks, inputs, outputs, counters and also other special types such as alarm or driver variables.

The driver object type determines:

▸ The area on the controller

▸ Which granularity the driver has in this area

▸ Which data types can be created on the area

**Hint:** Not any data type can be created in any area. There, when creating a variable, you should always choose:

▸ The driver first,

▸ then the driver object type and

▸ lastly the data type

### EXTRA INFORMATION: GRANULARITY

Granularity is primarily important for numerically-addressed controllers such as the Siemens S7 for example. Not every controller has the same grid and not all areas in a controller have the same grid.

For example, the datablock area of the Siemens S5 is word-orientated, but the marker are is byte-orientated. This means that the smallest unit that can be addressed is a byte or a word. If you want to write a bit in such areas, it is necessary to read at least a complete byte, mask out the corresponding bit, change it and then write the complete byte/word to the controller again.

The same information is also required when addressing the areas, called offset. You start at zero and count on in byte or word steps. If you use automatic adressing (on page 17)in zenon granularity is automatically considered. The granularity does not depend on the driver, but on the driver object type.

### DRIVER VARIABLES

The driver variables are offered by each driver and provide many advantages for project configuration. They do not communicate with the controller, but read an internal save area in the driver, which primarily consists of statistical information. However special functions can also be controlled with variables, such as telephone numbers for modem connections or commands for dialing or hanging up.

You can find more detailed information on this in the driver variables help chapter.

A dBase file with the most important driver variables is included on the installation medium of zenon. You can therefore import variables for each driver instead of creating them manually.

## 4.3.2  Data types

Data types are the heart of object-oriented parameterization. As you cannot change anything in the driver -object types, they cannot be used for object-orientation. The data types, on the other hand, offer many possibilities.

We distinguish between three types:

- ▶ Simple data types
- ▶ Structure data types
- ▶ Structure elements

## 4.3.2.1 Simple Data Type

Simple data types are always IEC data types, that means the data types defined by the IEC in the 61131-3 standard, like BOOL, INT, USINT, UDINT, STRING, WSTRING, etc. The length of the value range is defined.

**Examples:**

- ▶ BOOL: *1* Bit
- ▶ INT: *16* bit unsigned
- ▶ UINT: *16* bit unsigned

The data types in zenon allow not only IEC data type, a lot more properties are available. These have the same identification as the properties of the variables:

- ▶ Identification
- ▶ Unit
- ▶ Value range
- ▶ Limit Values

Why this "dual data retention"?

The background is as simple as it is practical:
Just as a dynamical element takes over the font type and size from the linked font, the variable also takes over the value calculation, the unit and the limit values from the data type it is based on.

## OBJECT ORIENTATION

In contrast to the linking of dynamic elements and fonts, this involves an object-orientated approach: The data type object passes its properties on to the variable. The difference?

Linked/derived properties can be separated or overwritten for the variables. This works for each property individually but also for all properties at the same time. You can, for example, take on the unit from the data type centrally, but directly overwrite the identification or the address for the variable. This works as follows:

▶ Change the relevant value of the variable, then the link is separated.
You can verify that by looking in the properties: The check is no longer there.

You can use the following functions via the context menu in the project properties:

▶ **Deriving a property from the data type:**

If you have separated a linking, you can reestablish this again.

▶ **Unlink a property from the data type:**

You can separate an individual property without having to change the value. If you subsequently change the data type, you can be certain that the variable is not influenced by this. You can set whether the inheritance concept takes effect or not.

▶ **Link all properties with datatype**
Links all properties in the properties node to the data type.

▶ **Unlink all properties from datatype**
Disconnects all properties in the properties node from the data type.

To open the context menu, click on a property in the project properties with the right mouse button.

As a particular help, you can also accept or separate all properties from the data type with a mouse click. You can thus restore the original state (everything derived) very quickly or rescind this. How does this all help in practice?

Particularly high flexibility: If you have many variables with the same limit value, such as an alarm at value 1, then set this for the data type. You the set the limit value text and other optional parameters separately for each variable. You thus save the creation and maintenance of the limit value for each individual variable and your engineering output increases considerably. The data type BOOL, created as standard, also has a limit value at 0 and at 1!

If you do not even have a limit value for each bool variable, you do not however need to take extra care for each variable and turn the limit value off again. Instead, you can conveniently use a feature of zenon and create your own BOOL data type. Proceed as follows:

▶ Simply leave a standard data type as it is

▶ Create a new data type (via the **New simple datatype…** entry in the project manager in the context menu of the **Datatypes** entry or via the entry in the toolbar)

▶ Give it a name, e.g. *MyBool*

- ▶ A list with the existing data types is offered
- ▶ Select *standard bool*
- ▶ All properties of the new data type are automatically taken from the old one
- ▶ Delete limit values for *MyBool*

You then create all bool variables that are not to have limit values for this new *MyBool* bool data type – and the limit values you don't want have already disappeared. Those that are wanted are retained however.

You cannot of course create just your own bool data type, but as many as you want. All variables that need identical properties over wide ranges get their own data type. If a property changes, change this centrally and all derived variables are also changed automatically – with the exception of those for which the linking was changed.

## 4.3.2.2 Structure data types - structure elements - structure array

### STRUCTURE DATA TYPE

Structure data type means a group of data types that are precisely defined in terms of their order and arrangement. For example, the structure motor can consist of the elements of actual speed and power consumption.

A structure can also be nested, whereby as many hierarchy levels as desired are possible.

With structure data types, the variable household can be be structured in precisely the same way as it actually exists in the controller or logically.

### STRUCTURE ELEMENT

A structure data type is actually just a hull that bears the name of the structure but does not have any properties of its own.

These first come with the structure elements that are added to the structure data type. In doing so, a distinction is made between:

- ▶ Linked structure data type:

  Reference to a pre-existing data type. All properties are taken on, with the exception of the name.

  Advantage: If the structure element has many properties of a pre-existing data type, these can easily be reused. It is of course still possible to change each of the variable properties or separate them from the data type.

- ▶ Separate embedded structure data type:

Allows individual addressing settings. This new data type is only valid within this structure.

> ### 💡 Information
>
> Advantage of object orientation: If the structure changes at any point, it is then sufficient to change the structure data type in order to automatically amend all variables that are on this structure.
>
> Changes are possible for all forms:
> - simple properties like the unit of the data type
> - complex properties like adding or deleting a limit value
> - Change to the complete structure, such as a change to the sequence of elements or addition or deletion of individual structure elements in a structure data type

## STRUCTURE ARRAY

If it is not just one machine, but many, the structure array is used.

For example: 100 Pumps instead of 1 pump.
You simply change the structure variables to a structure array. 100 variables thus become 100 structure variables with the click of a mouse.

If we, for example, have to create 12 individual variables for each pump, we need a total of 1200 variables. The method of "setting parameters in an object-orientated manner" allows this with a few mouse clicks. In addition there is the advantage that all variables have already been preset. Each individual variable already provides all properties that it needs, such as unit, value determination, alarms, CEL entries etc.

> ### 💡 Information
>
> Structure data types are also suitable for the reuse (on page 30) of variables.

## 4.3.2.2.1 Example pump:

## INITIAL SITUATION

A pump consists of two motors. Each of these motors has some variables like:
- *Actual speed (RPM)*

▸  *Power consumption*

▸  *Output*

▸  etc.

