zenon manual

Efficient engineering

v.7.10

Contents

1. Welcome to COPA-DATA help

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 (mailto: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 (mailto: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 (mailto: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. This chapter is recommended even when you already gathered some experience with other process control systems 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 monitoring is done in the Runtime. Editor and Runtime are compatible across versions from version 6.20 on. A project that was created with version 6.50 for example also works with Runtime version 6.22 and vice versa.

zenon mainly differs from other systems by:

- Easy operation (on page 5): 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 7): You work continuously with objects that can be parameterized swiftly instead of being programmed rather time-consumingly . Once defined, objects can be used across projects. You profit from inheritance and can change properties in the blink of an eye.
- Excellent reusability (on page 19) of objects and elements that have been configured: Many elements, from individual variables to complete projects can be very easily and effectively reused in zenon.
- Integrated network (on page 41): Distributed engineering in the network and 100% redundancy are no longer a big effort. You can create networks by activating the relevant checkbox. This way, you can also network individual stand-alone projects to become 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 (mailto:sales@copadata.com).

2.1 Easy 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. You can choose from them at the beginning of the properties lists. That way you can swiftly access the properties you need frequently.

GENERAL KEYBOARD OPERATION IN THE EDITOR

The Editor can be easily operated via symbols and menus with the mouse; you can also universally use the keyboard. The shortcut key combinations from Windows also work in zenon. For example:

- `CTRL+C` and `CTRL+V` for**`copy`** and **`paste`**
- `INSERT` creates new objects
- `DEL`deletes selected objects
- **`Mouse cursor`**keys navigate in the views
- `Esc` terminates the current action
- `Backspace` jumps in the directory tree to the next higher level
- `CTRL+Z` undoes the last actions with the exception of database operations

Read more about this in chapter Keyboard shortcuts.

DRAG&DROP

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

- Change the order of menu entries
- Link fonts with dynamic elements like universal slider, clock, combined element etc.
- Change the order of structure elements for structure data types
- Change the order within a script
- Copy or move functions using different scripts
- Copy scripts by moving them onto the master node

INTELLIGENT 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. We recommend the wizards for the universal slider, the combined element and the archiving especially for beginners.

TOOLTIPS FOR ELEMENTS AND PROPERTIES

If you move the mouse pointer above a display element (for example, a text button) or a linked property (for example the function linked to a text button), a tooltip pops up. It shows information about the linked elements, like the linked function and its parameters.

EXTENDED DRAWING POSSIBILITIES

The graphical editor offers a broad range of functions that make drawing the equipment easy and also offers you a lot of possibilities. From stepless zoom to scaling with a click, locking element corner points as well as automatic distribution of elements following rules. You can find details in chapter Images.

GO TO LINKED ELEMENT

For all graphical elements, functions, variables and other elements it´s possible to jump directly to the linked element. For example, you can click on a display element and are directly shown the linked variable, if you click on it you come directly to the used data type and by clicking on it you can even see the used unit. If you need to, you can conveniently take back all these steps up to the starting point with some mouse clicks.

2.2 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 8):

   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.

2.2.1 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. <u>Predefined modules save time</u>
   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. <u>The product from the manufacturer is already mature</u>
   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. <u>Easy maintenance and adaptation of the project</u>
   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

- A clear structure: 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. <u>Easy adaptation for different machines</u>
   zenon can be adapted easily and swiftly to new or changed machines. Project adaptations can be done with just a few mouse clicks.

ADAPT ZENON

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.

- <u>VBA/VSTA:</u>
  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.
- <u>zenon Logic:</u>
  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.
- <u>PCE:</u>
  The integrated Process Control Engine allows you to use VB-Script and Java Script for automation tasks.
- <u>ActiveX Controls:</u>
  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.
- <u>WPF-Elements</u>
  The direct integration of Microsoft´s 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.
- <u>COM-interface:</u>
  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.

2.2.2 Global and central instead of local

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

- directly on the object
- centrally
- 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.

Info

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.

2.2.3 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:

- Drivers (on page 11)
- Data type (on page 13)

Drivers

Drivers are not directly integrated in zenon, they are implemented via a driver object type. This driver object type defines the area of the PLC that will be addressed.

DRIVER OBJECT TYPE

There are many different driver object types, e.g. standard PLC markers, data blocks, inputs, outputs, counters, etc. – but also special types like alarm or driver variables.

The driver object type determines:

- the area of the PLC
- the driver´s granularity in that area
- it defines the data types that can be created in the memory area

Hint: Not any data type can be created in any area. Therefore, always proceed as directed when creating a variable:

- first the driver,
- then driver object type and
- lastly the data type

ASIDE: GRANULARITY

Granularity is particularly important for numerical adressing PLCs, like for example Siemens S7. Not every PLC has the same resolution, and not every area in a PLC has the same resolution.

For example, the data block area of a Siemens S5 PLC is word-oriented, whereas its marker area is byte-oriented. This means that the smallest unit that can be addressed is one byte or word. Now if you want to write a bit into such an area, you have to read at least a whole byte/word, mask the desired bit, change it, and then write the whole byte/word back to the PLC.

The same information is required when addressing the areas, called offset. You start with zero and go on counting in byte/word steps. If you use automatic adressing (on page 18)in zenon granularity is automatically considered. So, granularity does not depend on the driver but on the driver object type.

