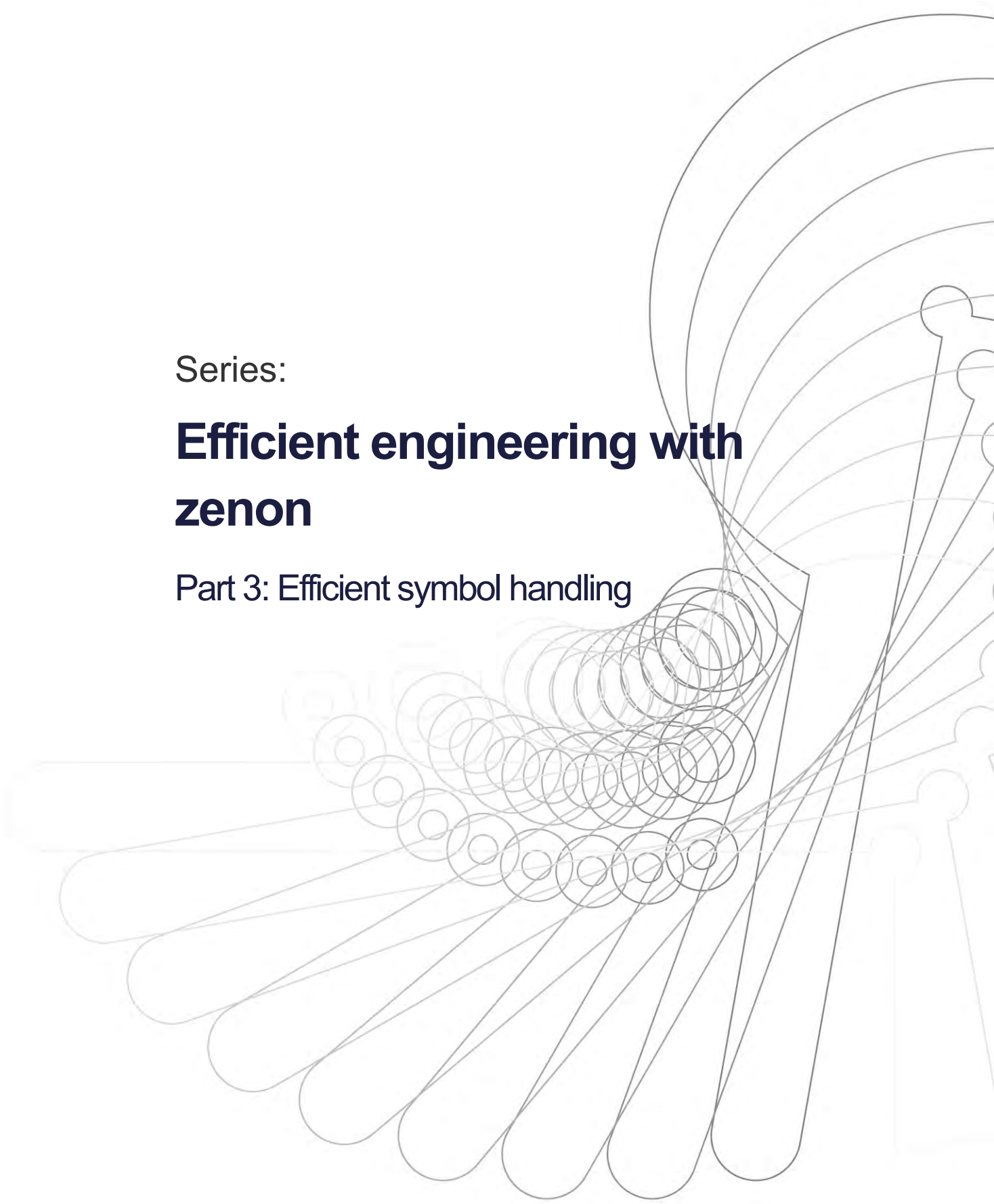
Series:

