

# ZenOn 7.20 Manual

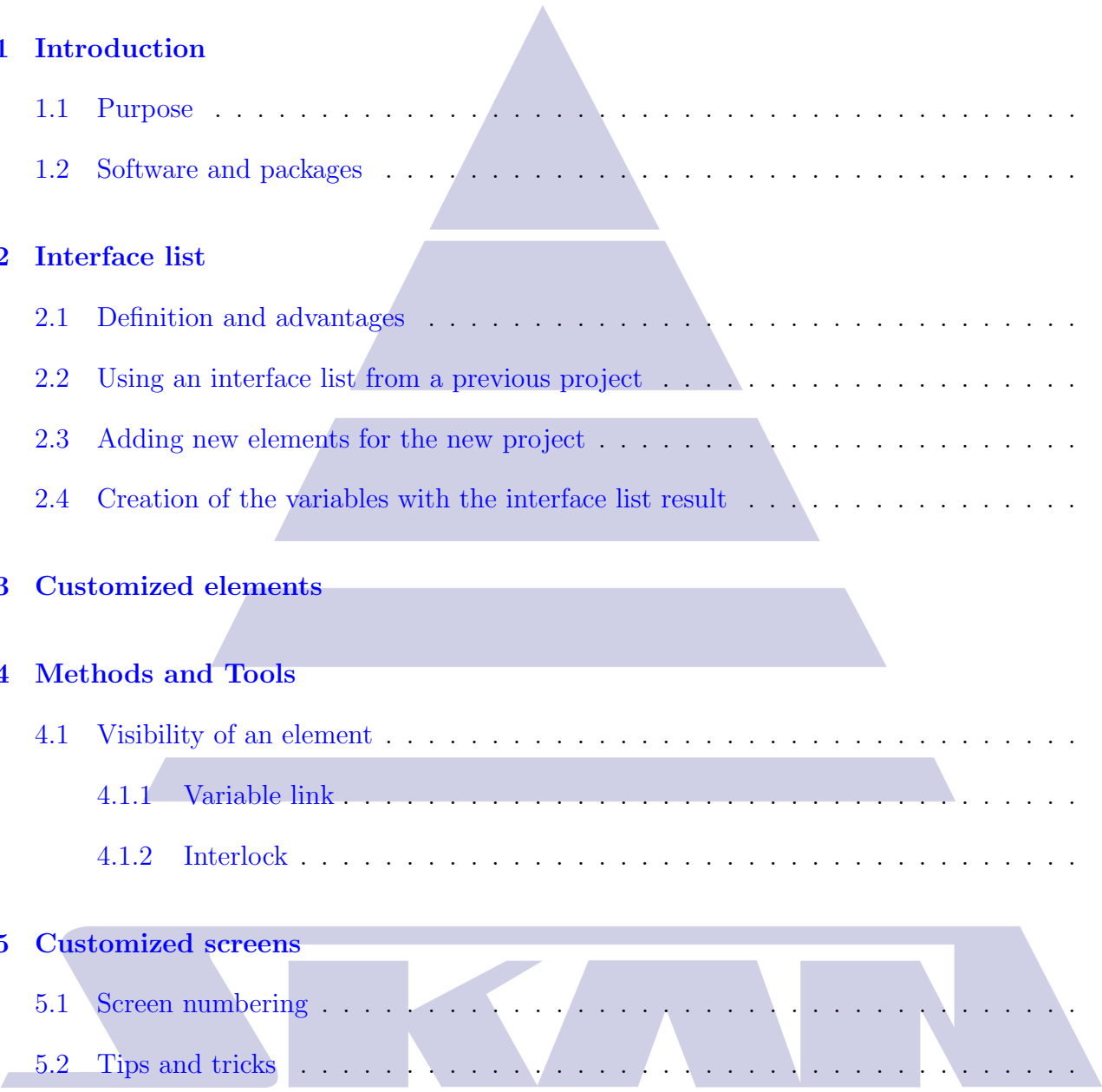
Daniel Barmaimon  
with contributions from  
Viktor Boger

July, 2018



**zenon**  
do it your way

# Contents



<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Purpose . . . . .	3
1.2	Software and packages . . . . .	3
<b>2</b>	<b>Interface list</b>	<b>5</b>
2.1	Definition and advantages . . . . .	5
2.2	Using an interface list from a previous project . . . . .	5
2.3	Adding new elements for the new project . . . . .	7
2.4	Creation of the variables with the interface list result . . . . .	10
<b>3</b>	<b>Customized elements</b>	<b>12</b>
<b>4</b>	<b>Methods and Tools</b>	<b>13</b>
4.1	Visibility of an element . . . . .	13
4.1.1	Variable link . . . . .	13
4.1.2	Interlock . . . . .	14
<b>5</b>	<b>Customized screens</b>	<b>16</b>
5.1	Screen numbering . . . . .	16
5.2	Tips and tricks . . . . .	17
5.2.1	Switch screen with substitution parameter . . . . .	17
5.3	Service Screen . . . . .	19
5.3.1	P800 - SKAN AG info . . . . .	19

5.3.2	P810 - Service functions	20
5.3.3	P820 - Force mode	25
<b>6</b>	<b>Trends</b>	<b>26</b>
<b>7</b>	<b>Alarms</b>	<b>27</b>
7.1	Alarm Matrix - ProFile Document	27
7.2	List manipulation with Excel	27
7.3	Importing into ZenOn	32
7.4	Special requirements	37
7.4.1	Single alarm acknowledgment	37
7.4.2	Acknowledgment class E alarm only for some users	37
<b>8</b>	<b>CEL: Chronological Event List</b>	<b>38</b>
<b>9</b>	<b>Reports</b>	<b>39</b>
9.1	Introduction	39
9.2	Report Definition Files (RDL)	40
9.2.1	RDL creation and dataset configuration	40
9.2.2	Archives for reports	42
9.2.3	Report layout and configuration	42
9.2.4	Customized fields	44
9.3	Models for report generation	48
9.3.1	Lot archive with allocations	48
<b>10</b>	<b>VBA scripts</b>	<b>53</b>
10.1	Introduction	53
10.2	Decontamination cycle simulation	53
10.2.1	How does it work?	54
10.2.2	How can we use it?	55

# Chapter 1

## Introduction

### 1.1 Purpose

The current manual has been created for internal use of the automation department at SKAN. The main purpose of the manual is to help using ZenOn 7.20 for the development of our projects, reducing the learning and troubleshooting time.

The document will be in constant development and improvements and modifications could be done by any person related to the department. Please keep a copy of the last document (both *.pdf* and *.tex* files) before making any modification.

### 1.2 Software and packages

The current documentation has been written using  $\text{\LaTeX}$ . In order to be able to modify it there some initial steps should be followed.

1. Install LaTeX package manager (recommended **MiKTeX**)
2. Install LaTeX editor and environment (recommended IDE **TeXstudio**)
3. Install the external packages listed below. If a new package is used the list below should be updated accordingly.
  - caption
  - excel2Latex
  - pgf
  - ms
  - xcolor
  - mptopdf

- tikz
- titlesec
- titlepic
- background

If help is needed during this process, please take a look at this link [How to write a thesis using latex](#)

# Chapter 2

## Interface list

### 2.1 Definition and advantages

The interface list is a document that collects all the signals that are going to be integrated in the project. The main advantages of using this document are the following:

1. **PLC - HMI link** Mapping between the two environments is defined in a clear way
2. **Automatic address setting** Avoid issues of duplicated addresses or inefficient memory management.
3. **Time optimization** By using previous project's interface list

### 2.2 Using an interface list from a previous project

First step is to look for a similar project in ProFile and copy the interface list into our project (in case it was not done before by the project manager or the PLC engineer).

There will be no possibility of working on-line on the file if someone else have it as *In progress* in ProFile. In that case just create a local copy in your computer.