Each motor has a slider with the following variables:

▸  *Nominal speed (RPM)*

▸  *P-share*

▸  *I-share*

You build this constellation as directed:

1.  Create a structure **slider** from one data type each for

   ▸  *Nominal speed (RPM)*

   ▸  *P-share*

   ▸  *I-share*

2.  Create a structure **motor** with these data types each:

   ▸  *Actual speed (RPM)*

   ▸  *Power consumption*

   ▸  *Output*

   ▸  etc.

3.  Adopt the **slider**   in the structure **motor** .

4.  Finally, create the pump structure and integrate the motor into it.

   As we have two motors in our pump, you simply add the motor data type to your **pump** structure twice.

5.  Create a variable that is based on this structure data type:

   ▸  Create new variable

   ▸  Choose structure data type **pump**

   The individual elements of this structure variable are called Structure elements (on page 15) .You can use each of these elements everywhere in zenon- in screens, as alarms, in archives, in recipes, etc.

### 4.3.3  Tips for addressing and import/export

Practical tips for:

▸  Addressing

▸  Export/import

## ADDRESSING:

With numeric controllers, you have a choice of whether addressing takes place automatically or semi-automatically. The properties described here can also be reached by means of VBA/VSTA and can thus be used for automatic project configuration.

## AUTOMATIC ADDRESSING:

Offset and - if required - byte and bit address are automatically calculated on the basis of the position of the structure elements; with arrays it is throughout the complete array. If you have created the same structure in the PLC and in zenon, everything works fine and you do not have to care about addressing.

## SEMI-AUTOMATIC ADDRESSING:

You already issue individual start addresses for the data type. The further addresses are then calculated using these addresses. You can of course also amend these if required. You can find details in Automatic addressing.

For symbolically addressed PLCs, the same name must be given in zenon and in the PLC.

## XML EXPORT/IMPORT:

Information like data types, structures, inherited properties etc. is also included when importing or exporting. This means that, once defined, you can comfortably export structure data types and structure variables and import them in other projects to reuse them or adapt them as needed.

# 5 Reusing elements

The first time a project is created, you need time for the creation of variables, functions, screens and their links. You can reduce this time considerably in following projects. Because with zenon you have the possibility to easily transfer elements that have been created to other projects.

For example, in many projects you need the standard components of every project, such as screens for the system status or detail screens for hardware components that are used repeatedly (pumps, valves, motors etc.).

The are different ways for you to reuse components:

| Theme | Reuse of |
|---|---|
| Replacing variables and functions (on page 19) | Elements |
| Symbols (on page 24) | Screens |

| Theme | Reuse of |
|---|---|
| Structure data types (on page 30) | Variables |
| Reaction matrices (on page 31) | Variables |
| Global project (on page 32) | Central elements of a project |
| XML (on page 33) | Project parts by means of import and export |
| Wizards (on page 35) | Screens and elements by means of import or individualization |
| Reusing projects (on page 35) | Projects |

### 💡 Information

To be able to reuse elements efficiently, ensure when you create variables, functions and screens that they are clearly named (on page 19).

## 5.1   Replacing variables and functions

Variables and functions that are stored as dynamic elements can be replaced in an automatic manner. This can take place at different places:

▸   Replacing linking in a screen (on page 21)

▸   Replacing linking for screen switching (on page 22)

▸   Replacing indexes (on page 23)

▸   You can also find more about replacement of variables and functions in the Screens manual in the Replacing linking of variables and functions section.

### 5.1.1  Naming conventions

To be able to replace variables and other elements securely, the naming should be systematic and standardized if possible. You therefore support not only the reusability, but also maintenance and reverse engineering.

Different systems support you with systematic naming.

**FOR EXAMPLE: ENERGY INDUSTRY**

<u>Germany</u>

▸ KKS (Kraftwerk-Kennzeichen-System - Power Plant Classification System), for details (in German), see http://de.wikipedia.org/wiki/Kraftwerk-Kennzeichensystem (http://de.wikipedia.org/wiki/Kraftwerk-Kennzeichensystem)

▸ DIN 6779 (Kennzeichnungssystematik für technische Produkte und technische Produktdokumentation - Classification System for Technical Products and Technical Product Documentation), for details (in German), see http://de.wikipedia.org/wiki/DIN_6779 (http://de.wikipedia.org/wiki/DIN_6779)

▸ Equipment Classification System, for details (in German "Anlagenkennzeichnungssystem"), see http://de.wikipedia.org/wiki/Anlagenkennzeichnungssystem (http://de.wikipedia.org/wiki/Anlagenkennzeichnungssystem)

<u>Internationa</u>l

▸ KKS (Power Plant Classification System), for details, see http://en.wikipedia.org/wiki/KKS_Power_Plant_Classification_System (http://en.wikipedia.org/wiki/KKS_Power_Plant_Classification_System)

Such standards exist for all industries. It is recommended that their naming convention is used.

**KKS EXAMPLE:**

Variables are to be named in accordance with the KKS in an energy project.
A corresponding variable with the label **C01_MDY10-QA001_QA07** indicates:

▸ Wind energy equipment **C01** (row **C**, no. **1**)

▸ Wind turbine control **MDY10**

▸ Power part **QA001**

▸ Power protection **QA07**

## 5.1.2  Replacement possibilities

Replacements can be used at different points in a project:

▸ Replace linking in a screen (on page 21): Screens are copied and the links are replaced in the copied screen.

▸ Replacing linking for screen switching (on page 22): Only one screen is used for different PLCs; the links are adapted when switching.

▸ Replacing indices (on page 23): Replacement of variables in a process screen using the value of index variables.

## 5.1.2.1 Replace linking in screen

If variables are attached in a screen, these can be easily replaced by means of a replacement dialog. Requirement:

- ▶ Clear naming of variables (on page 19)

With this, screens that have been created can continue to be used by copying & pasting. The replacement is started using the context menu:

- ▶ Right-click on the screen element
- ▶ Click on **replace linkings**
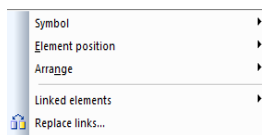


The dialog for replacement is thus opened.

## EXAMPLE

In our example, the project has a number of variables from different parts of the equipment. The following are in the process screen:

- ▶ 10 variables that come from the **MDY10** wind turbine control, with dynamic elements linked
- ▶ Buttons with screen switching to different areas of **MDY10** present

Using copy & paste, the person configuring the project intends to reuse the screen in its exact form for wind turbine control **MDY11** and to replace the variables or functions by the corresponding ones from the new PLC. To do this:

1. The dynamic elements on which variables and functions are linked are highlighted

2. The replacement dialog is opened
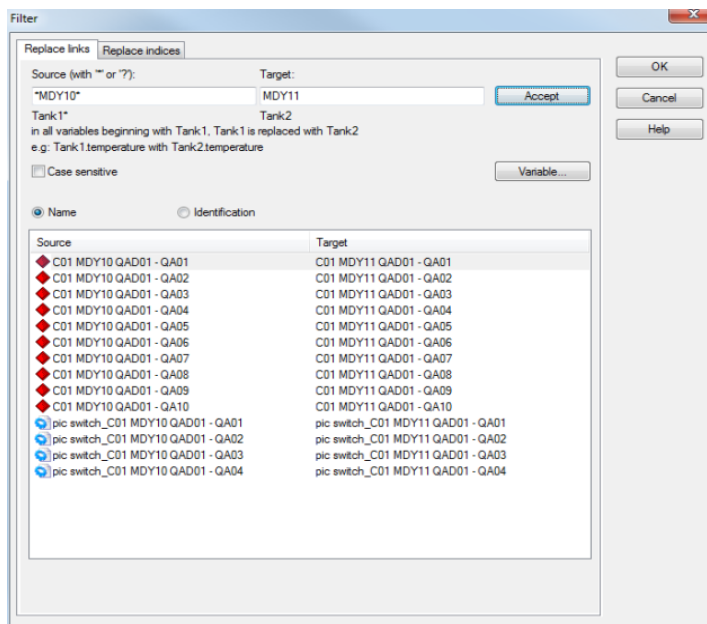


3. In the **Source** field, the equipment identification *MDY10** is entered
   (the first and last * characters are wild cards)

4. *MDY11* is entered in the **Target** field

5. Replacement is carried out with **Accept**, the replace and the dialog is closed with **OK**

The screen can now be used for the new PLC.