Efficient engineering with zenon

Part 3: Efficient symbol handling



zenon®

Contents

Efficient symbol handling

In this whitepaper, we will look at the topic of “efficient symbol handling” in more detail. We will show you how, with a few mouse clicks, you can create symbols universally, so that their handling saves a great deal of work. We will look at such key ideas as embedded symbols, linked symbols, symbol in symbol and “linking rule” versus “replace link”

Introduction

I would like to begin by repeating the most important messages from the last two articles in this series. Our starting point, as always, is the zenon philosophy: “engineering instead of programming” (Information Unlimited, edition no. 15). The reason for this is easily explained: projects are created more quickly, are easier to roll out and last, but not least, service and maintenance remain simple, even after many years of operation. The advantages are clear. We are totally convinced that our philosophy is the best way to help our customers to be efficient and successful - wherever their products are within the product lifecycle. The practical part of my first article looked at the topic “global/central instead of local” –global settings that apply throughout the project and central settings make it possible for you to change things very quickly. Using them, you are in a position to change the design and behavior of entire projects with just a few mouse clicks and without resorting to a laborious “find and replace”. We will also use this central approach in the course of this article and you will find the approach has been improved upon still further.

The second article of the series looked at “object-orientated parameterization” (Information Unlimited, edition no. 17). Here, we saw how structure variables, arrays and the auxiliary aids of legacy and overwriting can lead to a very flexible and efficient variable set.

Now we come to the topic of this article: symbols. What is a symbol in zenon exactly? A symbol can be expressed in many ways. It can be a graphical element – for example, a right angle or a line – a numerical value or a universal regulator that has been transformed into a symbol with the help of the context menu or the “sort” tool bar. To create a symbol, the most sensible approach is to add several elements to each other. Using this approach, a symbol is a group of elements combined into a single unit and which are then characterized by common properties.

Therefore, a symbol is simply a group of elements that are treated as a single entity when, for example, they are moved or have their size changed. In principle, there are two types of symbols: embedded symbols and linked symbols. Embedded symbols are created directly in the screen. They are one-offs and are not related to the symbol library in any way. Linked symbols are different; they are not saved in the screen but in the symbol library. Only a reference to the symbol is saved in the screen. The advantage of using a linked symbol is that it can be used as often as desired: in one screen, in several screens or even in several projects.

Symbol libraries

The symbol library is no more than a save location for symbols. Users already familiar with zenon will know that there are two symbol libraries in the product: the global symbol library and the project symbol library.

The global library (Figure 1) provides symbols that can be used across multiple projects. There, the symbols are administered in groups. Each group is effectively a file that is stored in the data directory of the respective zenon Editors as a .SYM file. There are certain limitations implied - each zenon version comes with a new data directory, which means that the symbol files must be copied manually. Furthermore, the symbols are not considered a part of the project and are therefore not included when the project is backed up.

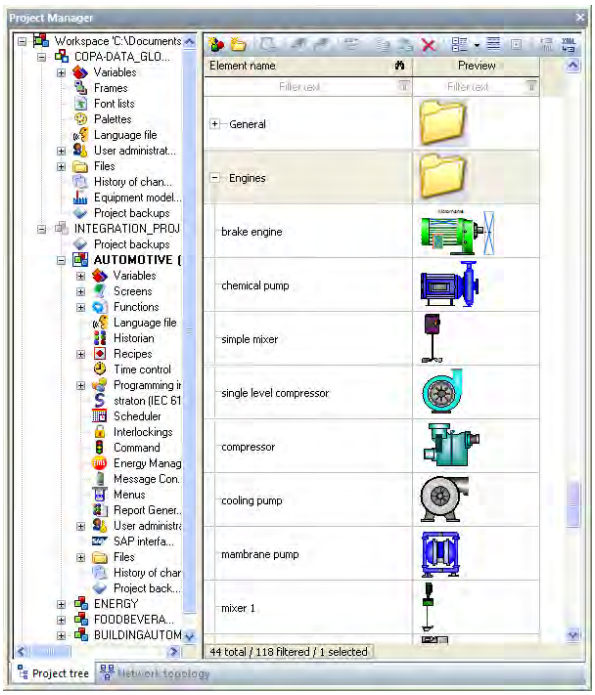


Figure 1: Symbol administration based on groups: the global symbol library in zenon.