The document looks like in Fig. 2.2

At a first glance the main Excel sheet could be divided into three different groups of columns that could be clearly differentiated by the header's color:

- **Yellow** Element description
- **Orange** PLC interface data bloc
- **Green** SCADA variable configuration

WF	Project number	Project name	Customer	Customer address	Project type	PM Skan	Creation date	Created by
106-17-453	MSEZ-3-FV Vial Filling Line Isolator	Serum Institute of India Pvt. Ltd	Mangar BK. Tal. Haveli, Pune, India	Filling Line Isolator	Matthias Bauer	16.01.2018	huetma1	

Preview	Vp Sp	Proj-No	Profile D-ID	SKAN D-ID	Document title	Ver	Re	Edited on	Edited by	W Status	WF on	WF by	Category	Type of Document	Created by
		106-17-453	564658	283128	HDS - Hardware Design Specification - Isolator	A	01	25.Mai.18	jeanm1	In Progress			Specification	HDS - Hardware Design Specificat...	jeanm
		106-17-453	564658	283130	Herstellungsspezifikation Stahlbau	A	01	30.Mai.18	helbma1	Released	30.Mai.18	helbma1	QHB Document	FORM	helbm
		106-17-453	564658	283130	HMI Design Specification Isolator	A	01			In Progress			Specification	HMI Specification	jeanm
		106-17-453	564660	283131	HMI Network	A	01			In Progress			Specification	Design Specification	jeanm
		106-17-453	564661	283132	HMI Parameter List Isolator	A	01			In Progress			List	General List	jeanm
		106-17-453	564662	283133	IO-List Isolator	A	01			In Progress			List	General List	jeanm
		106-17-453	568304	277163	Instrument List of Sensors	A	02	07.Jun.18	bauema1	Draft	07.Jun.18	bauema1	List	Sensor List	baue
		106-17-453	564656	283130	Interface List (with SCADA) Isolator/Airlock	A	01			In Progress			Specification	Interface	jeanm
		106-17-453	564663	283134	Interface List (with third-party systems) Isolator	A	01			In Progress			Specification	Interface	jeanm
		106-17-453	534199	275168	IQ-Master-plan Filling Line Isolator (GAMP 5)	A	01	18.Jan.18	bechma2	Released	18.Jan.18	bechma2	QHB Document	FORM	bechr
		106-17-453	534212	275178	IR-Investigation Report	A	01	18.Jan.18	bechma2	Released	18.Jan.18	bechma2	QHB Document	FORM	bechr

Figure 2.1: Example of how to find the interface list in ProFile

Version control	Element description	element configuration	comment	PLC interface data block	variable tag	variable address	Tag name
	pic_L1 Fan Status	1100 HVAC		A_SCADA_IF_DM_STS	Status_I	502 30 0	pic_L1/Fan/Status-1100
	pic_L1 Fan CmdRev	1100 HVAC		A_SCADA_IF_DM_CMD	Cmd	505 1 2	pic_L1/Fan/CmdRev-1100
	pic_L1 Fan SpRev	1100 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 24 0	pic_L1/Fan/SpRev-1100
	pic_L1 Fan Sp1	1100 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 26 0	pic_L1/Fan/Sp1-1100
	pic_L1 Fan Sp2	1100 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 28 0	pic_L1/Fan/Sp2-1100
	pic_L1 Fan Sp3	1100 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 30 0	pic_L1/Fan/Sp3-1100
	pic_L1 Fan Sp4	1100 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 32 0	pic_L1/Fan/Sp4-1100
	pic_L1 Fan ActVal	1100 HVAC		A_SCADA_IF_DM_ACT	Act_Val_I	501 18 0	pic_L1/Fan/ActVal-1100
	pic_L1 Fan Status	1150 HVAC		A_SCADA_IF_DM_STS	Status_I	502 32 0	pic_L1/Fan/Status-1150
	pic_L1 Fan CmdRev	1150 HVAC		A_SCADA_IF_DM_CMD	Cmd	505 1 3	pic_L1/Fan/CmdRev-1150
	pic_L1 Fan SpRev	1150 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 34 0	pic_L1/Fan/SpRev-1150
	pic_L1 Fan Sp1	1150 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 36 0	pic_L1/Fan/Sp1-1150
	pic_L1 Fan Sp2	1150 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 38 0	pic_L1/Fan/Sp2-1150
	pic_L1 Fan Sp3	1150 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 40 0	pic_L1/Fan/Sp3-1150
	pic_L1 Fan Sp4	1150 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 42 0	pic_L1/Fan/Sp4-1150
	pic_L1 Fan Status	1160 HVAC		A_SCADA_IF_DM_STS	Status_I	502 34 0	pic_L1/Fan/Status-1160
	pic_L1 Fan CmdRev	1160 HVAC		A_SCADA_IF_DM_CMD	Cmd	505 1 4	pic_L1/Fan/CmdRev-1160
	pic_L1 Fan SpRev	1160 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 44 0	pic_L1/Fan/SpRev-1160
	pic_L1 Fan Sp1	1160 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 46 0	pic_L1/Fan/Sp1-1160
	pic_L1 Fan Sp2	1160 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 48 0	pic_L1/Fan/Sp2-1160
	pic_L1 Fan Sp3	1160 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 50 0	pic_L1/Fan/Sp3-1160
	pic_L1 Fan Sp4	1160 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 52 0	pic_L1/Fan/Sp4-1160
	pic_L1 Fan ActVal	1160 HVAC		A_SCADA_IF_DM_ACT	Act_Val_I	501 22 0	pic_L1/Fan/ActVal-1160
	pic_L1 Fan Status	1170 HVAC		A_SCADA_IF_DM_STS	Status_I	502 36 0	pic_L1/Fan/Status-1170
	pic_L1 Fan CmdRev	1170 HVAC		A_SCADA_IF_DM_CMD	Cmd	505 1 5	pic_L1/Fan/CmdRev-1170
	pic_L1 Fan SpRev	1170 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 54 0	pic_L1/Fan/SpRev-1170
	pic_L1 Fan Sp1	1170 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 56 0	pic_L1/Fan/Sp1-1170
	pic_L1 Fan Sp2	1170 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 58 0	pic_L1/Fan/Sp2-1170
	pic_L1 Fan Sp3	1170 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 60 0	pic_L1/Fan/Sp3-1170
	pic_L1 Fan Sp4	1170 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 62 0	pic_L1/Fan/Sp4-1170
	pic_L1 Fan ActVal	1170 HVAC		A_SCADA_IF_DM_ACT	Act_Val_I	501 24 0	pic_L1/Fan/ActVal-1170
	pic_L1 Fan Status	1200 HVAC		A_SCADA_IF_DM_STS	Status_I	502 38 0	pic_L1/Fan/Status-1200
	pic_L1 Fan CmdRev	1200 HVAC		A_SCADA_IF_DM_CMD	Cmd	505 1 6	pic_L1/Fan/CmdRev-1200
	pic_L1 Fan SpRev	1200 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 64 0	pic_L1/Fan/SpRev-1200
	pic_L1 Fan Sp1	1200 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 66 0	pic_L1/Fan/Sp1-1200
	pic_L1 Fan Sp2	1200 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 68 0	pic_L1/Fan/Sp2-1200
	pic_L1 Fan Sp3	1200 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 70 0	pic_L1/Fan/Sp3-1200
	pic_L1 Fan Sp4	1200 HVAC		A_SCADA_IF_DM_SET	Set_Val_I	504 72 0	pic_L1/Fan/Sp4-1200
	pic_L1 Fan ActVal	1200 HVAC		A_SCADA_IF_DM_ACT	Act_Val_I	501 26 0	pic_L1/Fan/ActVal-1200
	pic_L1 Fan Status	1210 HVAC		A_SCADA_IF_DM_STS	Status_I	502 40 0	pic_L1/Fan/Status-1210

Figure 2.2: General view of the interface list

Each one of the four first columns in *Element description* section will be part of the variables' names in the HMI environment as is can be checked in *SCADA variable configuration* part.

Most of the document's information will be reused if the project to work in is similar to a previous one.