DRIVER VARIABLES

Driver variables are offered by every driver and offer many advantages for projecting. They do not communicate with the PLC but read out an internal memory area of the driver which contains mainly statistical information. But you can also control special functions with variables like telephone numbers for modem connections or dial/hang-up commands.

You can find more information in the chapter Driver variables.

A dBase file with the most important driver variables is included on the installation CD of zenon. You can use it for the simple and efficient import of these variables into every driver instead of creating them manually.

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

Simple data types

Simple data types are always IEC data types, that means data types defined by the IEC in the 61131-3 standard, like BOOL, INT, USINT, UDINT, STRING, WSTRING, etc. They help to define the size of an area - for example using
BOOL: 1 Bit
INT: 16 Bit signed
UINT: 16 Bit unsigned

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

- Identification
- Unit
- Value range
- Limits

Why this "double accounting"?

The reason is easy to explain:
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 limits from the data type it is based on.

OBJECT-ORIENTATION

As opposed to the linking of dynamic elements and fonts, we are dealing with an object-oriented approach here: The data type object inherits its properties to the variable. The difference?

The linked/derived properties of the variable can be separated/overwritten. This works for every single property, but also for all properties at once. This means you can take over the unit from the data type centrally, yet overwrite the identification or the address directly at the variable. This is how it works:

- 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 further functions via the context menu:

- Link property with datatype:

  If you have separated the link, you can reestablish it.

- Separate property:

  You can separate a single property without having to change the value. If you change the value of the data type later on, the variable will not be modified by this action. This allows you to define exactly where the inheritance concept should be applied.

A useful option: you can also link or separate all properties from the data type with one mouse click. This allows you to switch to and from the original state (everything inherited) really quickly. How do I, as the user, profit from all this?

Increased flexibility: If you have many variables with the same limit, e.g. an alarm for value 1, you can simply set it at the data type. The actual limit and other optional parameters can then be set for every variable separately. You no longer have to create and update the limit for every single variable, which will drastically increase your engineering performance. The default data type BOOL has a predefined limit at 0 and 1!

If you don´t use a limit for each BOOL variable, you don´t have to take care and to manually deactivate the limit for each variable. Instead, you can conveniently use a feature of zenon and create your own BOOL data type. Proceed as directed:

- Leave the standard data type as it is

- Click on button `New data type .`
- You will then be offered the list with the already existing data types.
- Choose `Standard BOOL`:
- Assign a name, e.g. `MyBOOL`
- All the properties of the new data type are automatically taken over by the old one
- Delete the limits in`MyBool`

Use this new data type `MyBool` to create all BOOL variables that are not supposed to have limits – this removes all unwanted limits. The other limits, which you do want to stay, will remain untouched.

Of course you can also create more of your own BOOL data types, as many as you like. All variables that require the same settings for most of their properties get their own data type. If one of the properties changes, you can perform the changes centrally and all derived variables will automatically be updated – except the ones whose links you separated.

Structure data types - structure elements - structure array

STRUCTURE DATA TYPES

Structure data type designates a group of data types whose sequence and arrangement is exactly defined. For example, a structure "Motor" can consist of the elements "Actual speed (RPM)" and "Power input".

A structure can also be nested, different hierarchy levels are possible.

With structure data types, you can build up your variable pool exactly as it exists in the PLC or in reality.

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.

The properties will be assigned only after adding structure elements to the structure data type. We differentiate between the following types:

- Linked structure data type:

  Reference to an existing data type. All properties are taken over except for the name.

Advantage: If the structure element provides many properties of an already existing data type, they can be reused. Of course, you can still change each of the variable properties or separate them from the data type.

- own embedded structure data type:

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

Info

Advantage object orientation: If the structure changes somehow, it suffices to change the structure data type to adapt all variables in this structure automatically.

Changes are possible in all specifications:

- simple properties like the unit of the data type
- complex properties like adding or deleting a limit
- Changing the whole structure, e.g. by changing the order of elements or adding / deleting unique structure elements in the structure data type

STRUCTURE ARRAY

If not only one machine is concerned 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. With a single mouse click, 100 variables become 100 structure variables.

Sticking to our example, we have to create 12 variables for each of our pumps, which results in a total of 1200 variables. The method "object-oriented parametrization" makes that possible with just a few mouse clicks. As additional advantage, all variables are predefined. Every single variable already comes with all the properties it needs, e.g. unit, value calculation, alarms, CEL entries etc.

Info

Structure data types are also suitable for the reuse (on page 31) of variables.

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.

Tips for adressing and import / export

Practical examples:

- Addressing
- Import / Export

ADDRESSING:

With numeric controls you can choose if adressing shall be automatic or semi-automatic. All properties described here can be reached, too, via VBA/VSTA and can thus be used for automatic projecting.

AUTOMATIC ADDRESSING:

The offset and, if required, the byte and bit addresses are calculated automatically for structure elements, based on the position of the structure elements - for arrays across the whole 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:

Here, you assign unique start addresses already with the data type. The other addresses will then be determined based on these addresses. Of course, you can also change them later. Details can be found in Automatic addressing.

For symbolically adressed PLCs, you have to use the same name in zenon and in the PLC.

XML IMPORT / EXPORT:

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.