## 5.1.2.2 Replace linking for screen switching

With this method, it is always the original screen that is used and called up in the Runtime with different variables and functions. The screen contains different variables and functions, as in the "Replace linking in the screen (on page 21)" example. Replacement is carried out when switching. To do this:

1. Several screen switching functions are configured to this screen

2. The dialog for replacement (on page 21) is offered when the function is created

3. Each function contains its own replacement process
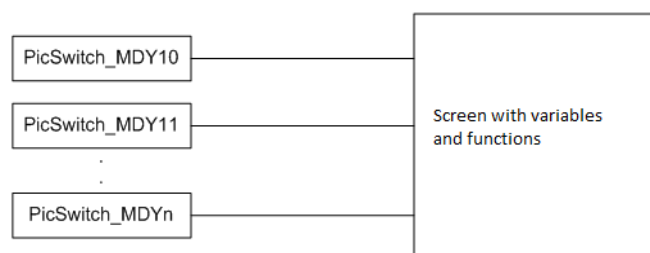


A variable for this, which only comes with a screen switching: Replace indices (on page 23) for arrays.

## 5.1.2.3 Replace indices

The replacement for variables can be carried out using index variables in arrays.

As with Replacing linking for screen switching (on page 22), the screen is only used once. The linked variables are replaced when switching. Screen switching is only configured once (different to replace linking in the screen (on page 21)) and can be reused more than once using an index variable.

### EXAMPLE

Screen switching is carried out on a process screen that contains 10 variables of the wind turbine control **MDY10**. The aim of the person configuring the project is to reuse this screen 1:1, because the wind turbine controllers **MDY11**, **MDY12** and **MDY13** have the same number of variables.
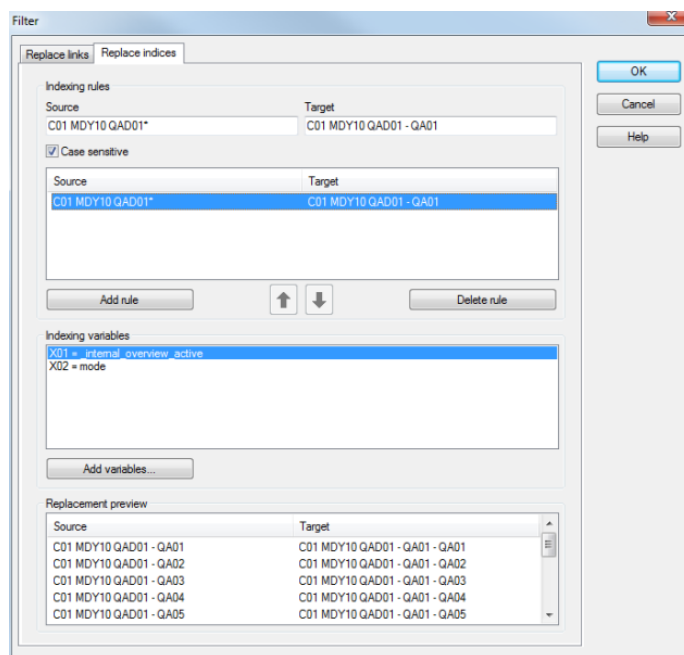


The indexing rule:

  ‣ Source: C01 MDY10 QA001*

  ‣ Goal: C01 MDY{X01} QA001

Has the following effect:

  ‣ With screen switching, the function is informed that it must use the variable value of **X01 Index wind turbine control** in the variable name for screen switching in Runtime.

    Example: If this variable has the value of 12, the process screen with all variables of the wind turbine controller **MDY12** are displayed when screen switching is executed

## 5.2 Symbols

Symbols offer great potential for reuse. Symbols can be built up in a very complex manner and support inheritance. Symbols can be embedded or linked. With linking, modification in one place is sufficient to update all screens that use this symbol. Symbols can also be linked to other symbols.

### EXAMPLE OF BUTTON BAR

A symbol (empty right corner) is linked as a basis in all buttons.
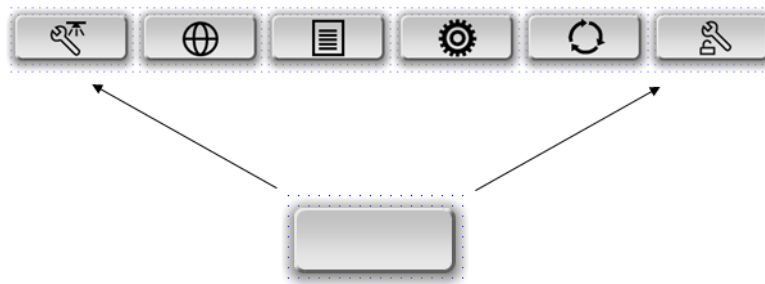


Free-access properties (on page 24) make it possible to issue each button with different graphics. If the form or color of the buttons is changed, only the "empty button" symbol needs to be modified. All buttons automatically take on the new form and/or color.

### 5.2.1 Free access properties

Symbols pass on their properties to the objects in which they were linked. However, individual properties can be released from inheritance. The displayed graphics in our button for example. If inheritance is released, this property can be set individually for each object. Changes to this detail in the initial element no longer influence the other objects.

### REUSING SCREENS

The combination of symbols and released properties can be an effective solution for the reuse of process screens.

### EXAMPLE

There is a central dialog for setting parameters in a project, which is to be used for various setting of parameters. Because this is used in various areas, there is also a requirement that certain adaptations (such as background colors, screens ...) should be possible.

**<u>Copy and paste:</u>**

1. Create a process screen that contains this dialog for setting parameters
2. Duplicate the screen using copy & paste

3. The process screens that are duplicated can be adapted to the requirements graphically

**But:** Inheritance is not possible here. If the person configuring the project subsequently wishes to make changes centrally (in relation to the basic structure of the dialog for setting parameters), they must then drag these to each individual process screen.