The project symbol library is different: the symbols saved in it are treated as a component of the project and are therefore saved when the project is backed up. However, in return they cannot be used throughout several projects. It is not possible to “group” the symbols saved in the project symbol library in the same way as you are able to with the symbols saved in the global symbol library. However, the project symbol library does allow you to assign the properties “category” and “description” to the symbols saved within it, which can be effectively used to sort the symbols into groups. To do this, simply right click on

the category or description columns and select the function “Group according to this column”.

Efficient handling

At the start of the text, I explained how a symbol was created: simply select one or more screen elements and execute the “create symbol” command. But what if you later want to expand the symbol by an element? This presents no problem for the zenon user. There is a function for precisely this purpose: “Insert into existing embedded symbol”. Using this function, the selected elements are added to the symbol. It is important to note that the arrangement of the elements is retained. If the new elements are outside the previous symbol limits, the symbol is correspondingly larger.

A further very important function in relation to symbol handling is the “symbol/element individual editing mode” function. You use this to switch all embedded symbols in the screen to the individual editing mode. In this mode, you can select each of the elements in the symbol and change the properties of the individual elements: for example, adjusting a color or changing the position of an element in the symbol. The good thing about it: the symbol itself is not deleted, but remains with all its properties. If you deactivate the individual editing mode, you can edit the symbol as a whole again.

Another tip for editing: it is often the case that several elements are overlayed upon each other, so that only the uppermost is visible for editing. However, using zenon, you do not have to drag the other elements to one side in order to reach obscured elements. Simply click on the uppermost element whilst holding down the ALT key to select the element beneath it. In this way, it is possible to click through all elements that are overlayed upon each other and change their properties, using the property windows or key operation.

Working with the symbol library

Assuming we want to use the symbol that was created, not just once but several times, what do we do? First, the symbol must be inserted into one of the two symbol libraries. You do this by using the context menu, right clicking on the symbol you wish to reuse and selecting the option “Insert into symbol library”. By default, the embedded symbol remains on the screen. If it is no longer needed, it can be deleted. The symbol is now in the library and can also be edited there.

The symbol is opened in the symbol editor by double clicking on it, and it can be changed or expanded there. The symbol editor offers the same functionally as the screen editor. However, the symbol size can also be changed via the properties. The symbol can be dragged from the library to the screen using a simple „drag and drop" mechanism so that a copy of that symbol is inserted there as a linked symbol. The advantage of this is that, if the symbol is changed in the library, it is changed at all locations at the same time. Tip: it is possible to open a linked symbol for immediate editing in the symbol editor using the context menu or to jump directly to the symbol in the library via “linked elements” on the menu.

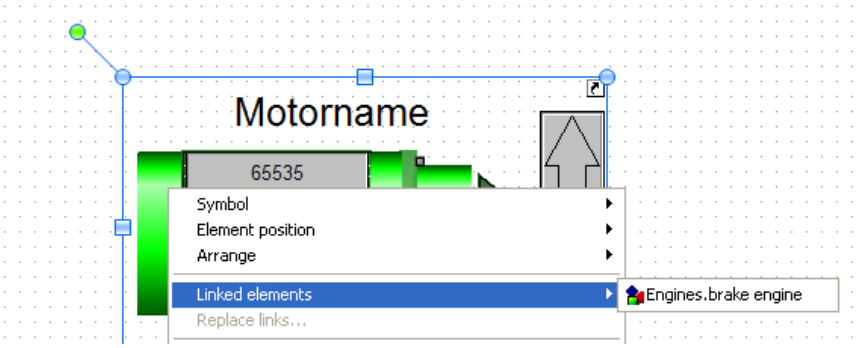


Figure 2: The symbol shows its source in the symbol library; this can be jumped to directly via the “linked symbol” menu item.

zenon v. 6.50 brought a further improvement to this symbol handling. Since that version, it is possible to insert symbols not only into a screen, but also into another symbol.