2.3 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:

Topic	Reuse of
Replacing variables and functions (on page 19)	Elements
Symbols (on page 26)	Screens
Structure data types (on page 31)	Variables
Reaction matrices (on page 34)	Variables
Global project (on page 34)	Central elements of a project
XML (on page 36)	Project parts by means of import and export
Wizards (on page 37)	Screens and elements by means of import or individualization
Reusing projects (on page 38)	Projects

Info

To be able to reuse elements efficiently, ensure when you create variables, functions and screens that they are clearly named (on page 20).

2.3.1 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 23)
- Replace indices (on page 24)
- You can also find more about replacement of variables and functions in the Screens manual in the Replacing linking of variables and functions section.

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

Germany

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

International

- 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.

POWER PLANT CLASSIFICATION SYSTEM 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`

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 23): Only one screen is used for different PLCs; the links are adapted when switching.
- Replacing indices (on page 24): Replacement of variables in a process screen using the value of index variables.

Replacing linking in a screen

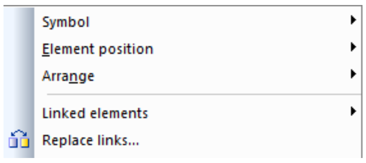
If variables are attached in a screen, these can be easily replaced by means of a replacement dialog. Requirement:

- Clear naming of variables (on page 20)

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

- Right-click on the screen element

- Click on **`Replace links`**



The dialog for replacement is then opened.

EXAMPLE

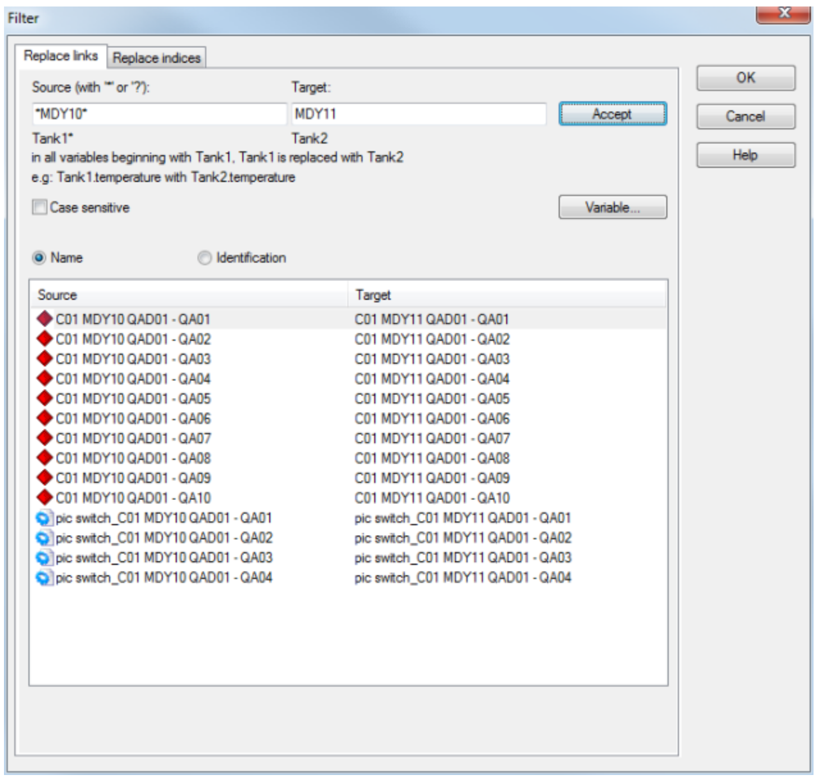
In our example, the project has a number of variables from different equipment parts. The following are on the process screen:

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

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 and/or functions by the corresponding ones from the new PLC. To do this:

1. The dynamic elements that the variables and functions are linked to are highlighted

2. The replacement dialog is opened



Filter

Replace links | Replace indices

Source (with "*" or "?"): *MDY10*

Target: MDY11

Accept

Tank1*	Tank2

in all variables beginning with Tank1, Tank1 is replaced with Tank2

e.g: Tank1.temperature with Tank2.temperature

Case sensitive

Variable...

(●) Name ( ) Identification

Source	Target
C01 MDY10 QAD01 - QA01	C01 MDY11 QAD01 - QA01
C01 MDY10 QAD01 - QA02	C01 MDY11 QAD01 - QA02
C01 MDY10 QAD01 - QA03	C01 MDY11 QAD01 - QA03
C01 MDY10 QAD01 - QA04	C01 MDY11 QAD01 - QA04
C01 MDY10 QAD01 - QA05	C01 MDY11 QAD01 - QA05
C01 MDY10 QAD01 - QA06	C01 MDY11 QAD01 - QA06
C01 MDY10 QAD01 - QA07	C01 MDY11 QAD01 - QA07
C01 MDY10 QAD01 - QA08	C01 MDY11 QAD01 - QA08
C01 MDY10 QAD01 - QA09	C01 MDY11 QAD01 - QA09
C01 MDY10 QAD01 - QA10	C01 MDY11 QAD01 - QA10
pic switch_C01 MDY10 QAD01 - QA01	pic switch_C01 MDY11 QAD01 - QA01
pic switch_C01 MDY10 QAD01 - QA02	pic switch_C01 MDY11 QAD01 - QA02
pic switch_C01 MDY10 QAD01 - QA03	pic switch_C01 MDY11 QAD01 - QA03
pic switch_C01 MDY10 QAD01 - QA04	pic switch_C01 MDY11 QAD01 - QA04

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.