## Approach using symbols with released properties:

1. The dialog for setting parameters is created using a symbol

2. Elements that have to be adapted for different parameter points are decoupled via released properties from the inheritance concept

3. The symbol is linked, positioned and adapted in the corresponding screens.
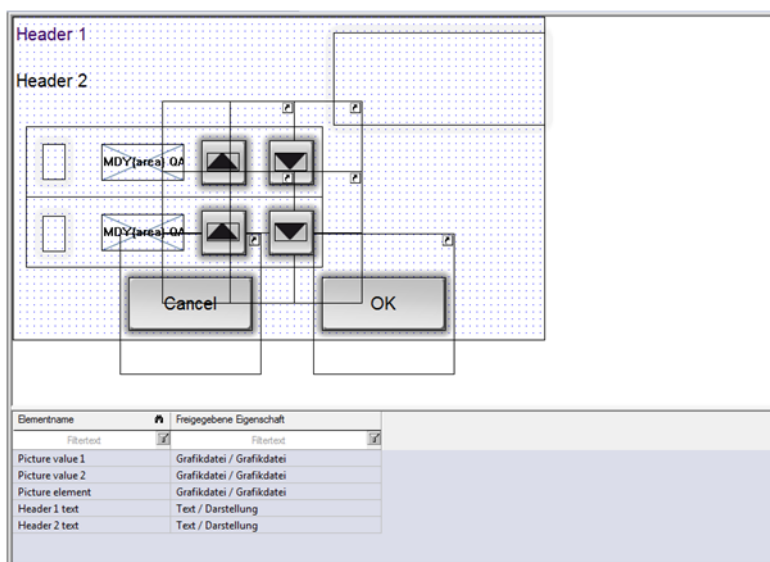


4. If the person configuring the project decides to change the general appearance of the dialog, they make this change centrally in the symbol. This change then affects all items in which the symbol is linked.
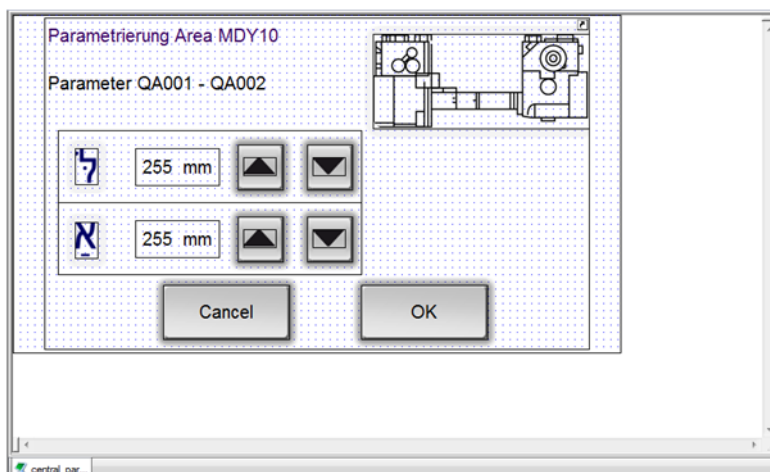
5.   A global symbol can be adapted individually using released properties and corresponding replacement (on page 26).
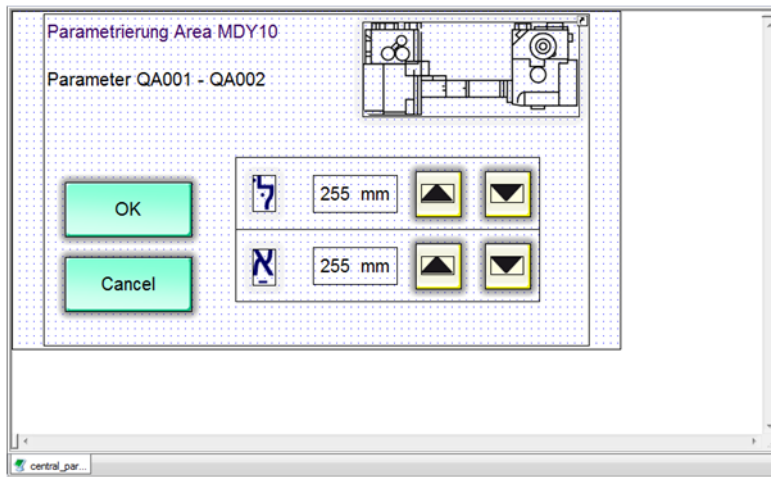


6.   In this case, the person configuring the project has decided to change the general graphical user interface of the dialog for setting parameters. This change is made centrally at the "Setting parameters" symbol. The individual properties of the individual parameter dialogs remain unchanged; only the properties that are contained in inheritance are contained.

## 5.2.2 Replacement with symbols

The process described in the Replacing variables and functions (on page 19) section can also be carried out for symbols. Here too, well-thought naming (on page 19) of the objects is a requirement.

When carrying out replacements for symbols, a distinction must be made between two different approaches: A symbol from a library (project/global) can be

▸   inserted into a screen as an element group

▸   linked into a screen

Depending on the approach, different dialogs are switched for the replacement of variables/functions.

### ELEMENT GROUPS

Here, variables are replaced directly via the replacement dialog. The dialog offers the possibility to select variables via a variable selection dialog. Using symbols in this manner is not ideally suited for reuse.

### SYMBOLS

With this process, work is carried out using replacement strings. These strings are replaced with the corresponding variables during compilation. It is not possible, via the dialog called up, to make

replacements by means of a variable selection dialog. The person configuring the project works using individual replacement strings here, which are separated from each other by a semi-colon (;).
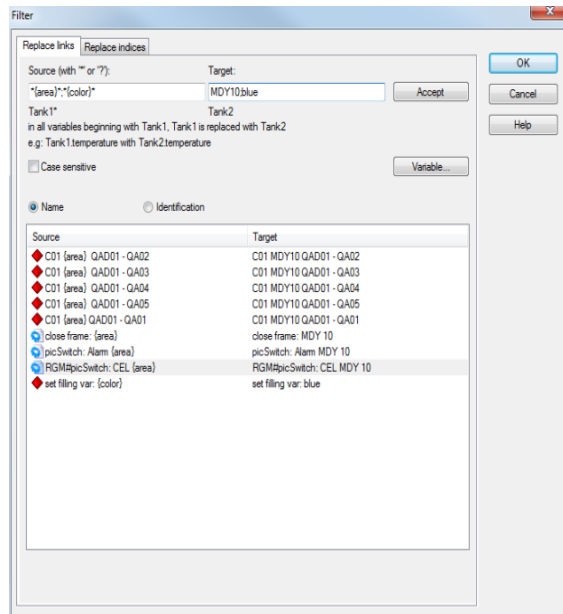


The symbol behind this replacement dialog can now be reused in various parts of the equipment. This example assumes that similar variables and functions are used in each equipment area (only differentiated by the equipment abbreviation in the name) and that these can be replaced quickly and with a clear overview.

You can find more about replacement for symbols in the Screens manual in the Symbols section.