The background color of the *Element description* section and *Version Control* is just an indicator for the SCADA engineer / PLC engineer / project manager of what is the current state of the document's development, or just to share some remarks. In this case it is recommended to remove all the background color and comments when starting to work with an interface list for a new project.

Let's focus on the tabs at the bottom of the document. There are as many tabs as elements could be found in the project, as for example *isolator (IL Isolator)*, *airlock (IL Airlock)*. The tabs with a red background color as *ALARMS* or *TEXT LISTS* are not needed at the moment. Another important tab is *Konfig* (Fig. 2.3) in which the starting data-block addresses are defined for each kind of variable.

	A	B	C	D	E	F	G	H	I	J
1										
2					DB	DBx	Anzahl Bits		zenon Typ	
3	A_SCADA_IF_DM_ACT	Act_Val_I		501	0		16		INT	
4		Act_Val_D		501	200		32		DINT	
5		Act_Val_R		501	600		32		REAL	
6		Act_Val_S		501	1000		defined manually		STRING	
7	A_SCADA_IF_DM_STS	Status_B		502	0		1		BOOL	
8		Status_I		502	26		16		INT	
9		Status_D		502	426		32		DINT	
10	A_SCADA_IF_DM_SET	Set_Val_I		504	0		16		INT	
11		Set_Val_D		504	400		32		DINT	
12		Set_Val_R		504	600		32		REAL	
13		Set_Val_S		504	1000		defined manually		STRING	
14	A_SCADA_IF_DM_CMD	Cmd		505	0		1		BOOL	
15										

Figure 2.3: Configuration tab in the interface list

There are four different group of variables namely *Active values*, *Status values*, *Set values* and *Commands*. As for the data types it is possible to find *boolean*, *integer*, *double integer*, *real* and *string*.

*Linked data* tab was used in the past to link the different reaction matrices and colors with specific definitions in ZenOn. These are obsolete and are not used anymore.

**Important** To be able to link the variables that are going to be created through this interface list to specific reaction matrix and colors in ZenOn it is mandatory to have them already defined in ZenOn. Otherwise the links to these variables will be lost and it will be needed to make this connection one by one manually.

## 2.3 Adding new elements for the new project

It is possible to delete all the variables in ZenOn, use then the interface list to help the generation once again (using a couple of scripts that will be shown in this section). This is **strongly not recommended**. The reason is that if there were any script or special property linked to an existing variable, this link will be lost during the process. It is better to overwrite the variables than starting from scratch.

The only con regarding this procedure is that it could happen that several variables from a previous project are not needed any more. At the moment this manual was written there was no special mechanism to automatically delete the extra variables.

In case new known elements are needed for the new project (i.e. an extra fan), the cells should be selected as rows copy and paste. Remember to change the values of the PI&D numbers (at the



detail 3 column of the *Element description* group). In the Fig. 2.4 can be seen how to select a whole element, and how to copy it after right-click over it.

Version control	Element description				comment
connection name	detail 1	detail 2	detail 3		
plc_L1	Fan	Sp1	1.220		
plc_L1	Fan	Sp2	1.220		
plc_L1	Fan	Sp3	1.220		
plc_L1	Fan	Sp4	1.220		
plc_L1	Fan	ActVal	1.220		
plc_L1	Fan	Status	1.230		
plc_L1	Fan	CmdRev	1.230		
plc_L1	Fan	SpRev	1.230		
plc_L1	Fan	Sp1	1.230		
plc_L1	Fan	Sp2	1.230		
plc_L1	Fan	Sp3	1.230		
plc_L1	Fan	Sp4	1.230		
plc_L1	Fan	ActVal	1.230		
plc_L1	Fan	Status	1.240		
plc_L1	Fan	CmdRev	1.240		
plc_L1	Fan	SpRev	1.240		
plc_L1	Fan	Sp1	1.240		
plc_L1	Fan	Sp2	1.240		
plc_L1	Fan	Sp3	1.240		
plc_L1	Fan	Sp4	1.240		
plc_L1	Fan	ActVal	1.240		
plc_L1	Flap	Status	1.500		
plc_L1	Flap	CmdRev	1.500		

data bloc	type	elmt n°	variable
A_SCADA_IF_DM_SET	Set_Val_I	44	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	45	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	46	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	47	A_SCAD
A_SCADA_IF_DM_ACT	Act_Val_I	16	A_SCAD
A_SCADA_IF_DM_STS	Status_I	10	A_SCAD
A_SCADA_IF_DM_CMD	Cmd	18	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	48	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	49	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	50	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	51	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	52	A_SCAD
A_SCADA_IF_DM_ACT	Act_Val_I	17	A_SCAD
A_SCADA_IF_DM_STS	Status_I	11	A_SCAD
A_SCADA_IF_DM_CMD	Cmd	19	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	53	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	54	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	55	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	56	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	57	A_SCAD
A_SCADA_IF_DM_ACT	Act_Val_I	18	A_SCAD
A_SCADA_IF_DM_STS	Status_I	12	A_SCAD
A_SCADA_IF_DM_CMD	Cmd	20	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	58	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	59	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	60	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	61	A_SCAD
A_SCADA_IF_DM_SET	Set_Val_I	62	A_SCAD
A_SCADA_IF_DM_ACT	Act_Val_I	19	A_SCAD
A_SCADA_IF_DM_STS	Status_I	13	A_SCAD
A_SCADA_IF_DM_CMD	Cmd	21	A_SCAD

Figure 2.4: Process to duplicate an element for the interface list.

After this step it will be necessary to remove the *element number* and the *variable address* for this new element as shown in Fig. 2.5. The background color of the cells to remove is light blue when is properly set by the macro and red when is not established yet.