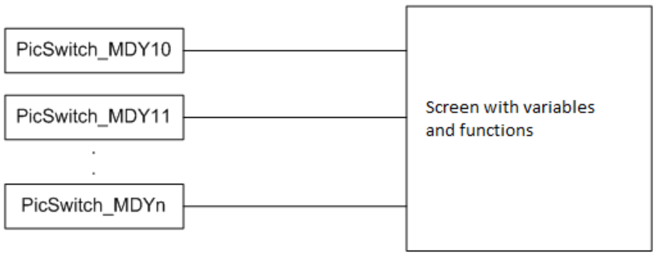
Replacing linking for screen switching

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

1. Several screen switching functions are configured on this screen

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

3. Each function contains its own replacement process



A variant of this that works with only one screen switching: Replacing indexes (on page 24) for arrays.

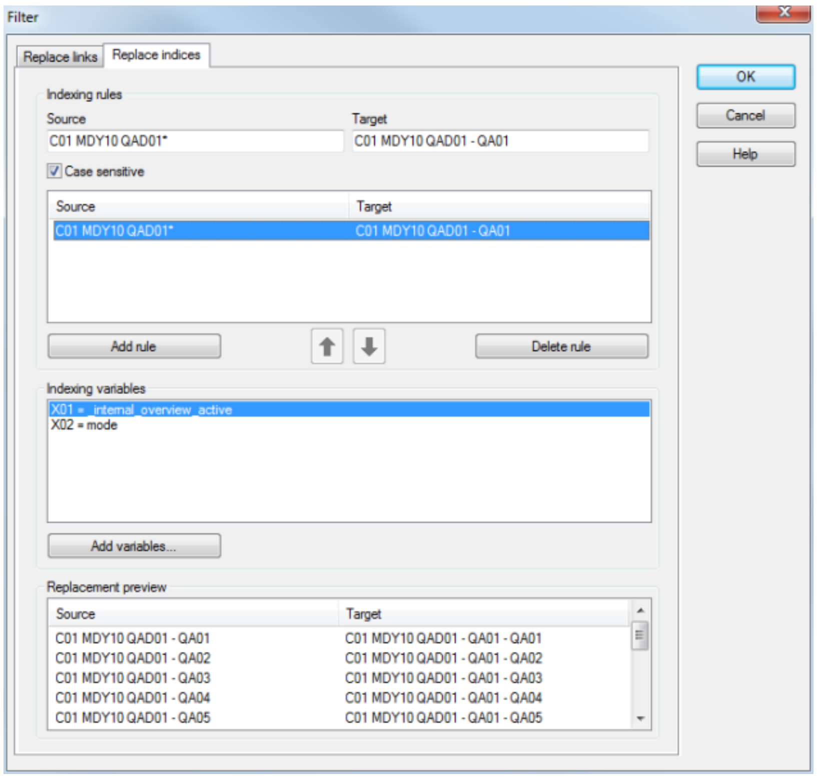
Replace indices

In Arrays, replacement for variables can be carried out via index variables.

As with Replacing linking for screen switching (on page 23), the screen is only used once. The linked variables are replaced during the switching. Screen switching is only configured once (different to Replacing linking in the screen (on page 21)) and can be reused multiple times via an index variable.

EXAMPLE

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


Filter

Replace links | Replace indices

Indexing rules

Source: C01 MDY10 QAD01*

Target: C01 MDY10 QAD01 - QA01

[✓] Case sensitive

Source	Target
C01 MDY10 QAD01*	C01 MDY10 QAD01 - QA01

Add rule | Delete rule

Indexing variables

X01 = _internal_overview_active
X02 = mode

Add variables...

Replacement preview

Source	Target
C01 MDY10 QAD01 - QA01	C01 MDY10 QAD01 - QA01 - QA01
C01 MDY10 QAD01 - QA02	C01 MDY10 QAD01 - QA01 - QA02
C01 MDY10 QAD01 - QA03	C01 MDY10 QAD01 - QA01 - QA03
C01 MDY10 QAD01 - QA04	C01 MDY10 QAD01 - QA01 - QA04
C01 MDY10 QAD01 - QA05	C01 MDY10 QAD01 - QA01 - QA05

OK | Cancel | Help

The indexing rule:

- Source: C01 MDY10 QA001*
- Target: C01 MDY{X01} QA001

the effect of this:

- With screen switching, the function is informed that it must use the variable value of **`X01 wind turbine control index`** in the variable names.

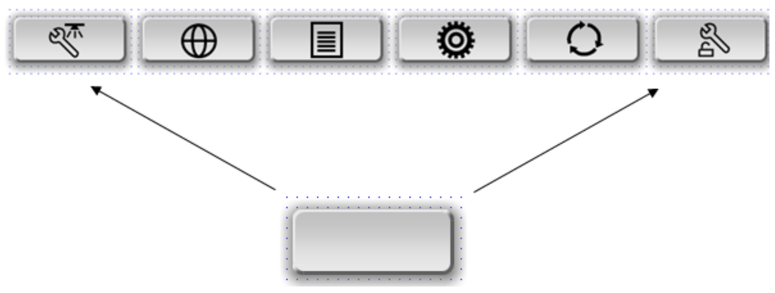
  Example: If this variable has a value of 12, then the process screen is displayed with all wind turbine control variables **`MDY12`** when screen switching is executed

2.3.2 Symbols

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

EXAMPLE OF A BUTTON BAR

A symbol (empty rectangle) is linked as the basis in all buttons.