## 5.2.3 General symbol library

There are several possibilities for the administration of symbols. The symbols can be administered in:

▶    The symbol library of the local project

▶    In the symbol library of the global project

▶    In the general symbol library

| Symbol Library | Property |
| --- | --- |
| **General symbol library** | Symbols are available in all projects. The **general symbol library** node is located in the project manager below the currently-loaded projects.<br><br>Label when linking in the screen: **[symbol group]/[symbol name]**<br><br><u>**Rules:**</u><br><br>▶    The dialog for selecting variables offers all projects of the workspace for linking.<br><br>▶    These symbols are saved in the zenon program folder and only |

| Symbol Library | Property |
|---|---|
| | updated when the Editor starts. These symbols are not saved during project backup. |
| | ▸ Interlockings and aliases for ALC cannot be configured. **Attention:** If symbols that contain interlockings or aliases are added, these settings are removed. |
| **SYMBOL LIBRARY IN THE GLOBAL PROJECT** | Symbols are available for all projects of the workspace. |
| | Label when linking in the screen: **Global project_[Symbol name]** |
| | <u>Rules:</u> |
| | ▸ The symbols copied into the symbol library of the global project retain their variable linking without changes. |
| | ▸ The variable dialog offers all projects of the workspace for linking. Linked variables are placed in front of the name of the respective project. |
| | ▸ The name of the **Variable** can be amended in the properties window. This way, for example, the prefix can be deleted with the project origin. |
| | ▸ The symbols are also backed up when a project is backed up. |
| | ▸ Interlockings and aliases for ALC cannot be configured. **Attention:** If symbols that contain interlockings or aliases are added, these settings are removed. |
| **Symbol library in the project** | Symbols are only available in the current project. |
| | Label when linking in the screen: **[Symbol name]** |
| | <u>Rules:</u> |
| | ▸ The symbols are saved in the project folder. The project symbol library is in the current project in the **Screens** node and is backed up together with project backup. |
| | ▸ Interlockings and aliases for ALC can be configured. These properties are also retained when symbols are added. |

## COPYING OF SYMBOLS BETWEEN LIBRARIES

Symbols can be copied with **Drag&Drop** .
For this, the following applies:

▸ Only symbols of the first level of the tree can be copied.

▸ Groups and folders cannot be copied.

▸ If there are already names of symbols in the target library, the added symbols are automatically renamed.
Renaming is carried out by adding an underscore and an ordinal number, for example: **Calibrator_1**.

▸ Copying between libraries is only possible to a limited extent.

Overview of copying between symbol libraries using **Drag&Drop**:

| Copying using Drag&Drop possible | General symbol library | Symbol library in the global project | Symbol library in the project |
|---|---|---|---|
| **General symbol library** | -- | + | -- |
| **Symbol library in the global project** | -- | -- | + |
| **Symbol library in the project** | -- | + | -- |

**Key:**

▸ **+**: copying possible

▸ **--**: copying not possible

## EXAMPLE

A central configuration point compiles a collection of symbols that are to be used by the configuration points distributed around the world in their individualized projects. For example, complex symbols that display important parts of process screens are displayed. This collection of symbols is in a folder or a file, for example "*Global_Used_Symbols.SYM*". If central graphical changes are carried out to this global collection of symbols, "*Global_Used_Symbols.SYM*" is simply sent to all configuration points and replaced there. Individual adaptation is carried out by means of released properties (on page 24) and replacement (on page 26).

It is not always necessary to distribute the whole collection of symbols. An XML export/import (on page 33) can also be used to distribute individual selected symbols.

> ⚠ **Attention**
>
> There must be clear rules when the global symbol library is used: Local configuration points must not change the symbols from the symbol collection. Each change would be overwritten at the next update.

## 5.3   Structure Data Type

Variables can be reused using structure data types. Complex structures can thus be created and the advantages of inheritance can be used.

### EXAMPLE

The "motor" structure data type consists of three elements:

- ▸ Status
- ▸ Speed
- ▸ Temperature

| motor | Structure datatype | 0 |
|---|---|---|
| status | Structure element | INT/<embed... |
| rotati... | Structure element | INT/<embed... |
| temp... | Structure element | INT |

The inheritance type of the elements is different:

- ▸ **Status**" and "**Speed**" are derived from the *INT* data type, but their inheritance was broken off by the embedding.
  This means: Subsequent updating of the *INT* data type does not have an effect on these two elements.

- ▸ "**Temperature**" also comes from the *INT* data type, the inheritance is intact.
  This means: Subsequent changes to the *INT* data type also has an effect on the settings of this structure element.

If, for example 20 motors with a structure that is always identical are configured, variables based on the self-created "motor" data type can be created. This can, for example, happen by creating an array of this data type. The graphic display is triggered by means of a symbol in this example.
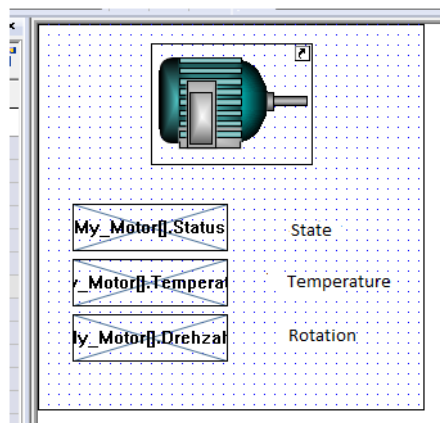


When linking the symbol in a process screen, the placeholder variables of the symbol are replaced by those of the self-created structure data type. Ensure that you have well-thought out naming (on page 19) from the start.

If a process screen is created as an overview of all motors, the inserted symbol can be easily duplicated by copying & pasting. Only the target variable needs to be replaced with the corresponding sequence number on each symbol:
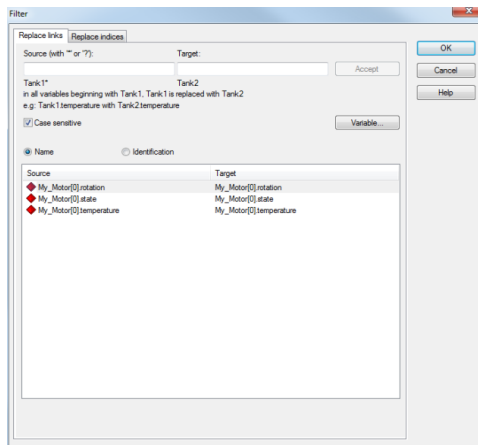


If limit values for the "speed" variables are to be added at a later time, you benefit from the existing inheritance display between the self-created data type and the variables that are created as a result. For the planned change, a new limit value is created for the "**speed**" structure element of the separate "*motor*" data type. This change made at a position has an effect on all 20 motors that have been created.

However the person configuring the project is free to undo this inheritance relationship. The limit value for "**speed**" can be changed for one of the 20 motor variables that have been created.

**Note:** Individual properties or all properties of a variable are disconnected from the data type.

---

### 💡 Information

You can also read tips on the use of structure data types in the Structure data types - structure elements - structure array (on page 15) section and in the Variables manual in the Structure data types section.

---

## 5.4   Reaction matrices

Reaction matrixes ensure that a variable is the same throughout the project. In contrast to limit values, they have a central approach: A reaction matrix is configured once and then assigned to any number of variables. All variables that are linked to the reaction matrix react in the same way. The benefit: central and simple maintenance.

## 5.5 Global project

With a global project, standards can be defined throughout a project. It is thus possible to define certain modules or elements for all sites, whilst local projects stipulate individual parameters.

The following can be configured in the global project:

▸ **Alarming:**

Declaration of the alarm groups, alarm classes and alarm areas to be used globally.

▸ **Equipment modeling:**

This can support the central project configuration, because you can undertake a division into two "plants". In doing so, the project configuration elements that come from the central project configuration point and those that come from the local project configuration point must be configured.

▸ **User:**

If a standard envisages the existence of certain globally-valid users, these can be determined using the global project.

▸ **Files:**

There can for example be graphics as a result of a standard, which must be used in individually-created projects. These can be provided globally.

▸ **Frames:**

Determination of the process screen arrangement and size of the projects to be developed. It is possible, for example, to clearly define in a guideline that in a project, newly-created screens can only be created using templates of a global project. Uniform global graphics are thus guaranteed.