Let's use the example of a symbol that displays a pipeline cable. This cable contains five valves. You could now draw each valve separately. However, it would be considerably more efficient to work as follows: you draw the valve once and save it as a valve in the symbol library. Now, draw your pipeline cable and insert it wherever you require the “valve” symbol by dragging and dropping it. If you would like to change the valve - for example, its size, color or behavior - you now only need to do this in one place.

Naturally, you can also create several levels: a symbol can contain a symbol, which in turn can contain another symbol and so on. To move between each level more easily, you can jump one level at a time using the “linked elements” command.

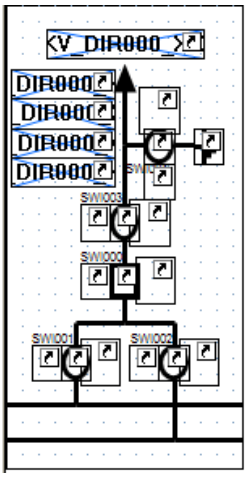


Figure 3: Branch with many linked symbols, which can be changed from a central location.

Symbols and variables/functions

Elements can naturally contain variables within the symbol. The symbol can also be linked to variables, for example, to make the symbol visible/invisible or to rotate it. In addition, operating elements such as a button or a combined element with a linked function can be contained in the symbol. These variables/functions can be changed very easily.

First, for the embedded symbol: by right-clicking on the symbol, the context menu with the “replace links” command opens. This then makes all the variables/functions of the selected symbol available to be changed. With the help of wildcards, you can carry out multiple replacements. Individual changes are also possible. By clicking on OK, these settings are saved to the symbol.

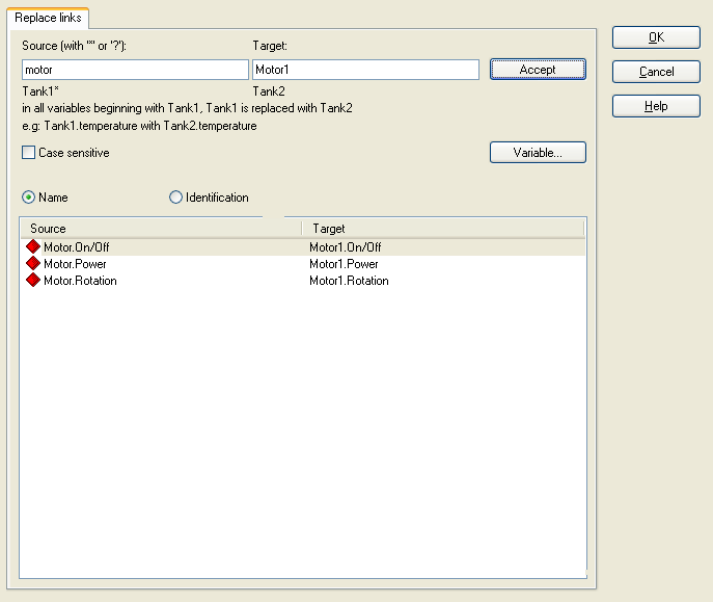


Figure 4: Links for embedded symbols can be replaced via the corresponding context menu.

Replacing variables in linked symbols

It is somewhat different with linked symbols. The symbol isn't saved in the screen itself; there is only a reference to the symbol. Thus, no variables/functions can be edited directly. However, there are two certain options that still allow for the editing or replacing of variables: either with the help of a linking rule or with the help of automatic naming (hierarchical names).

I will explain these two different methods using examples.