Free-access properties (on page 26) make it possible to issue each button with different graphics. If the form or color of the buttons is to be changed, the "empty button" symbol must be changed. All buttons automatically accept the new form and/or color.

Free access properties

Linked symbols pass their properties on 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.

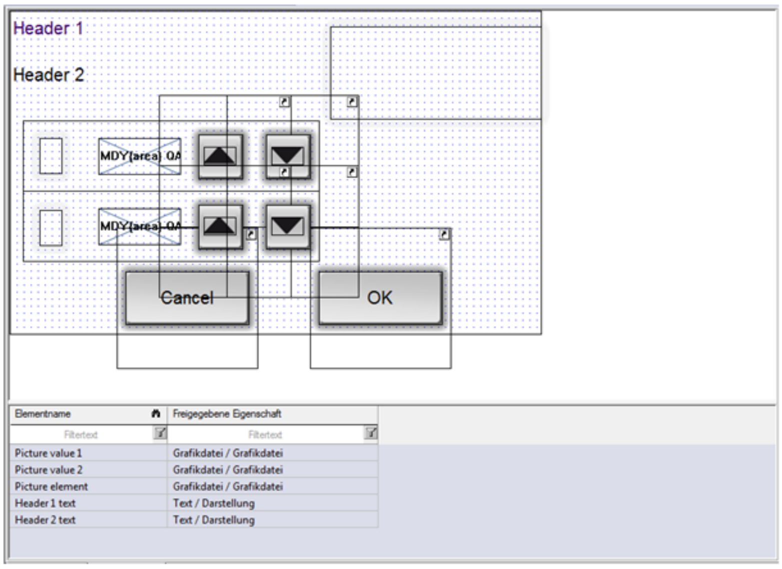
Copy and paste:

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.

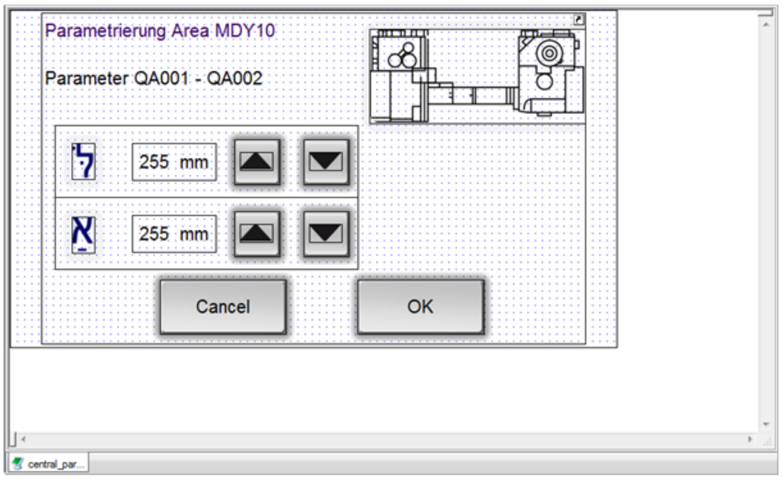
Approach using symbols with released properties:

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 to the corresponding screens.

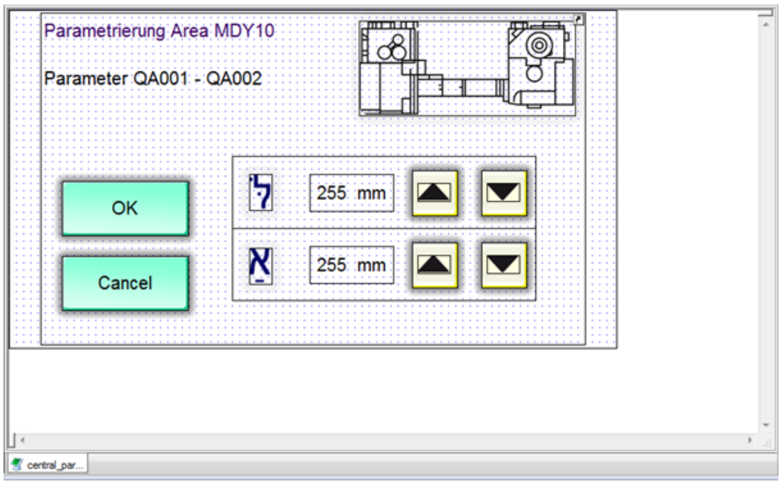


Elementname	Freigegebene Eigenschaft
Filtertext	Filtertext
Picture value 1	Grafikdatei / Grafikdatei
Picture value 2	Grafikdatei / Grafikdatei
Picture element	Grafikdatei / Grafikdatei
Header 1 text	Text / Darstellung
Header 2 text	Text / Darstellung

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. Global symbols can be adapted individually using released properties and corresponding replacement (on page 29).



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.