Version control	Element description				comment	data bloc	type	elmt n°	variable tag	variable address DB	Offset	Bit	SCADA variable config
connection name	detail 1	detail 2	detail 3										Tag name
plc_L1	Fan	Sp1	1.220			A_SCADA_IF_DM_SET	Set_Val_I	44	A_SCADA_IF_DM_SET.Val_I[44]	504	86	0	plc_L1/Fan/Sp1-1.220
plc_L1	Fan	Sp2	1.220			A_SCADA_IF_DM_SET	Set_Val_I	45	A_SCADA_IF_DM_SET.Val_I[45]	504	88	0	plc_L1/Fan/Sp2-1.220
plc_L1	Fan	Sp3	1.220			A_SCADA_IF_DM_SET	Set_Val_I	46	A_SCADA_IF_DM_SET.Val_I[46]	504	90	0	plc_L1/Fan/Sp3-1.220
plc_L1	Fan	Sp4	1.220			A_SCADA_IF_DM_SET	Set_Val_I	47	A_SCADA_IF_DM_SET.Val_I[47]	504	92	0	plc_L1/Fan/Sp4-1.220
plc_L1	Fan	ActVal	1.220			A_SCADA_IF_DM_ACT	Act_Val_I	16	A_SCADA_IF_DM_ACT.Val_I[16]	501	30	0	plc_L1/Fan/ActVal-1.220
plc_L1	Fan	Status	1.230			A_SCADA_IF_DM_STS	Status_I	10	A_SCADA_IF_DM_STS.Status_I[10]	502	44	0	plc_L1/Fan/Status-1.230
plc_L1	Fan	CmdRev	1.230			A_SCADA_IF_DM_CMD	Cmd	18	A_SCADA_IF_DM_CMD.Cmd[18]	505	2	1	plc_L1/Fan/CmdRev-1.230
plc_L1	Fan	SpRev	1.230			A_SCADA_IF_DM_SET	Set_Val_I	48	A_SCADA_IF_DM_SET.Val_I[48]	504	94	0	plc_L1/Fan/SpRev-1.230
plc_L1	Fan	Sp1	1.230			A_SCADA_IF_DM_SET	Set_Val_I	49	A_SCADA_IF_DM_SET.Val_I[49]	504	96	0	plc_L1/Fan/Sp1-1.230
plc_L1	Fan	Sp2	1.230			A_SCADA_IF_DM_SET	Set_Val_I	50	A_SCADA_IF_DM_SET.Val_I[50]	504	98	0	plc_L1/Fan/Sp2-1.230
plc_L1	Fan	Sp3	1.230			A_SCADA_IF_DM_SET	Set_Val_I	51	A_SCADA_IF_DM_SET.Val_I[51]	504	100	0	plc_L1/Fan/Sp3-1.230
plc_L1	Fan	Sp4	1.230			A_SCADA_IF_DM_SET	Set_Val_I	52	A_SCADA_IF_DM_SET.Val_I[52]	504	102	0	plc_L1/Fan/Sp4-1.230
plc_L1	Fan	ActVal	1.230			A_SCADA_IF_DM_ACT	Act_Val_I	17	A_SCADA_IF_DM_ACT.Val_I[17]	501	32	0	plc_L1/Fan/ActVal-1.230
plc_L1	Fan	Status	1.240			A_SCADA_IF_DM_STS	Status_I	11	A_SCADA_IF_DM_STS.Status_I[11]	502	46	0	plc_L1/Fan/Status-1.240
plc_L1	Fan	CmdRev	1.240			A_SCADA_IF_DM_CMD	Cmd	19	A_SCADA_IF_DM_CMD.Cmd[19]	505	2	2	plc_L1/Fan/CmdRev-1.240
plc_L1	Fan	SpRev	1.240			A_SCADA_IF_DM_SET	Set_Val_I	53	A_SCADA_IF_DM_SET.Val_I[53]	504	104	0	plc_L1/Fan/SpRev-1.240
plc_L1	Fan	Sp1	1.240			A_SCADA_IF_DM_SET	Set_Val_I	54	A_SCADA_IF_DM_SET.Val_I[54]	504	106	0	plc_L1/Fan/Sp1-1.240
plc_L1	Fan	Sp2	1.240			A_SCADA_IF_DM_SET	Set_Val_I	55	A_SCADA_IF_DM_SET.Val_I[55]	504	108	0	plc_L1/Fan/Sp2-1.240
plc_L1	Fan	Sp3	1.240			A_SCADA_IF_DM_SET	Set_Val_I	56	A_SCADA_IF_DM_SET.Val_I[56]	504	110	0	plc_L1/Fan/Sp3-1.240
plc_L1	Fan	Sp4	1.240			A_SCADA_IF_DM_SET	Set_Val_I	57	A_SCADA_IF_DM_SET.Val_I[57]	504	112	0	plc_L1/Fan/Sp4-1.240
plc_L1	Fan	ActVal	1.240			A_SCADA_IF_DM_ACT	Act_Val_I	18	A_SCADA_IF_DM_ACT.Val_I[18]	501	34	0	plc_L1/Fan/ActVal-1.240
plc_L1	Fan	Status	1.250			A_SCADA_IF_DM_STS	Status_I	12	A_SCADA_IF_DM_STS.Status_I[12]	502	48	0	plc_L1/Fan/Status-1.250
plc_L1	Fan	CmdRev	1.250			A_SCADA_IF_DM_CMD	Cmd	20	A_SCADA_IF_DM_CMD.Cmd[20]	505	2	3	plc_L1/Fan/CmdRev-1.250
plc_L1	Fan	SpRev	1.250			A_SCADA_IF_DM_SET	Set_Val_I	58	A_SCADA_IF_DM_SET.Val_I[58]	504	114	0	plc_L1/Fan/SpRev-1.250
plc_L1	Fan	Sp1	1.250			A_SCADA_IF_DM_SET	Set_Val_I	59	A_SCADA_IF_DM_SET.Val_I[59]	504	116	0	plc_L1/Fan/Sp1-1.250
plc_L1	Fan	Sp2	1.250			A_SCADA_IF_DM_SET	Set_Val_I	60	A_SCADA_IF_DM_SET.Val_I[60]	504	118	0	plc_L1/Fan/Sp2-1.250
plc_L1	Fan	Sp3	1.250			A_SCADA_IF_DM_SET	Set_Val_I	61	A_SCADA_IF_DM_SET.Val_I[61]	504	120	0	plc_L1/Fan/Sp3-1.250
plc_L1	Fan	ActVal	1.250			A_SCADA_IF_DM_ACT	Act_Val_I	19	A_SCADA_IF_DM_ACT.Val_I[19]	501	36	0	plc_L1/Fan/ActVal-1.250
plc_L1	Fan	Status	1.260			A_SCADA_IF_DM_STS	Status_I	13	A_SCADA_IF_DM_STS.Status_I[13]	502	50	0	plc_L1/Fan/Status-1.260
plc_L1	Fan	CmdRev	1.260			A_SCADA_IF_DM_CMD	Cmd	21	A_SCADA_IF_DM_CMD.Cmd[21]	505	2	4	plc_L1/Fan/CmdRev-1.260
plc_L1	Fan	SpRev	1.260			A_SCADA_IF_DM_SET	Set_Val_I	62	A_SCADA_IF_DM_SET.Val_I[62]	504	122	0	plc_L1/Fan/SpRev-1.260
plc_L1	Fan	Sp1	1.260			A_SCADA_IF_DM_SET	Set_Val_I	63	A_SCADA_IF_DM_SET.Val_I[63]	504	124	0	plc_L1/Fan/Sp1-1.260
plc_L1	Fan	Sp2	1.260			A_SCADA_IF_DM_SET	Set_Val_I	64	A_SCADA_IF_DM_SET.Val_I[64]	504	126	0	plc_L1/Fan/Sp2-1.260
plc_L1	Fan	Sp3	1.260			A_SCADA_IF_DM_SET	Set_Val_I	65	A_SCADA_IF_DM_SET.Val_I[65]	504	128	0	plc_L1/Fan/Sp3-1.260
plc_L1	Fan	Sp4	1.260			A_SCADA_IF_DM_SET	Set_Val_I	66	A_SCADA_IF_DM_SET.Val_I[66]	504	130	0	plc_L1/Fan/Sp4-1.260
plc_L1	Fan	ActVal	1.260			A_SCADA_IF_DM_ACT	Act_Val_I	20	A_SCADA_IF_DM_ACT.Val_I[20]	501	38	0	plc_L1/Fan/ActVal-1.260
plc_L1	Flap	Status	1.500			A_SCADA_IF_DM_STS	Status_I	14	A_SCADA_IF_DM_STS.Status_I[14]	502	54	0	plc_L1/Flap/Status-1.500
plc_L1	Flap	CmdRev	1.500			A_SCADA_IF_DM_CMD	Cmd	22	A_SCADA_IF_DM_CMD.Cmd[22]	505	2	5	plc_L1/Flap/CmdRev-1.500

Figure 2.5: Remove element number for duplicated objects.

The *element number* of the new element variables will be set by clicking the blue button at the top *Fill element n.* A warning message will appear (Fig. 2.6a). After running the macro behind this button it will be returned the elements that have been created Fig. 2.6b. It will be ready to be imported by ZenOn.