▸ **Fonts and color palettes:**

In order to achieve uniformity with displayed texts and colors, a pre-defined set of font lists or color palettes can be stipulated for global use.

▸ **Language tables:**

If a determined set of terms and statements to be used is configured in the project configuration, it is possible to stipulate these using the defined language tables and to carry out the corresponding translations at the same time.

### EXAMPLE

A central project configuration point defines standards in the ares of color, fonts, templates etc. throughout a global project for all people configuring the project. The global project is distributed to all teams as a project backup (on page 36). They import it into their workspace. A guideline must be used to clearly defined that local project configuration points must not make changes to a global project,

because these are repeatedly overwritten by the input of updates from the central project configuration point.

### EXAMPLE OF EQUIPMENT MODEL
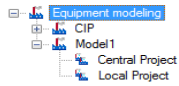


A simple equipment model that only makes a distinction between a central and a local project configuration group. In an individualized project, this distinction could now be used to help for visualization in the Editor. In detail, this would look as though each configured element in the Editor is given a corresponding equipment group assignment.



The existing process screens are displayed after equipment assignment.

The local project configuration point can thus clearly detect which screens are updated by a central project configuration point via the XML import. For the person configuring the local project, it is t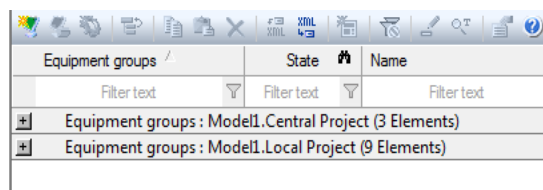hus also clear which screens they must not change. This is because each change to globally-maintained screens is overwritten again on the next update.

Hint: To update a global project, it is not always necessary to create and import a project backup. Individual changes can also be updated locally by means of XML (on page 33) export and import.

## 5.6  XML

Import and Export via the XML interface offers many possibilities for the reuse of project components.

> ⚠️  **Attention**
>
> Existing elements with the same name are overwritten on import.

### DEPENDENCIES ON IMPORT

If several parts of a project are to be imported, there are also more XML files with different content accordingly. In order to accept the content of several XML files correctly, existing dependencies must be taken into account:

If, for example a screen is imported and the variables linked therein do not exist in the project, the linking cannot be restored. In order to create all linking correctly, it may be necessary to carry out the import of a screen or a variable twice.

**Example:** Import of a screen via XML

The most comprehensive variant is to import a screen. In this case, the created XML file must not contain just the screen and the included elements, but also the frame, linked variables and functions.

The XML file needs all content that can be seen in the screen and is thus linked to it directly.

In screen "**A**", a function "**B**" is executed by means of a button, in order to then in turn open screen "**A**":

> The "**B**" function needs screen "**A**" in order for this to be linked in function "**B**".

> The screen "**A**" then in turn needs function "**B**" in order for this to be linked to the button.

Import:

1. First the function

2. Then the screen

3. Then the same function once again
   **Note:** The reference to the screen cannot be established in the function otherwise.

## HARMONIZE PROJECTS THROUGHOUT A FACTORY

XML export/import is also suitable for guaranteeing cross-factory harmonization of projects. If a company establishes that certain certain parts of the project must be designed in a uniform manner throughout the world, then the uniform elements are:

1. Configured centrally

2. Exported to an XML file

3. Distributed as an XML file

4. Applied by each local team as an XML import

Alternatively, export and import can also be carried out by means of a specific wizard.

Because existing elements are overwritten with an XML import, these uniform elements must be named clearly and bindingly. Local changes of these elements are overwritten again on the next import.

## WIZARDS

A zenon-based wizard is offered to support the user when importing. In the wizard, the user can then select what is to be imported using decision-making aids. A possible approach for the wizard is to create an XML library that contains different variants of screens and functions.

**CUSTOMIZATION WITH THE WIZARD**

A wizard can also be used to customize standard functions that are in XML format. One function per XML is imported and then amended to individual requirements by changing certain properties that are not the same for each object. For example, there can be a screen switch function in XML format and this can be used several times by customizing the properties.

## 5.7   Wizards

The reuse of elements and parts of projects can be simplified and supported with wizards. Wizards are primarily suitable for:

▸ **Creation of template projects:**

A wizard makes it possible to create certain standard parts of a project to be created using a few mouse clicks. In contrast to simple input of a completed project backup (on page 36) of the template project, the person configuring the project has more scope for customization here.

For example, a wizard can create different project types regardless of machine.

▸ **Creation of pre-defined parts of a project:**

Wizards can also help to create certain areas in a project.

For example: Creation of a "**motor**" project area.
The wizard creates pre-defined elements for this area. It is thus possible to create corresponding screens, functions, variables etc. with a few clicks of the mouse. In addition to shorter project configuration times, you also get a uniform appearance, because the wizards are programmed by a central project configuration point and used by the local project configuration points.

▸ **Configuration using pre-defined databases or files:**

Project databases or pre-defined files can also support project creation or project expansion. In doing so, the elements to be configured are described in a database or an Excel file (function name, function type, function parameter, variable name, offset ...). These are read off with the help of a wizard and corresponding configurations are undertaken in zenon.

You can find out more about wizards and the creation of these in the Wizards manual

## 5.8   Reusing projects

Projects can also be used in a different way. In doing so, mechanisms from the other chapter of reusability sometimes play a significant role.

Project Backup (on page 36)

Save as (on page 36)

Multi-project administration (on page 37)

### 5.8.1 Project backup

Project backups make it possible to accept complete projects.

For example, a project can be distributed to all teams as a template. These create a project backup and individualize the project.

> ⚠ **Attention**
>
> This method is not suitable for transferring project changes, because all local configuration is overwritten when it is accepted.

To maintain projects that have been created this way consistently throughout all teams, the use of XML exports/imports (on page 33) is recommended.

In doing so, please note that pre-existing elements with the same name are overwritten during import! When configuring a project, the configured elements that can be overwritten must be clearly marked. For example, for the necessary identification, a corresponding addition in the element name or the procedure using equipment groups, see the global project (on page 32) section.

### 5.8.2 Save as

**Save as** creates a copy of a project in the active working area under a new name and with a new GUID.

#### EXAMPLE

Individual projects for different machine types are to be created, based on a template project. To do this:

1. The project backup (on page 36) of the template project is loaded in

2. Several differently-named projects are created in the active workspace using **Save as** (such as **Machine 1**, **Machine 2**, … )

Each machine project has the same initial project situation. However, the projects no longer have any connection to a template project. Changes that are made in a template project are not automatically accepted into the copied projects. However changes to the template can be introduced into individualized projects by means of XML export/import (on page 33).

In doing so, please note that pre-existing elements with the same name are overwritten during import! When configuring a project, the configured elements that can be overwritten must be clearly marked. For example, for the necessary identification, a corresponding addition in the element name or the procedure using equipment groups, see the global project (on page 32) section.

### 5.8.3 Multi-project administration

Projects can also be compiled in a workspace in zenon multi-project administration as part of an integration project. It is thus possible to change a project centrally, whilst local project configurations are carried out in another project.