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 20) 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

- embedded in a process screen.
- linked in a process screen

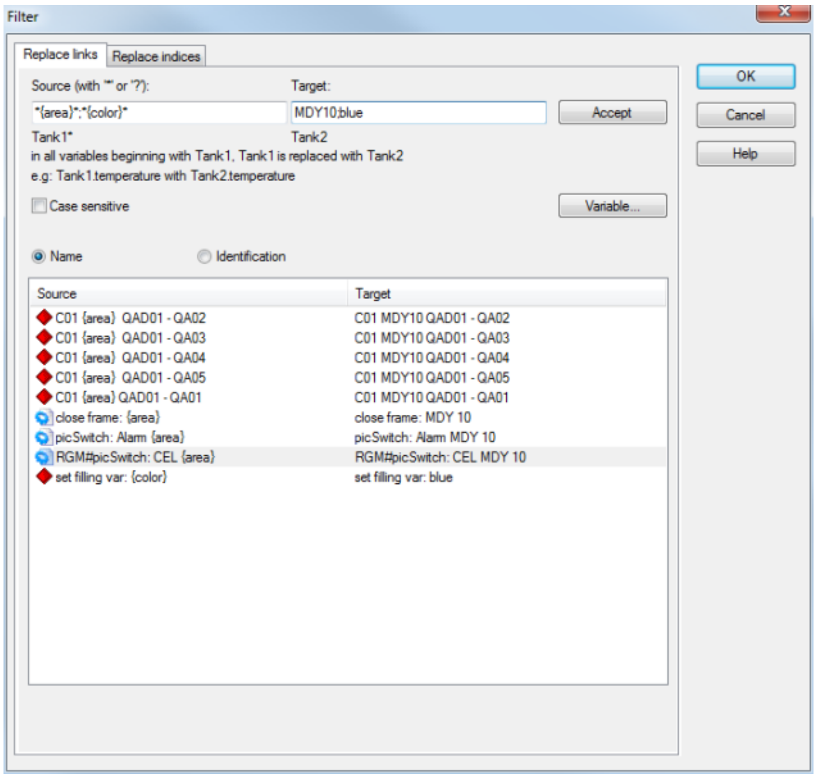
Depending on the approach, different dialogs are switched for the replacement of variables/functions.

EMBEDDED SYMBOLS

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.

LINKED 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 (;).



Source (with "*" or '?'): *{area}*;*{color}*
Target: MDY10;blue

Tank1* Tank2
in all variables beginning with Tank1, Tank1 is replaced with Tank2
e.g: Tank1.temperature with Tank2.temperature

☐ Case sensitive

[✓ Name] Identification

Source	Target
C01 {area} QAD01 - QA02	C01 MDY10 QAD01 - QA02
C01 {area} QAD01 - QA03	C01 MDY10 QAD01 - QA03
C01 {area} QAD01 - QA04	C01 MDY10 QAD01 - QA04
C01 {area} QAD01 - QA05	C01 MDY10 QAD01 - QA05
C01 {area} QAD01 - QA01	C01 MDY10 QAD01 - QA01
close frame: {area}	close frame: MDY 10
picSwitch: Alarm {area}	picSwitch: Alarm MDY 10
RGM#picSwitch: CEL {area}	RGM#picSwitch: CEL MDY 10
set filling var: {color}	set filling var: blue

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 Linked symbols section.

Global system library

Symbols are either administered in the project symbol library or in the global symbol library. The most important difference:

Symbol library	Property
`Global system library`	Symbols are available in all projects. The **Global symbol library** node is in the project manager below the currently-loaded projects Global symbols are saved in the zenon program folder and only updated when the Editor starts. These symbols are not saved during project backup.
`Project Symbol Library`	Symbols are only available in the current project. 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.

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 26) and replacement (on page 29).

It is not always necessary to distribute the whole collection of symbols. An XML export/import (on page 36) 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.

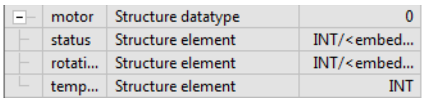
2.3.3 Structure data types

Variables can be reused using structure data types. Complex structures can therefore 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 elements differ in the type of inheritance:

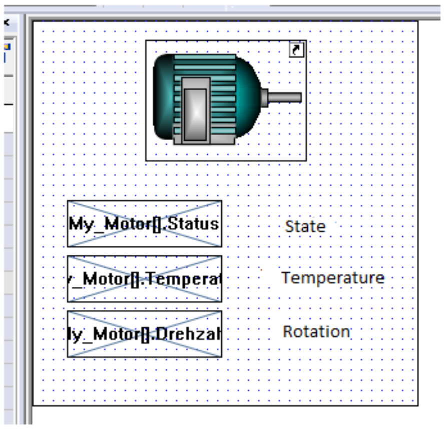
- `Status`" and "`Speed`" are derived from the `INT` data type, but their inheritance was broken off by the embedding.

  This means: Subsequent updates to `INT` data type have no effect on these two elements.

- "`Temperature`" also comes from the `INT` data type, the inheritance is intact.