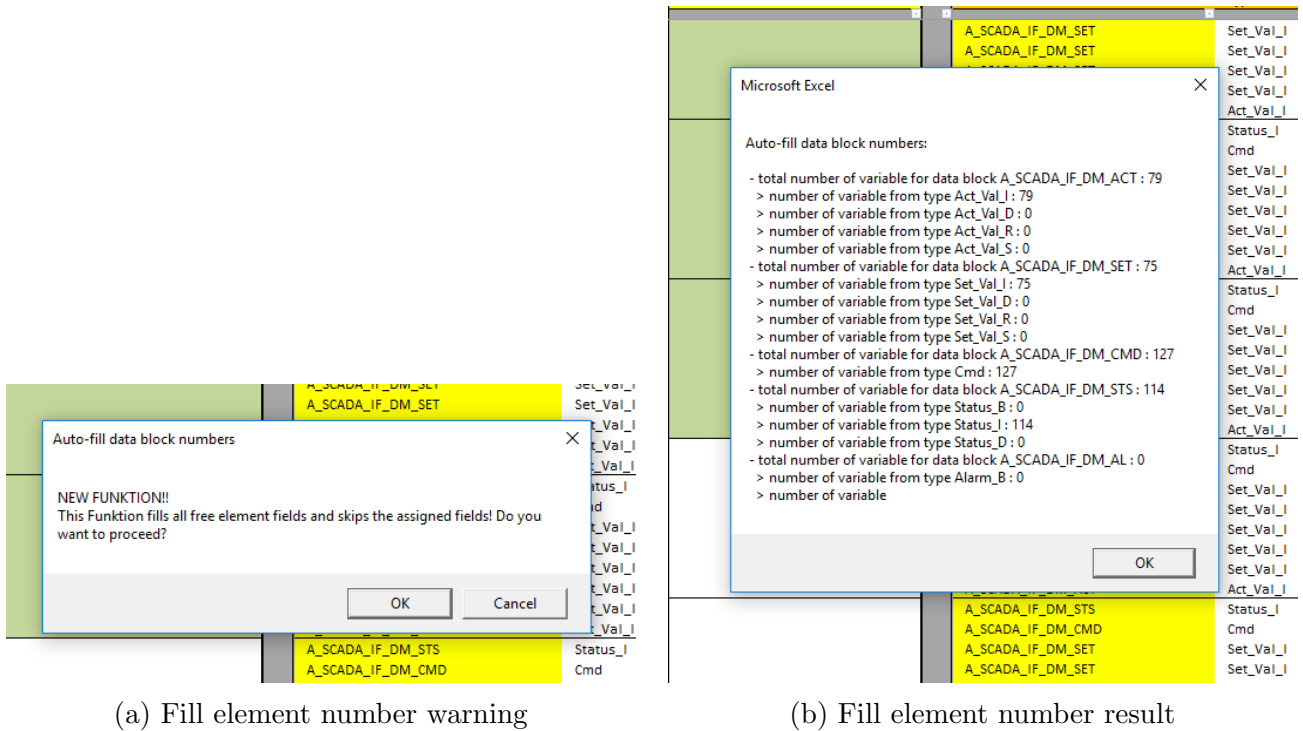


Figure 2.6: Fill element number

After getting the new element numbers it is possible to get the addresses for the variables related to the new element by clicking the button *Fill address* located just above the *PLC interface data block* header. Similarly to the previous case images for the warning and the result are shown in Fig. 2.7

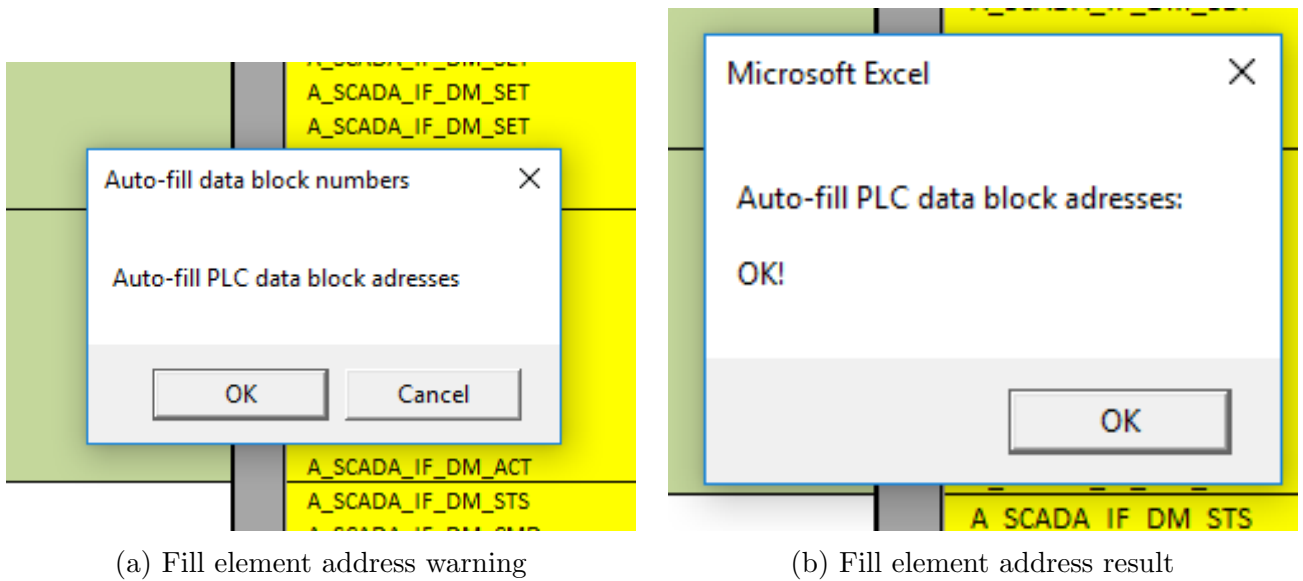


Figure 2.7: Fill element address

Last step before interacting with ZenOn will be to define which of the interface list's elements are going to be uploaded / updated in the current variable list in ZenOn. Take into account that the process of overriding a variable previously defined is not a problem for the future use of the application but a waste of time in case the variables were already there.

To avoid that it is recommended to mark only with an  $x$  the variables to upload at the column *Export?* (Fig. 2.8)

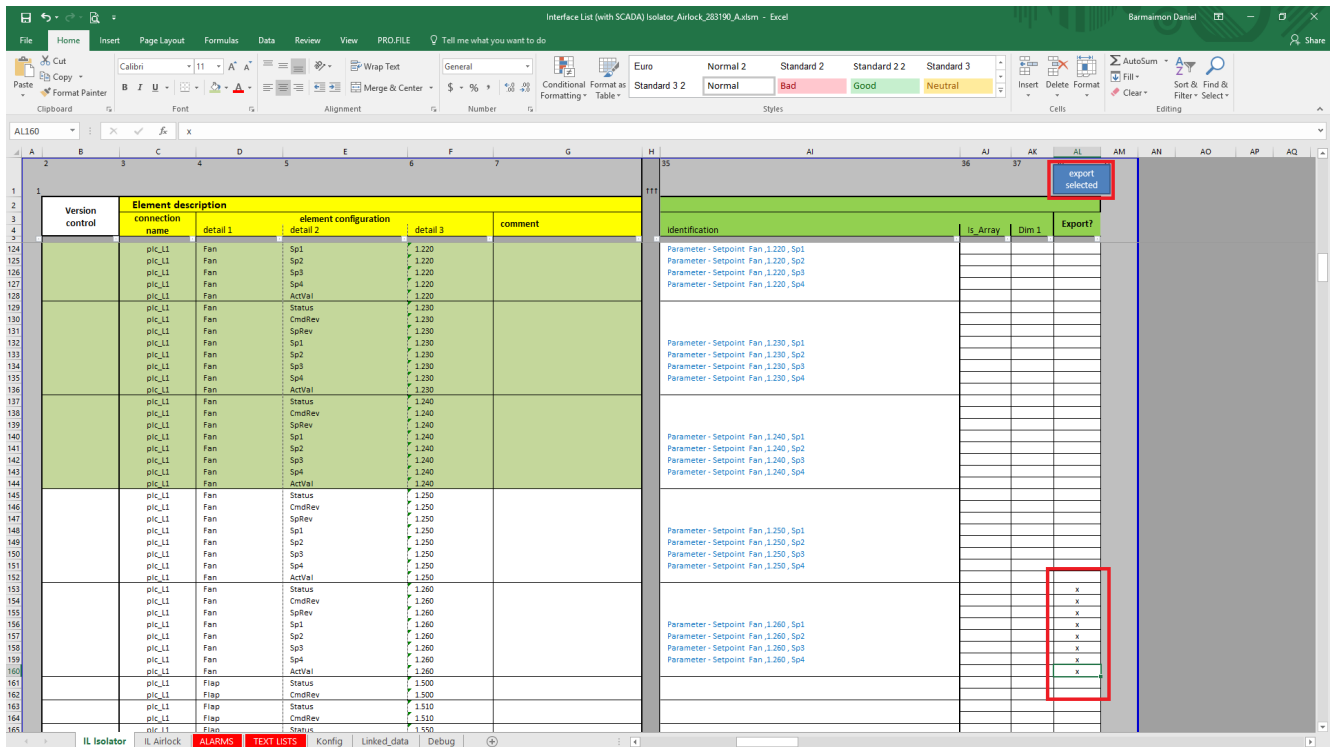


Figure 2.8: Selection of the variables to create/update in ZenOn

After clicking over *Export selected* and choosing the destination folder a *.txt* file will be generated with a script that will be used by ZenOn to create/update the variables that were chosen.