First, we need a variable. A structure variable is best suited for the purposes of our example, e.g. the structure „motor". This contains the structure element „power" and a second element; „speed". From this, we form a structure variable „Motor1". This consists of the variables Motor1.Power and Motor1.Speed. We now connect these two variables with, for example, a bar graph and a numerical value for the symbol stored in the library. We then create the structure variables Motor2, Motor 3 etc. Naturally, this could also be set up as an array.

1.) Linking rule

If we now insert the symbol into the screen, the “edit linking rule” dialog box (see Figure 5) appears, which is very similar to the “replace link” dialog box (see Figure 4). Using the “edit linking rule” dialog box, you can change one or more linking rules via variable assignment. In our example, we want to replace Motor1 with Motor2. To do this, we enter Motor1* at source and Motor2 (without an asterisk) at target. Tip: multiple rules are separated from each other with a semi-colon. For example, Source: Motor1*; Pump1* Target: Motor2; Pump2

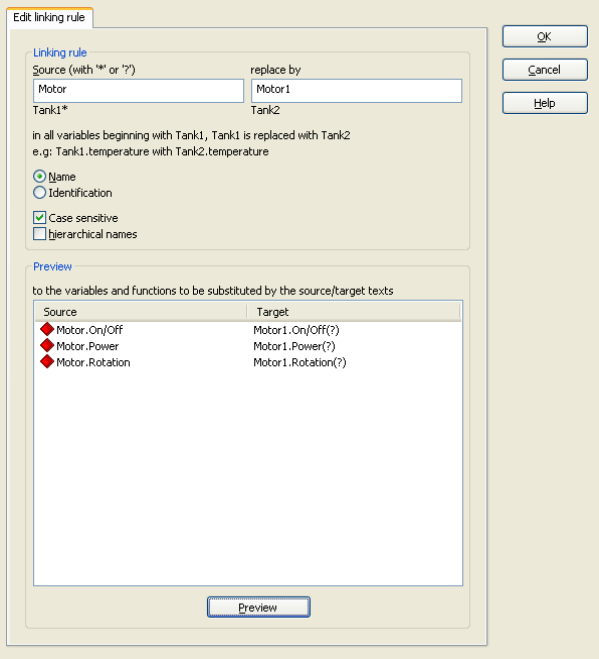


Figure 5: You can define one or more rules for linked symbols via the “Edit linking rule” context menu.

The variables Motor2.Power and Motor2.Speed are then shown in zenon Runtime. The rule can subsequently be edited using the properties window.

2.) Hierarchical names

The variable name in the symbol consists of the variable name in the element and the symbol name combined. In order for the name compound to work, the variable names must be edited in the element first. To do this, the bar graph and the numerical value must be selected and the variable name must be changed manually by entering the text. In our example, the label Motor1 must be removed: Motor1.Power becomes .Power and Motor1.Speed becomes .Speed. When inserting the symbol into the screen, the “edit linking rule” dialog box should be ignored. You then activate the “hierarchical names” property in the properties. If

you now change the name of the symbol in Motor1, Motor2 etc., the variables are automatically allocated correctly. Symbol name Motor1 + variable name .Power automatically results in: Motor1.Power.

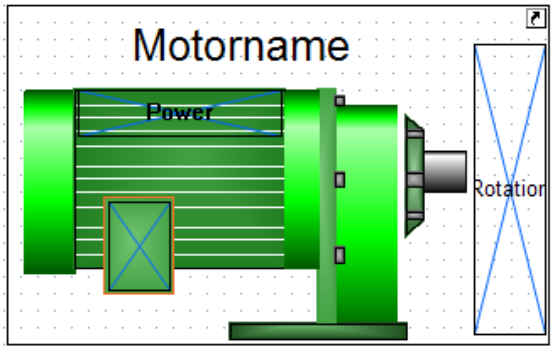


Figure 6: Symbol on the screen, without allocation of variables.

Figure 7: The names are automatically linked if "Hierarchical names" has been activated.

