# Automated Engineering

Quicker results with wizards

www.copadata.com
sales@copadata.com

zenon
do it your way

## Content

1. THE PROGRAMMING INTERFACE AS A BASIS FOR THE GENERATOR ....... 2

2. THE ADVANTAGES OF A WIZARD ...................................................................... 2

3. RULES OF PROJECT CONFIGURATION: WHO? HOW? WHAT? WHERE? .... 4

4. SIMPLE CREATION OF THE GENERATOR ....................................................... 6

5. QUICK START – QUICK SUCCESS ..................................................................... 9

The "zenon automotive generator" (zag) is a wizard for the automated analysis of PLC data and implementation of visualization projects. It is primarily used in the automotive industry, which is traditionally very highly orientated towards standardized components and reuse.

The automation of engineering processes allows considerable time and cost savings when implementing a project. The deadlines, which are often tight, can thus be adhered to more easily. The person configuring the project can save simple and repetitive tasks in the wizard – thus leaving more time for demanding activities and the risk of incorrect project configuration is minimal.

In this White Paper, I will describe a few aspects of the creation of a wizard such as the "zenon automotive generator". In addition to the technical implementation, a few prior considerations are listed and the backgrounds are described.

## 1. The programming interface as a basis for the generator

The programing interface (API) of zenon is popular for task-specific expansions. In addition to the zenon Runtime API, there is also a corresponding interface to the zenon Editor that is available for the automation of engineering tasks. Examples of this are the wizards supplied with the zenon Editor. The openness of zenon allows any user of the Editor to create their own wizard or to amend existing ones on a project-by-project basis. The integrated programming languages such as VBA or VSTA (C# or VB.NET) are available for this. There is also a template of a wizard for programming with the Microsoft Visual Studio.

When selecting the programming language, "Know-How Protection" can also play a role. This way, the wizard can be created for amendment by third parties, in a type of "open source", or it is used in compiled form.

## 2. The advantages of a wizard

A major advantage when using a wizard is the reduced project configuration time. The duration of automatic project generation is usually only a fraction of the time required for manual project creation. There is also another time saving: the generator can read the information and interpret the saved rules considerably more quickly than a human could.

In order to determine the "added value" of the wizard, this time advantage – and thus the working time saved as a result – is set in relation to the time required to create the wizard. There is a linear relationship to the number of projects generated:

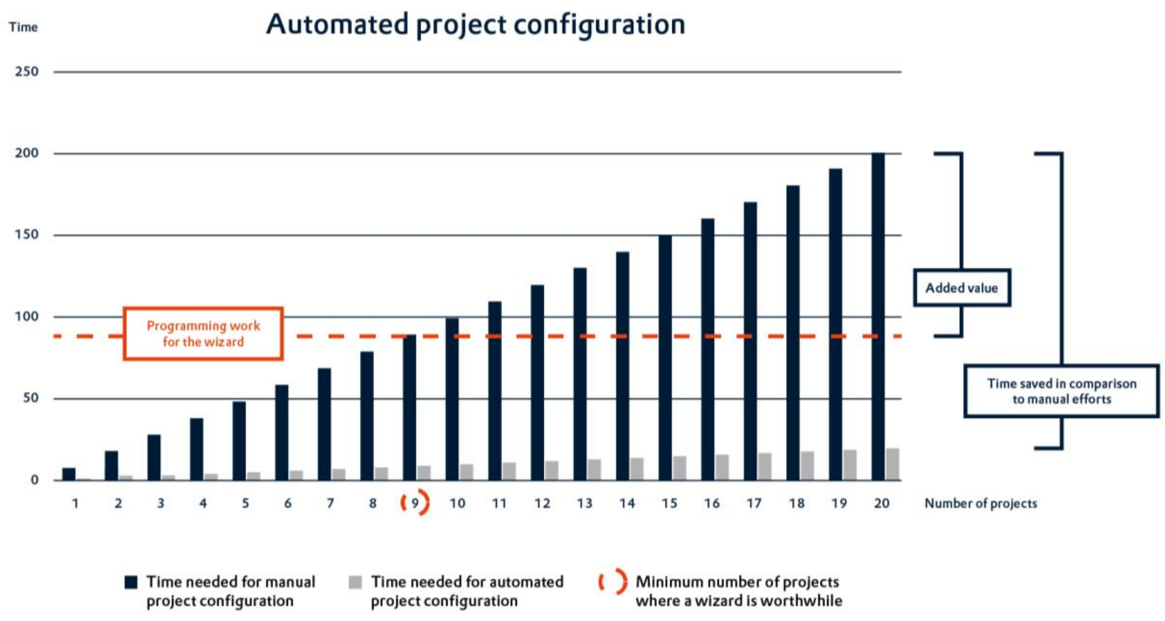


*Figure 1: Time saving through automated project configuration*

In this diagram, a ratio of 1 to 10 is assumed: whilst the manual activity requires 10 time units, the wizard only needs 1 time unit. With 10 projects, this therefore results in a time saving of 90 time units. If the estimation of time for the creation of code is connected to this curve, it is possible to see the projects where such a wizard is worthwhile. In this example, there is added value from the creation of a wizard from the tenth project onwards.

The working time saved with a wizard is frequently seen as the biggest advantage, which can also be measured directly. However, there are further advantages that cannot be measured directly, but have a positive effect on the quality of the projects:

- The wizard will always create the project precisely according to the saved rules. Errors that result from manual work are completely out of the question.
- The users of the wizard do not need any in-depth expertise for the project or the systems used. Depending on the user interface, brief instructions for execution are frequently sufficient.

- The wizard reads a data source for the evaluation of the information. If this cannot be interpreted correctly, this is due to incorrect information. The wizard thus carries out an additional quality check on the data.

## 3. Rules of project configuration: Who? How? What? Where?

Before the actual programming, the "rules" of project generation must be defined as precisely as possible. These describe which information is found where, and how this is to be interpreted and applied. Such rules are also applied for manually-created projects. However these are frequently not documented clearly enough for them to be able to be used for the creation of the generator. When creating a wizard, these rules are programmed so that they are applied exactly during execution. The wizard serves as an "interpreter" of a data source here, and as a translator in a zenon project.

Once these rules have been established, they can be documented for translation into the actual program code in a flow chart. This illustration describes the individual steps for processing by the wizard.