**EXAMPLE**

A central project configuration point is used to implement graphics reuse in local projects, in order to ensure a uniform user interface. To do this, the central configuration creates a template project that constitutes the graphics basis for all other project configuration teams. However, if a change is made to the user interface design in the central configuration, this must be subsequently applied at all project configuration levels. This can be carried out using multi-project administration: A project contains the elements to be administered centrally; another project has the local configuration.

Procedure:

1. A template project is initially sent to all configuration teams
2. This is input into the active workspace
3. Locally, a second, empty project is created that contains the specific amendments
4. There are now two projects in the workspace: the template project and the personalized project
5. The multi-hierarchical arrangement of the projects is carried out in the next step
6. The template project becomes the integration project, the individualized project becomes the sub project

> 💡 **Information**
>
> This method is not suitable for projects that are to run on Windows CE, because only one project can be started on CE.

# 6 Integrated network

Distributed engineering in the network and 100% redundancy are no longer a big effort for you using zenon You can create networks by checking the relevant checkbox in the project's properties. And it is precisely this easy to make efficiently networked, redundant systems from individual standalone projects.

If you have a TCP/IP Windows network, zenon automatically offers network functions with a mouse click, either as client / server model or as a multihierarchical system with substations, workspace centers and centers. It

▸ also create projects with other users in the network at the same time

▶ ensure equipment with the 100% zenon circular redundancy perfectly

▶ Create distributed systems without problems

▶ Access stations remotely

▶ monitor and control equipment via zenon Web Server

▶ see process data in all stations in real time

▶ Make actions of a workspace, such as the acknowledgment of alarms, visible on all others

▶ Have actions logged and archived centrally

▶ Use process data immediately for ERP systems such as SAP

zenon will automatically take care of the required time synchronization on all participating computers.

# 7  Tips and tricks

In zenon, you have many keyboard shortcuts and quick methods of project configuration available. We are presenting a selection of these to you here briefly.

Tips and tricks for

## 7.1  zenon Editor:

Tips for the zenon Editor:

**GENERAL**

▶ Save Editor views: With Editor profiles, you can save your individually-compiled window divisions and assign modules.

▶ Open the folder for the Editor files directly: Highlight the project and press the key combination **Ctrl+Alt+E**.

▶ Open the folder for the Runtime files directly: Highlight the project and press the **Ctrl+Alt+R** key combination.

## SCREEN/SYMBOL EDITOR

▶ Link elements:
You can link functions, variables, fonts etc. to **dynamic elements** by means of Drag&Drop.

▶ Add a copy at the same location:
With the **Ctrl+Shift+V** keyboard shortcut, you add copied elements at the exact same position as the source element.

Drag a circle:
To drag a circle, drag the **Ellipse/circle** element with the mouse whilst holding down the **Shift key**.

▶ Change line symmetrically:
If you press and hold the **Alt key** while pulling the outer corner points, the change is carried out symmetrically.

▶ Scaling:
You can scale several screen elements at the same time: To do this:

  ▶ Mark all elements.

  ▶ Convert this into an element group.

  ▶ Scale the element group as desired.

  ▶ The individual elements are scaled in the same scale.

  ▶ Break up the element group into individual elements again.

▶ Templates for screens:
You can convert any desired screen into a screen template.

▶ Manipulate the X/Y coordinates of an element:

You can enter the X/Y coordinates of an element or corner point of a polygon directly into a dialog or move it with the arrow keys.

  ▶ Enter directly: Double clicking on the sizing handle of an element or corner point of a polygon opens a dialog. You can enter the X/Y coordinates of this point directly.

  ▶ Move with the cursor: Place the mouse pointer over the sizing handle of an element or corner point of a polygon. You can now move the X/Y coordinates of this point directly with the help of the arrow keys.

▶ Jump to linked element:
Right clicking on a property in the properties window opens a context menu that allows you to jump directly to the linked element.

## LANGUAGE SWITCH

▶ Unit labeling:
Set the exact unit name as tool tip. In the usual environment, users normally think in the respective national unit. For trouble-shooting, optimizing internationalization and projects with

a focus on internationalization, the use of this tool tip can make their work significantly easier. Make this tool tip translatable; use international denominations for the country too. You can of course translate the measurement unit to the respective language.

▸ Value units as a character instead of as a word:
Use the unit display for the display of values. There will be no need for a translation then.
**Example temperature display:** The term degree(s) must be translated. The symbol ° however not only is the same symbol in all languages but can also equally be used for Celcius, Fahrenheit and Kelvin.
Set the corresponding symbol in the properties of the basic unit.

▸ Conversion of values:
Add the corresponding value conversion to every language change function. Otherwise misunderstandings might easily occur.

## SYMBOL EDITOR

▸ Unlock property:
You can unlock a property of an element by means of Drag&Drop, moving the property into the lower window of the symbol editor. In doing so, you must always click on the properties using their name, not the value.

▸ Individual editing mode for symbols:
With the **Alt+click** shortcut on an element, the element below it is activated.

## VARIABLES AND DRIVERS

▸ Substitution:
To be able to apply the possibility of substitution optimally:

  ▸ Use structure data types.

  ▸ Ensure that your variables are given a short and concise name.

  ▸ Take the possibility of substitution into account at the naming stage. This naming should be unique and easily-substitutable.

  ▸ Configure a screen as a template first, which can be reused as often as you like once it has been completed by means of a function.

▸ Automatically create variables by importing:
Many drivers support the automatic creation of variables with correct addressing by import from the PLC or from a file. To do this, right-click on the driver with the mouse and select **Import variables from the driver**.

## 7.2   zenon Runtime

Tips for zenon Runtime:

## GENERAL

▶ Display screen name:
Right click on an empty screen area and hold down the mouse button. The screen name is displayed.

▶ Display linkings:
Right click on a **dynamic element** and hold down the mouse button. Linked functions or variables are shown.

▶ Communication error:
Display by means of symbols on the numeric element:

  ▶ Red corner at the top left: No communication with the PLC.

  ▶ Blue corner at the top left: No communication with the server.

▶ Write set value:
Except value, also change command.

Only the value of a variable can be changed in the Runtime. If the standard dialog is used, the function that is to be executed can also be changed by means of the **Command** drop-down list, such as to *Switch off spontaneous value* for example. In order for the standard dialog to be displayed, the **Screen Keyboard** property must be deactivated in the **Dynamic elements** property.

▶ Write set value directly without executing dialog:
To write the set value directly, activate, in the properties of the **dynamic element**, in the **Write set value** group, the **without dialog** property.

This setting can also be set with the **Write/modify set value** function by activating the **Direct to hardware** option.

## HISTORIAN

▶ Show name of the ARX file in the screen:
The name of the ARX file can be shown in the list of the archives in the archive revision screen in a column. To do this, set, in **zenon6.ini**, in the **[ARCHEDIT]** section, the **KURZBEZEICHNUNG=**1 entry. The short description is thus displayed in the list. This corresponds to the ARX file names and is also part of all aggregation archives.

## KEYBOARD OPERATION

▶ Keyboard operation:
Runtime can also be operated in full with the keyboard. For details, see the keyboard operation chapter in the Runtime manual.

▶ Block keys:
Windows keyboard shortcuts can be blocked. For details, see the Block keyboard shortcuts chapter in the Runtime manual.

## 7.3   Keyboard shortcuts

In zenon, you can carry out many actions with keyboard shortcuts.