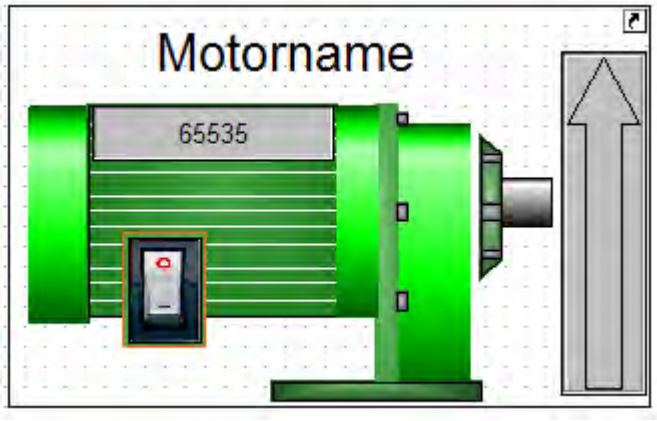


Figure 8: Symbol on the screen, with allocation of variables.

Ultimately, the two variants deliver similar results. However, with the linking rule, you have the advantage that you can replace several variables/functions at once.

Expansions are playfully easy

The topic really becomes exciting when you need to expand the symbol. Assume that another variable needs to be added in addition to speed and power; let‟s say „current consumption'.

An efficient project planner would relish such a challenge; now he can play all his trump cards. This is how it works:

1. Expand the structure data type by adding the „current consumption' structure element.
2. Activate all new variables with one mouse click (they are inactive as default).
3. Edit the symbol and add a new numerical value to represent the current consumption.
4. Save the symbol.

Finished - that‟s it! It doesn‟t matter if you display two or 1,000 motors - all motors now have a numerical value representing their current consumption. As a result of the fact that only one linking rule is saved with the symbol, this rule also applies for the new variable, Motor1.CurrentConsumption. Consequently, this variable is also replaced in the Runtime. You can also apply this technique when working with hierarchical names.

Several screens

zenon also offers a solution if you do not have space to display all of your motors on a single screen. Obviously, you do not need to draw a separate screen for each motor. One single screen is sufficient. When switching screens, you can again use the “replace link” dialog. Example: If you have Motor1 and Motor2 displayed on your screen, simply replace Motor1 with Motor3 and Motor2 with Motor4 when switching screens.

Last but not least: approving properties

The highlight of zenon‟s improved symbol handling is the option of individually adapting the properties of a linked symbol on the screen. This works in a similar way to overwriting the properties of variables that are derived from the data type (see part 2 of this series).

The highlight of zenon‟s improved symbol handling is the option of individually adapting the properties of a linked symbol on the screen. This works in a similar way to overwriting the properties of variables that are derived from the data type (see part 2 of this series).

If we continue to work with our example detailed above, imagine we would like to give our motor symbol a caption, so that the user immediately recognizes which motor it is. Therefore, we create a static text and enter „Motor‟ as the text. All motors on the screen now feature the heading of „Motor‟. The symbol is linked to the library. Therefore, we cannot change the text individually for each motor symbol. To edit the text of an individual symbol, we must first approve the text property. This is also achieved by a „drag and drop‟ mechanism. Simply select the property and drag it to the area below the symbol where all approved properties are administered.

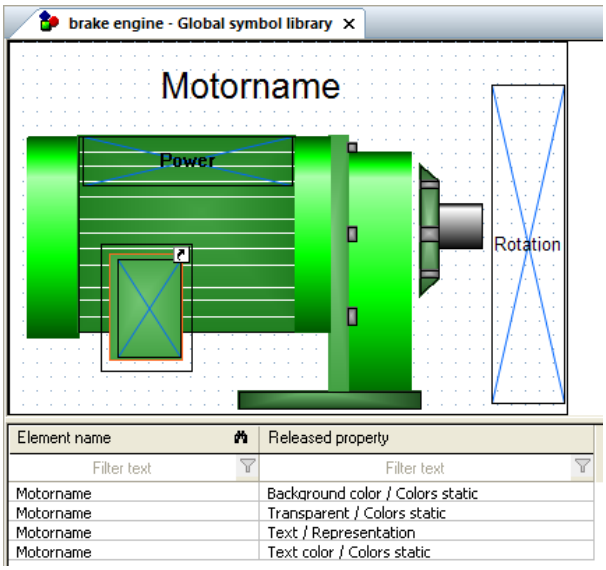


Figure 9: Symbol in the global symbol library with approved (released) properties.