## 2.4 Creation of the variables with the interface list result

To create/update the variables after having finished the previous section's process it will be needed to initialize ZenOn and open the VBA editor.

There are two scripts that will be important in the process. In *EDITOR\_scripts\_Variables* it is mandatory to copy the content of the *.txt* file that was generated by the *interface list* document. (Fig. 2.9)

After copying the *.txt* file remember to save the script.

The second script is the one that defines the drivers and call the first one. This script **should not be modified** (Fig. 2.10).

To generate/update the variables in ZenOn run this script.

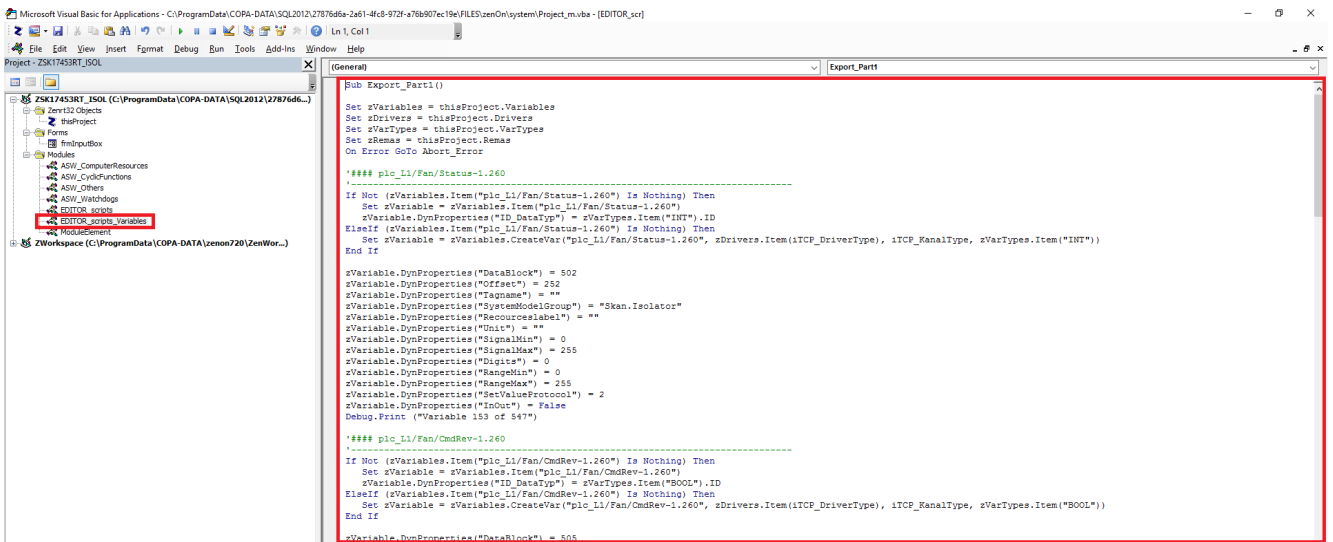


Figure 2.9: Configuration of the variables that will be generated

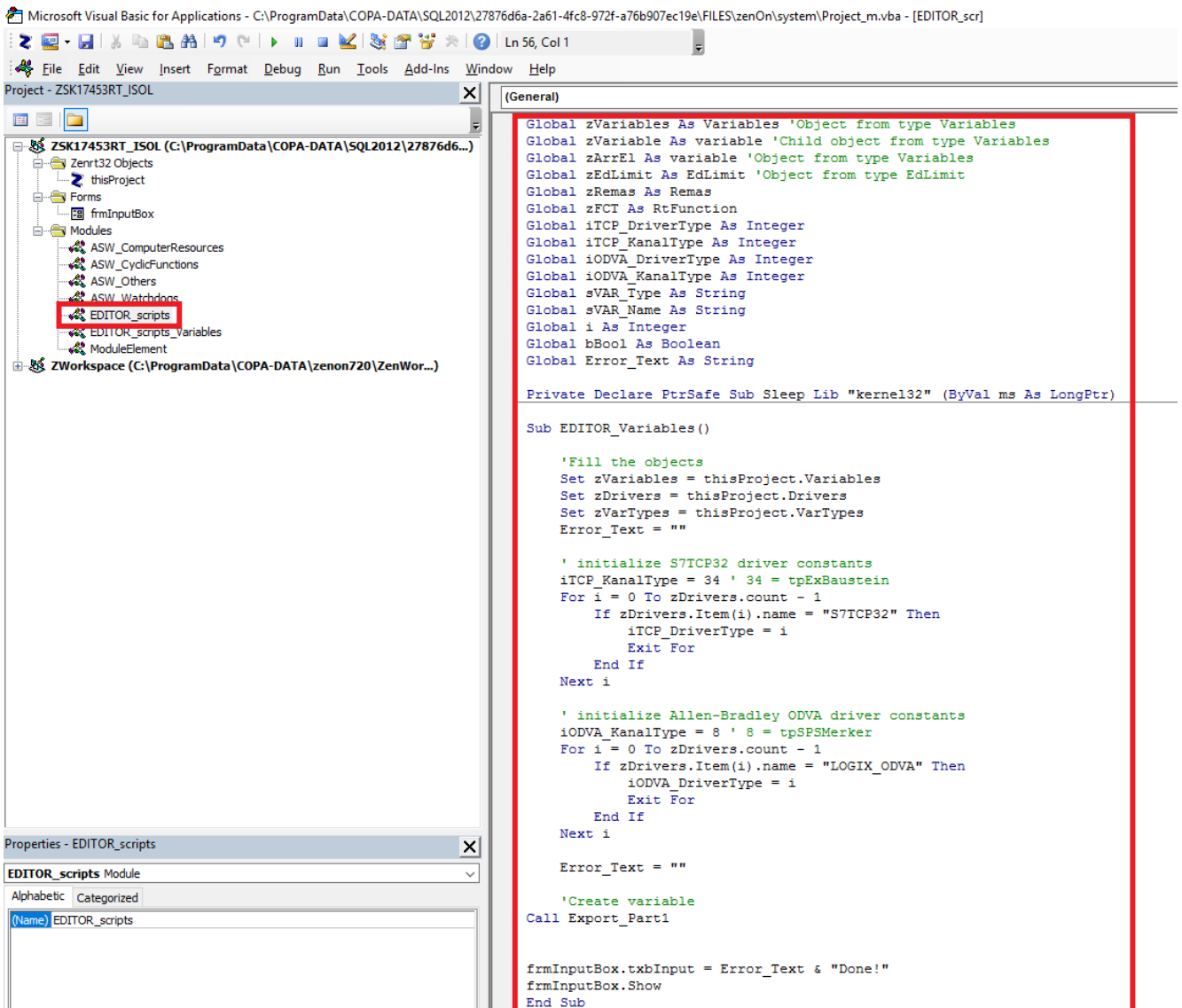


Figure 2.10: Setup the drivers and call the first script

# Chapter 3

## Customized elements

Some of the elements that are used in the projects are behaving similarly in several of them. To avoid repeating the same work in each of the projects it makes sense to create a library with these elements.

Each of the elements should have the following attributes properly defined before adding it to the library:

- **Representation** Drawing, color and flashing behaviour for each possible estate.
- **Reaction matrix** Could be the case that this RM is used for more than one element.