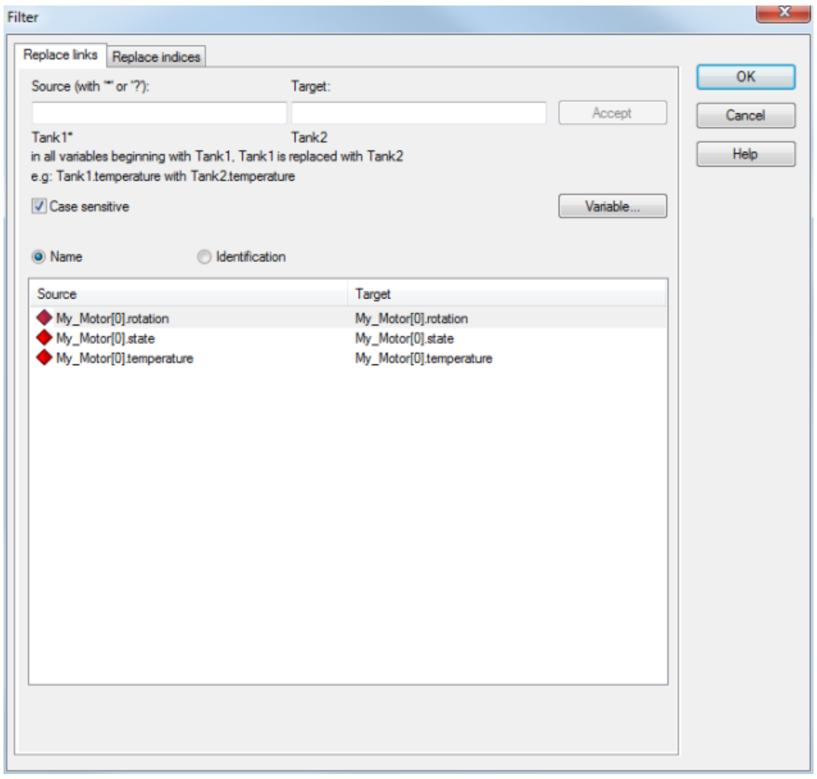
  This means: Subsequent changes to the `INT` data type also have an effect on the settings of this structure element.

If, for example, 20 motors are now configured, variables based on the same self-created "motor" data type can be created. This can, for example, happen by creating an array of this data type. Graphic display is solved in this example by means of a symbol.



When this symbol is linked in a process screen, the fill-in variables of the symbol are replaced by those of the self-created structure data type. Ensure that the naming (on page 20) is well thought out 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. The target variable must still be replaced by the corresponding sequence number at each symbol:



Filter

Replace links | Replace indices

Source (with "*" or '?'): Target:

Tank1* Tank2
in all variables beginning with Tank1, Tank1 is replaced with Tank2
e.g: Tank1.temperature with Tank2.temperature

[✓] Case sensitive

(●) Name ( ) Identification

Source	Target
My_Motor[0].rotation	My_Motor[0].rotation
My_Motor[0].state	My_Motor[0].state
My_Motor[0].temperature	My_Motor[0].temperature

If limit values for the "speed" are to be added at a later time, you benefit from the existing inheritance relationship 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 its "`motor`" data type. This change to a position that has been made affects all 20 created motors.

The person configuring the project is completely free to change this inheritance relationship. A limit value for "**speed**" can be changed for one of the 20 created motor variables.

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

Info

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

2.3.4 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.

2.3.5 Global project

Standards can be defined throughout projects with a global project. With this, it is possible to define certain modules or elements for standard locations, whilst local projects define 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 central project configuration, because a breakdown into two "plants" can be made. In doing so, it must be clearly defined which configuration elements come from the central configuration point and which come from the local configuration point.

- User:

  If a standard is to have certain globally-valid users, these can be defined using the global project.

- Files:

  Graphics can be prescribed by a standard, for example, which have to be used in individually-created projects. These can be made available globally.

- Frames:

  Defining the process screen arrangement and size of the projects to be developed. For example, it could be set out clearly in some guidelines that newly-created screens can only be created using frames from a global project. A uniform global, graphic appearance is therefore created.

- Fonts and color palettes:

  To create uniform displayed text and colors, a pre-defined set of font lists and/or color palettes can be defined for global use.

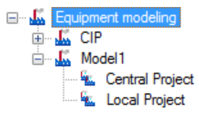
- Language tables:

  If, as a result of a standard, there is a defined set of terms and statements to be used in configuration, it is possible to define these using the defined language tables and to carry out the corresponding translations at the same time.

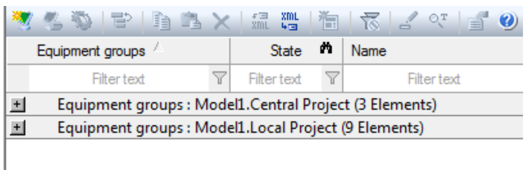
EXAMPLE

A central configuration point defines standards in the areas of color, fonts, templates, etc. using a global project for all people configuring the project. The global project is distributed to all teams as a project backup (on page 38). They import it into their workspace. A guideline must clearly stipulate that local configuration points must not make changes to a global project, because this would be repeatedly overwritten when updates from the central configuration point are made.

EXAMPLE OF AN EQUIPMENT MODEL



A simple equipment model, which only makes a distinction between a central and a local configuration group. In an individualized project, this distinction could now be used for assistance when visualizing in the editor. In detail, this would look as if each configured element in the Editor was issued with a corresponding equipment group.