If we now drag the symbol into the screen, or double click on a symbol on the screen, an assistant appears that lets us change the property dynamically. Therefore, it is easy to rename the motors as Motor1, Motor2 etc. These

approved properties can naturally also be changed by means of the properties window. There, they appear specifically marked as additional symbol properties.

It is also possible to allocate variable name and variable identification automatically with the syntax %n and %l. For further details also find information in the help at: Manual -> Screens -> Screen elements -> Combined element -> Display of variable name and variable identification

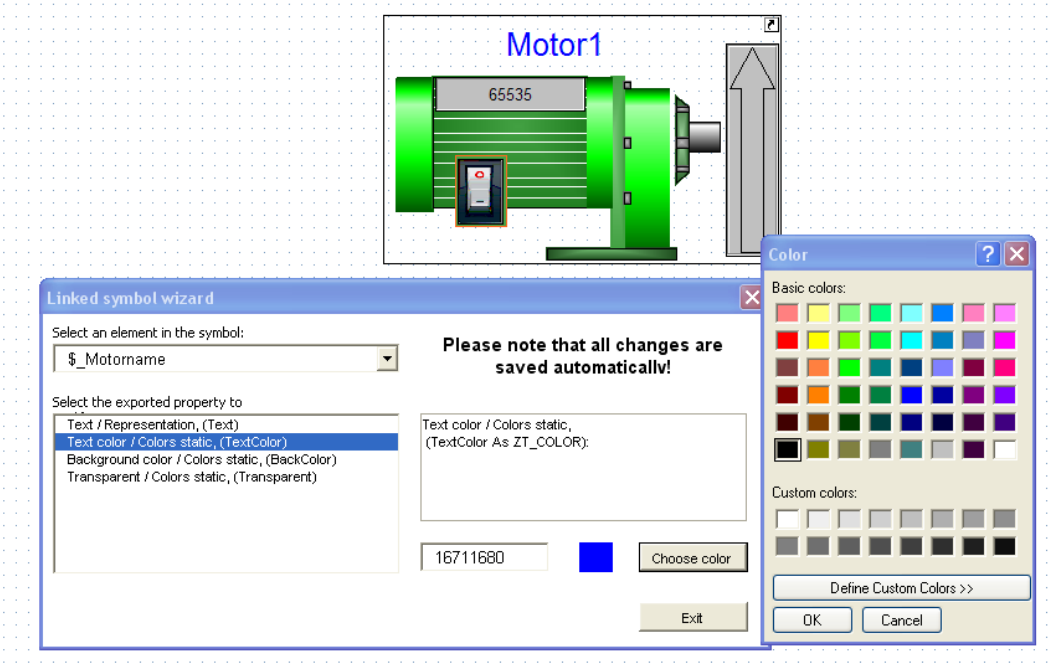


Figure 10: With the help of the assistant, any desired symbol property - for example the color (as shown in the picture) - can be changed.

Conclusion

Symbol administration in zenon is a powerful tool, designed to help you create graphical objects efficiently – and then to administer them. The advantages of the mechanisms outlined in this article are particularly evident if something has been changed or needs to be adjusted. Yet again, zenon proves that its philosophy of "engineering instead of programming" pays dividends for the canny developer and enables its users to freely implement their ideas very quickly and easily.

Have fun configuring!

Dieser Artikel ist in einer älteren Version im IU Magazin No. 19 erschienen.