# Chapter 4

## Methods and Tools

The main purpose of this section is to collect all possible methods we are using for different purposes. In case an issue could be solved by several ways it makes sense to explain which one we are using and its pros and cons with respect to the others.

### 4.1 Visibility of an element

#### 4.1.1 Variable link

The visibility of an element could be linked to the value of a variable. It is possible to choose the range of values for which the element will be visible.

An example of how to configure the visibility of an element (i.e.: button) using a variable is given in the Fig. 4.1. In this case only super users (with user level equal to 9) will be able to see the element.

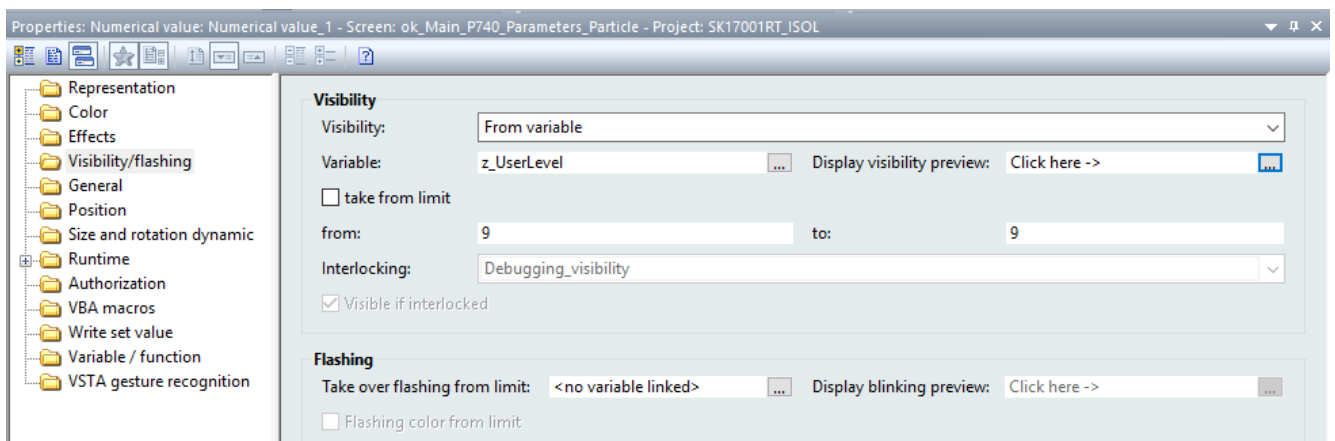


Figure 4.1: Linking element's visibility to a variable

## 4.1.2 Interlock

Interlocking allows to control the access to certain variables given values and combination of another. The use of interlock variables for controlling the visibility is very practical.

The first thing to do will be to create an interlock. In order to do this just right-click the 'Interlockings' button in the toolbar and then select 'New Interlocking'.

After giving a proper name to the interlock you can add variables and conditions that will be involved in the activation of the interlock. It makes sense that more than one variable or more than one condition occurs. Otherwise it will be easier to link the visibility of an object directly to a variable.

A menu as the following one in Fig. 4.2 will appear allowing to select the variables and the conditions to activate it.

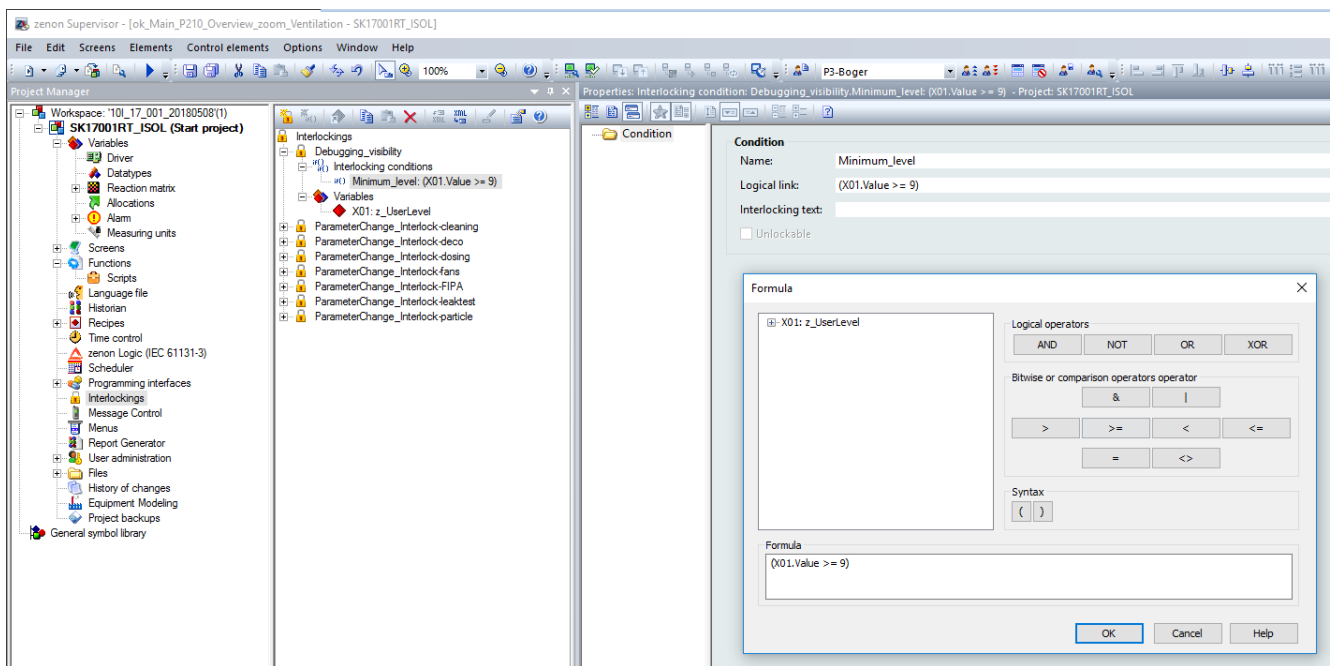


Figure 4.2: How to create an interlock with variables and conditions

In this example we choose as variable *user level* and a condition to be greater or equal to 9. This will allow only to super-users (once that have logged-in) to see the elements linked to the interlock.

To link an element to the visibility using the interlock, look at the example at the Fig. 4.3.

As it was mention above this way is more complicated than linking only to a variable. This was only an example but it will be important to allow the visibility of elements when more than one condition should be happening simultaneously.

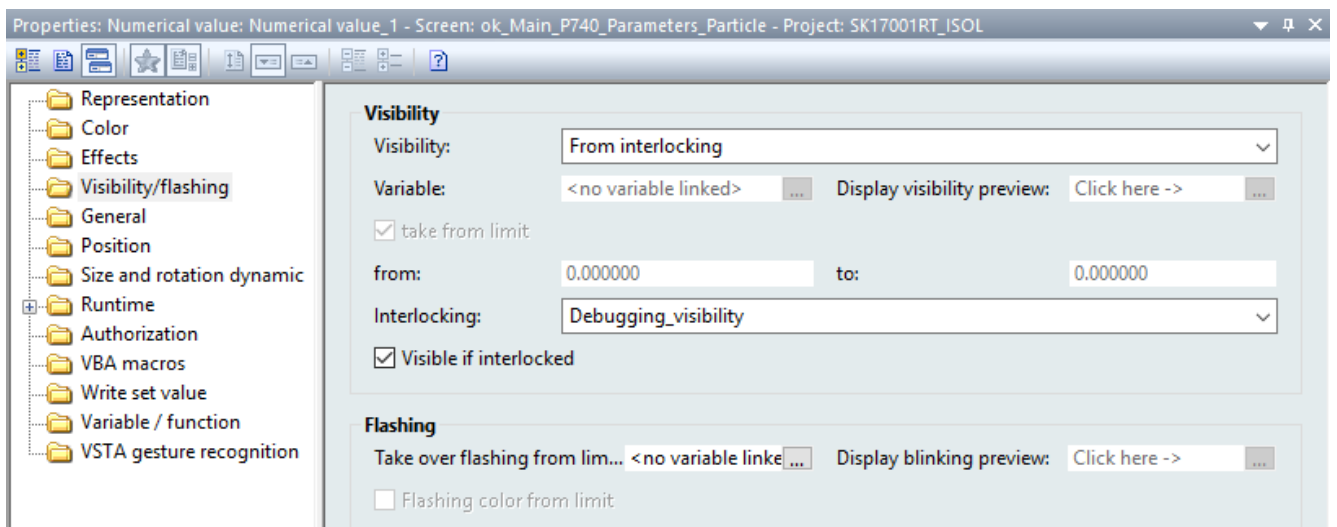


Figure 4.3: Linking a dynamic text box to an interlock



# Chapter 5

## Customized screens

### 5.1 Screen numbering

There is a standard established to set the names of each of the screens. The use of 3 digits to identify the screen was set. In Fig. 5.1 is described the way in which the digits are given.