Equipment groups	State	Name
Filter text	Filter text	Filter text
Equipment groups : Model1.Central Project (3 Elements)		
Equipment groups : Model1.Local Project (9 Elements)		

The existing process screens are displayed after the equipment has been assigned.

Therefore a local configuration point can clearly recognize which screens are updated by a central configuration point by means of an XML import. For people configuring a project locally, it is also clear which screens they are not permitted to change. This is because each change to globally-evaluated screens is overwritten again during the next update.

Hint: To update a global project, it is not always necessary to create and import a complete project backup. Individual changes can also be updated locally by exporting and importing via XML (on page 36).

2.3.6 XML

Import and export via the XML interface offers diverse possibilities for reusing project parts.

Attention

Existing elements with the same name are overwritten during import.

DEPENDENCIES DURING IMPORT

If several project parts are to be imported, there are several XML files with different contents accordingly. To accept all content of several XML files, existing dependencies must be taken into account:

If, for example, a screen is imported and its associated variables do not exist in the project, the links cannot be re-established! To create all variables correctly, it can be necessary to carry out the import of a screen or variables twice.

EXAMPLE FOR XML SCREEN:

The most comprehensive way is to export a screen, because the XML file created contains not only the screen and the contained elements, but also the frame, associated variables and functions – everything that can be seen in the screen and that is linked directly.

In screen "**A**", we use a button to execute a function "**B**", which opens screen "**A**" again.

- Function "**B**" requires screen "**A**", in order for this to be linked to function "**B**".
- Screen "**A**" requires function "**B**", in order for this to be linked to the button.

You must therefore:

1. First import the function,
2. then the screen and then
3. the same function once again, because otherwise the relationship to the screen cannot be created in the function.

HARMONIZING PROJECTS THROUGHOUT PLANTS

XML export/import is also suitable for guaranteeing uniformity of projects throughout a plant. If a company decides that certain project parts should be designed in a uniform manner, then the uniform elements are:

1. Configured centrally
2. Exported into an XML file
3. Distributed as an XML file
4. Accepted by each local team via XML import

Alternatively, the export and import can also be carried out with a specific wizard.

Because existing elements are overwritten are overwritten during XML import, these uniform elements must be named clearly and with a binding system. Local changes to these elements are overwritten again at the next import.

WIZARDS (ON PAGE 37)

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 guidance. A possible approach for this wizard is to create an XML library that contains different variants of screens and functions as individual data.

INDIVIDUALIZATION USING A WIZARD

A wizard can also be used to configure standard functions that are present in XML format. A function is imported via XML and then adapted to individual requirements by changing certain properties that are not the same for each object. For example, there can be a screen switching function in XML format and this can be reused many time by individualizing the properties.

2.3.7 Wizards

The reuse of elements and parts of projects can be simplified and supported with wizards. Examples of where wizards are suitable most of all are:

- Creation of template projects:

A wizard makes it possible to create certain standard parts of a new project that is to be created by means of a few mouse clicks. In contrast to simple loading in of a complete project backup (on page 38) of the template project, the person configuring the project now has more scope for individualization.

For example, a wizard can create different project types, depending on the machine.

- Creation of pre-defined projects:

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

For example: Creation of a "`Motor`" project area.
The wizard creates pre-defined elements for this area. With this, appropriate screens, functions, variables etc. can be configured with a few mouse clicks. In addition to the shorter configuration times, you also obtain a uniform appearance, because the wizards are programmed by a central configuration point and used by the local 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 parameters, 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

2.3.8 Reusing projects

Projects could also be reused in different ways. In doing so, mechanisms from the other chapter of reusability sometimes play a significant role.

Project backup (on page 38)

Save as (on page 39)

Multi-project administration (on page 40)

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 36) is recommended.

In doing so, note that existing elements with the same name are overwritten during import. During configuration, the configured elements that may be overwritten during import must be clearly identified. For example, a corresponding suffix to the element name is suitable , for example, or the procedure using equipment groups, see global project (on page 34) section.

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 38) 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 36).

In doing so, note that existing elements with the same name are overwritten during import. During configuration, the configured elements that may be overwritten during import must be clearly identified. For example, a corresponding suffix to the element name is suitable , for example, or the procedure using equipment groups, see global project (on page 34) section.

Multi-project administration

Projects can also be compiled in a workspace in zenon multi-project administration as part of of an integration project. This way, a project can be changed centrally, whilst local configuration is carried out in another project.

EXAMPLE

Graphics are to be reused in a local project by means of a central configuration point, in order to create a uniform user interface globally. To do this, the central configuration creates a template project, which displays the graphical basis for all other configuration teams. However, if a change to the user interface design is made at the centrally, this must be implemented at all project sites. 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 then uploaded to the active workspace
3. A second empty project is created locally, 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

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

2.4 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. Creating efficiently networked, redundant systems from isolated standalone projects is just as easy.

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

- create projects - also with several users - simultaneously over the network
- ensure equipment with the 100% zenon circular redundancy perfectly
- install distributed systems without any problems
- have remote access to stations
- monitor and control equipment via web server
- see process data in all stations in real time
- can display actions set on one workstation (e.g. acknowledging alarms) on all others, too
- Microsoft Report Designer:
- use process data instantly for ERP systems like SAP

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