### GENERAL

| Command | Key combination |
|---|---|
| Open help | **F1** |
| Start/stop full screen mode | **Shift**+**F9** |
| Remote: Close full-screen mode | **Ctrl**+**Alt**+**Shift**+**F** |
| Start VSTA Editor | **Alt**+**F10** |
| Start VBA Editor | **Alt**+**F11** |
| Wizards: Open selection | **Alt**+**F12** |
| Open file explorer for current project with focus on SQL folder.<br><br>Corresponds to:<br>*%ProgramData%\COPA-DATA\[SQL-Ordner]\[UID]\FILES* | **Ctrl+Alt+E** |
| Open file explorer with focus on project files from the current project.<br><br>For example:<br>*C:\Users\Public\Documents\zenon_Projects\[Project]\INI\[Rechner]\INI* | **Ctrl+Alt+D** |
| Open file explorer with focus on Runtime files from the current project.<br><br>For example:<br>*C:\Users\Public\Documents\zenon_Projects\[Project]\INI* | **Ctrl+Alt+R** |
| Start Runtime; create changed Runtime files beforehand. | **F5** |
| Create changed Runtime files. | **F7** |

## EDITOR PROFILES

| Command | Key combination |
|---|---|
| Load Editor profile 1 | **Shift**+**F1** |
| Load Editor profile 2 | **Shift**+**F2** |
| Load Editor profile 3 | **Shift**+**F3** |
| Load Editor profile 4 | **Shift**+**F4** |
| Load Editor profile 5 | **Shift**+**F5** |
| Load Editor profile 6 | **Shift**+**F6** |
| Load Editor profile 7 | **Shift**+**F7** |
| Load Editor profile 8 | **Shift**+**F8** |
|  |  |
| save current Editor view as: |  |
| Editor profile 1 | **Ctrl**+**Shift**+**F1** |
| Editor profile 2 | **Ctrl**+**Shift**+**F2** |
| Editor profile 3 | **Ctrl**+**Shift**+**F3** |
| Editor profile 4 | **Ctrl**+**Shift**+**F4** |
| Editor profile 5 | **Ctrl**+**Shift**+**F5** |
| Editor profile 6 | **Ctrl**+**Shift**+**F6** |
| Editor profile 7 | **Ctrl**+**Shift**+**F7** |
| Editor profile 8 | **Ctrl**+**Shift**+**F8** |

## GRAPHIC EDITOR:

**Note for shortcuts:** The plus sign (+) means that keys a pressed together.
For example:

**Ctrl+A** means: Hold down the **Control key** and then press the **A** key.

**Ctrl++** means: Hold down the **Control key** and then press the plus key.

## GENERAL

| Command | Key combination |
| --- | --- |
| Main window: Scroll content with 'moving hand' | Press and hold **Space** |
| Close current screen | **Ctrl**+**F4** |
| Open properties | **Alt**+**Enter key** |

## SELECT

| Command | Key combination |
| --- | --- |
| Select several objects | Hold down the **Shift** or **Control** key |
| Deselect selected object during multi-select | **Ctrl**+**mouse click** |
| Selection: Change sort order. Defines the element on which all others realign | Hold down the **Shift key** when selecting |
| Select hidden objects | 1. **Press the Alt key**<br>2. **Click object and move it** |
| Select all elements of a screen. | **Ctrl**+**A** |
| Select next element according to the order of their creation | **Tab** |
| Select previous element according to the order of their creation | **Shift key**+**Tab** |

## POSITIONING

| Command | Key combination |
| --- | --- |
| Move selected object. | **Arrow keys** |
| Move by 10 pixels each time you press an arrow key | **Shift key**+**Arrow key** |
| Move only horizontally or only vertically | Hold down the **Shift** key when moving |
| Centers the selected object in the working section | **H** |

## ACTIONS

| Command | Key combination |
|---|---|
| Saves changes | **Ctrl**+**S** |
| Pastes element from the clipboard | **Ctrl**+**V** <br><br> **Shift**+**Ins** |
| Inserts element from the clipboard at its original position; original and copy lie congruently on top of each user | **Ctrl**+**Shift**+**V** |
| Copies selected element. | **Ctrl**+**C** <br><br> **Ctrl**+**Ins** |
| Copy instead of move | Hold down the **Control key** when moving |
| Duplicates the selected element. <br><br> You can find more detailed information in the Duplicating elements section. | **Ctrl**+**D** |
| Deletes selected element | **Del** |
| Cuts out the selected element. | **Shift key**+**Del** <br><br> **Ctrl**+**X** |
| Opens the dialog to replace links for the selected element. | Ctrl+R |
| Undoes changes | **Ctrl+Z** <br><br> **Alt**+**Backspace** |
| Add or delete node in the selected element. Add: Mouse cursor turns to plus symbol (+). Delete: Mouse cursor turns to minus symbol (-). Works for polylines, polygons and pipe elements. | **Ctrl**+**Shift key** |
| Cancel drawing of polylines and polygons | **S** |
| Cancel drawing of polylines and polygons and delete the section which was drawn last | **Esc** |

| Command | Key combination |
|---|---|
| Move selected elements one level up | **+** |
| Move selected elements one level down | **-** |
| Move selected elements to the foreground | **Ctrl++** |
| Move selected elements to the background | **Ctrl+-** |

## SCALING

| Command | Key combination |
|---|---|
| Change size | Move mouse cursor to the handle so that the mouse cursor changes to an arrow. After that you can position, with pixel precision, using the **arrow keys** or in steps of 10 pixels with the **Shift** key held down.<br><br>**Note:** If an angel dissimilar to *0* via property **Rotation angle [°]** was defined for an element, scaling via arrow keys is not possible. |
| Scaling object around the center | Hold down the **Alt key** when scaling. |
| Proportional scaling | Hold down the **Shift** key when scaling. |

## ZOOM

| Reduce view | **Ctrl** + **Shift key** + **-** |
|---|---|
| Enlarge view | **Ctrl** + **Shift key** + **+** |
| Set view to 100% | **Ctrl** + **Alt** + **0** |
| Amend the view to the space available in the Editor and display it proportionally | **Ctrl** + **0** |

## ZOOM WITH THE MOUSE

| Reduce view | Ctrl + scroll wheel of the mouse downwards |
|---|---|
| Enlarge view | **Ctrl** + scroll wheel upwards |

## DETAIL VIEW

| Command | Key combination |
|---|---|
| Create a new element for the respective module | **Ins** |
| Edit the selected column | **F2** |
| Copy a selected list element | **Ctrl**+**C** |
| Insert a list element which was copied beforehand | **Ctrl**+**V** |
| Delete a selected list element | **Del** |
| Scroll up several elements in the list | **Pg up** |
| Scroll down several elements in the list | **Pg down** |
| Navigate in the list | **Arrow key** |

## RUNTIME

| Command | Key combination |
|---|---|
| Browse through open Window windows. | **Alt**+**Tab** |
| Stopping the Runtime. | **Alt**+**F4** |

## ELEMENTS CONTEXT MENU

| Create element group | Creates an element group. The keyboard shortcut Ctrl + G can also be used for this. |
|---|---|
| **Resolve** | Resolves an element group into its screen elements. The keyboard shortcut **Ctrl + Shift key + G** can also be used for this. |
| Full-screen mode | Close full-screen mode: **Shift key** + **F9** |

## 7.4 Test of projects

Tips for testing projects:

▶ Name of the variable in the screen:
  To display a variable name in the screen, configure a combined element without status and
  with the default text: **%n**.