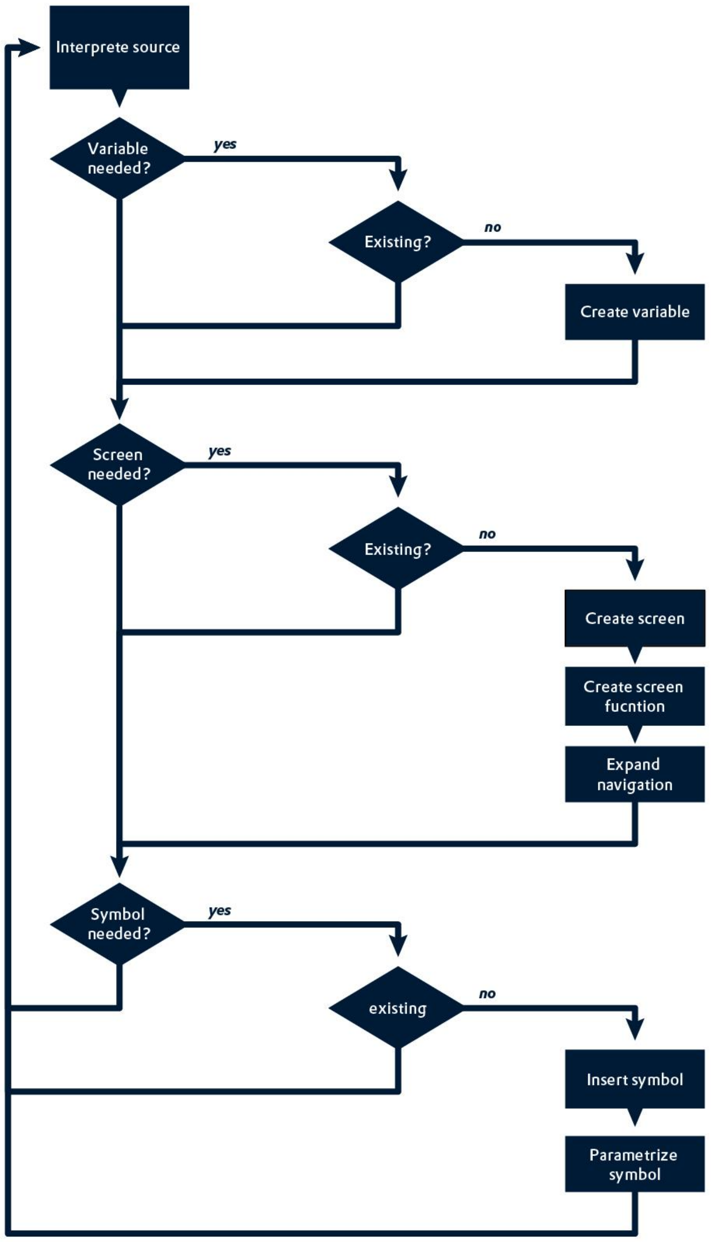


*Figure 2: Example of a flow chart as a basis for a wizard*

Corresponding work needs to be taken into account for the actual reading of the data source. For access to Excel files, there are many examples of code in the Internet. However, if it is a matter of accessing data in PLC programming software, sometimes a search using the access methods and data storage is necessary.

In addition to the description of the actions of the wizard, the programmer should also have a functional user interface. The user is informed of the actions of the wizard whilst it carries out its work in the background. The user can use the interface to configure any possible parameters, or it informs them of the progress of generation.

## 4. Simple creation of the generator

The programmer has the different zenon objects available for the creation of code. These are documented in the online help. The respective properties can also be found in the embedded help in zenon. These properties can be used to amend the respective zenon objects.

For the creation of the different project parts, the objects are created accordingly and given values. For example, variables are created with this VBA code:

```
Set zVar = MyWorkspace.ActiveDocument.Variables.CreateVar(a, b, c, d)
```

Here, the placeholders correspond to the following objects:

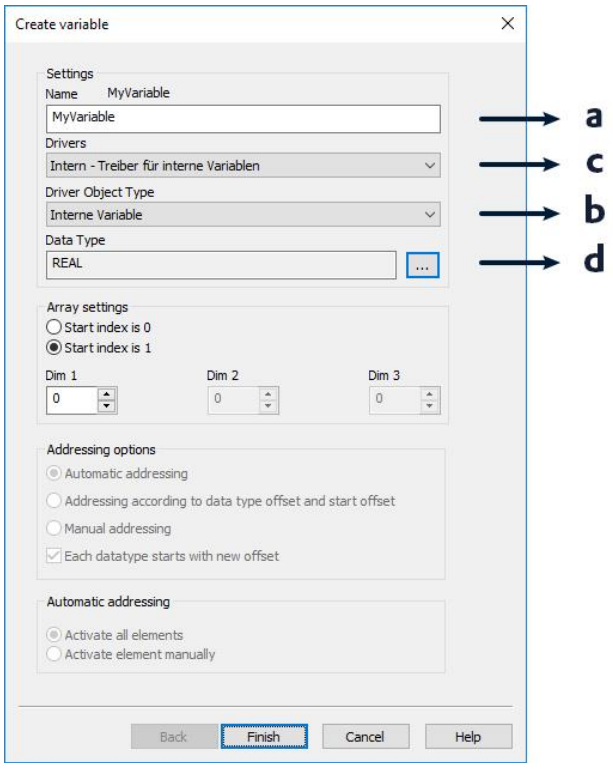
a = the name of the variable to be created

b = the driver object on which the variable is to be based

c = the driver object type of the variable

d = the data type of the variable

This information corresponds to the same parameters that are also to be entered when creating a variable in the zenon Editor:



Create variable

Settings
Name MyVariable
MyVariable
Drivers
Intern - Treiber für interne Variablen
Driver Object Type
Interne Variable
Data Type
REAL

Array settings
Start index is 0 [✓ Start index is 1]
Dim 1: 0
Dim 2: 0
Dim 3: 0

Addressing options
Automatic addressing
Addressing according to data type offset and start offset
Manual addressing
✓ Each datatype starts with new offset

Automatic addressing
Activate all elements
Activate element manually

*Figure 3: Parameter of a variable in zenon*

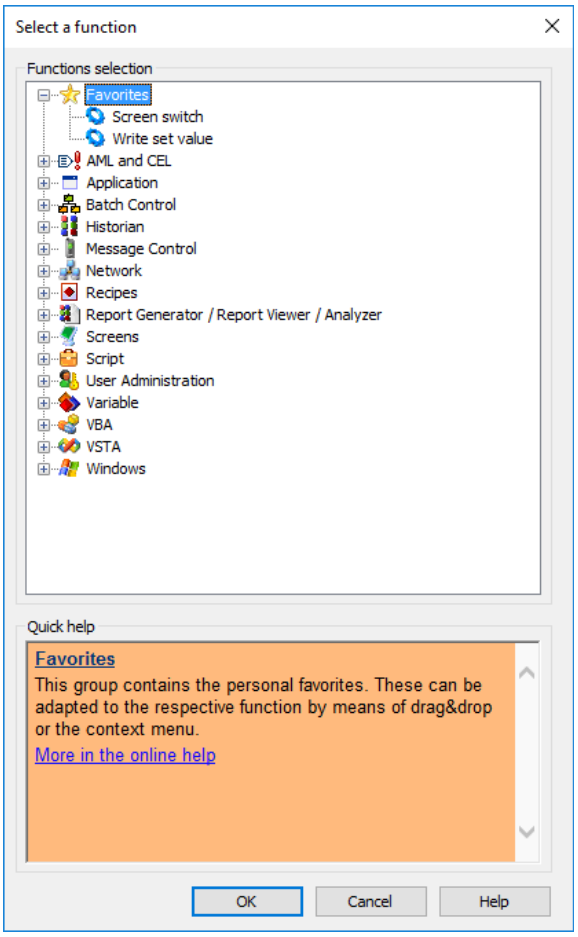
The parameter a in the above code corresponds to the "Name" field in this dialog. The parameter b corresponds to the "Driver Object Type" field, parameter c corresponds to the "Driver" combobox and parameter d corresponds to the "Data Type".

Further properties of this variable are amended using the "DynProperties" of the respective objects. To amend the variable recognition, for example, the respective property must be determined and configured. The name of this property can be determined in the zenon Editor from the embedded help. If you click on the variable detection of a variable, the name of the property ("Tagname" in this case) is shown underneath the actual help text. This can be amended with

```
zVar.DynProperties(„Tagname“) = "automotive"
```

accordingly.

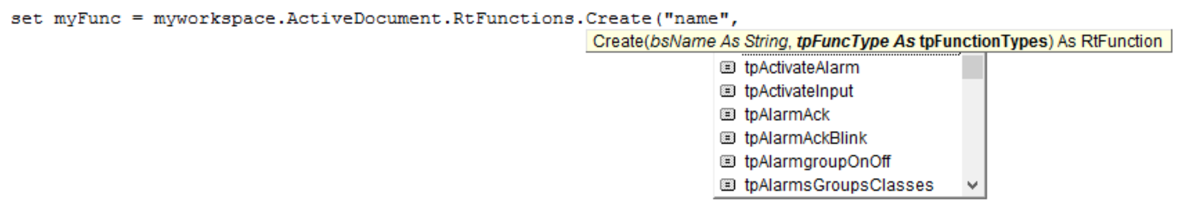
For individual, constant parameters, the programmer has these available as constants when creating code. An example of this is the different function types when creating a new zenon function:



Favorites
This group contains the personal favorites. These can be adapted to the respective function by means of drag&drop or the context menu.
More in the online help

*Figure 4: Window for selecting the zenon functions*

If you create a new function code, in addition to the name of the function, the function type is transferred as a constant parameter.



```
set myFunc = myworkspace.ActiveDocument.RtFunctions.Create("name",
                           Create(bsName As String, tpFuncType As tpFunctionTypes) As RtFunction
                                          tpActivateAlarm
                                          tpActivateInput
                                          tpAlarmAck
                                          tpAlarmAckBlink
                                          tpAlarmgroupOnOff
                                          tpAlarmsGroupsClasses
```

*Figure 5: Creation of a function with code*

Then, depending on the function type, the attendant properties are again configured using "DynProperties".

```
Set myFunc = myWorkspace.ActiveDokument.RtFunctions.Create(“func-name”,
tpPicture)

myFunc.DynProperties(“Picture”) = “screename”
```

With this code, a new function with the name "func-name", a screen switching type (= Konstante tpPicture), is created. The screen with the name "screenname" is used.

The wizard code can be created quickly with this procedure. Creation of zenon objects and setting the parameters for them is always carried out in accordance with this model.

In order to design the runtime of the generator as more stable, some requests for confirmation should be integrated. For example, no new variables can be created with the name of existing variables. This is why I recommend implementing such requests and reacting to them.

## 5. Quick start – quick success

With the various examples of code in the zenon online help and in the COPA-DATA forum, the first steps in the creation of a generator are completed quickly. With the definition of rules and the implementation in the wizard, the programmer gets the first version in a short time. The user can thus collect initial positive experiences.

A large amount of time can be saved in the project creation phase by using wizards. Errors that occur due to tasks always being the same are prevented by the automation of processes.