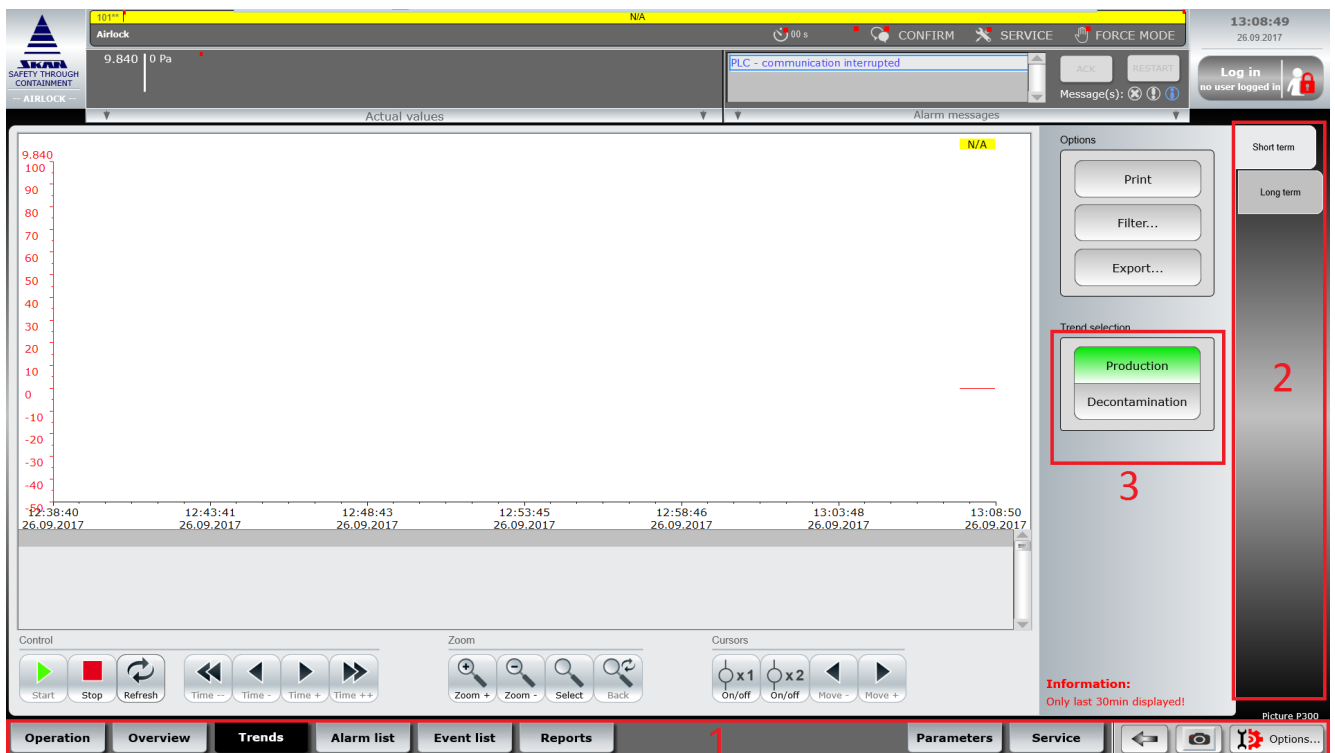


Figure 5.1: Screen numbering process

First digit is given by the position of the position of the selected button in the horizontal navigation panel located at the bottom of the screen. In the example we can see that the screen should start with the number 3 as 'Trends' button is located at the third position. So far we have the following number for the current page: **3XX**

For the second level in the numbering hierarchy the vertical navigation menu should be checked.

In this case it is located at the right side of the screen. It is important to remark that we will start the numbering of the second digit with 0 (at the first level we were starting at 1). Now we can update the page number: **30X**

If there is any other level of information that could lead to different pages for the already given menu we should repeat the process for the second digit. Finally we can get the final screen number for the example: **300**

## 5.2 Tips and tricks

In order to ease the development we will use some rules. The main ones are related with how to set the different elements in specific layers, using scripts to set those elements in the proper positions and how to handle their visibility.

### 5.2.1 Switch screen with substitution parameter

There are cases in which two or more screens are exactly the same but with different parameters. To design the screens in those cases there are two main ways to do it:

1. Create two or more screens with the specific parameters (variables, functions, etc.).
2. Use only one screen and multiple functions to call to it with parameter substitution.

Second option allows to have less screens and saves development time. For example, if a button is moved we will not need to move it in several screens as we only have one.

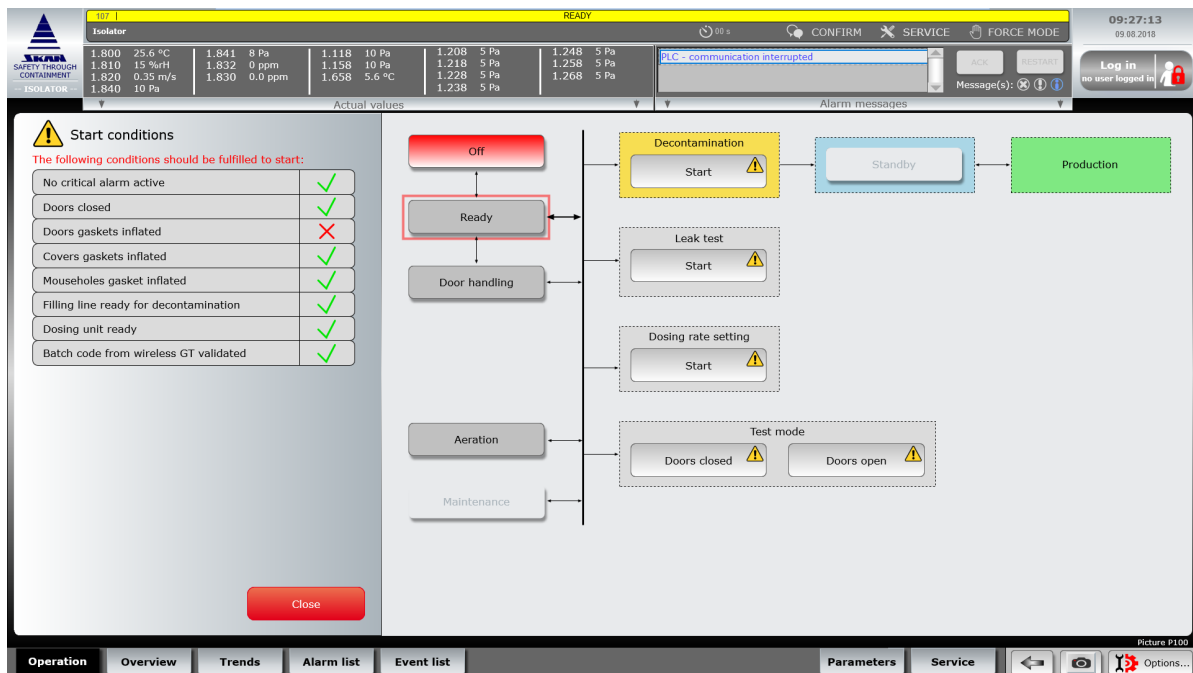
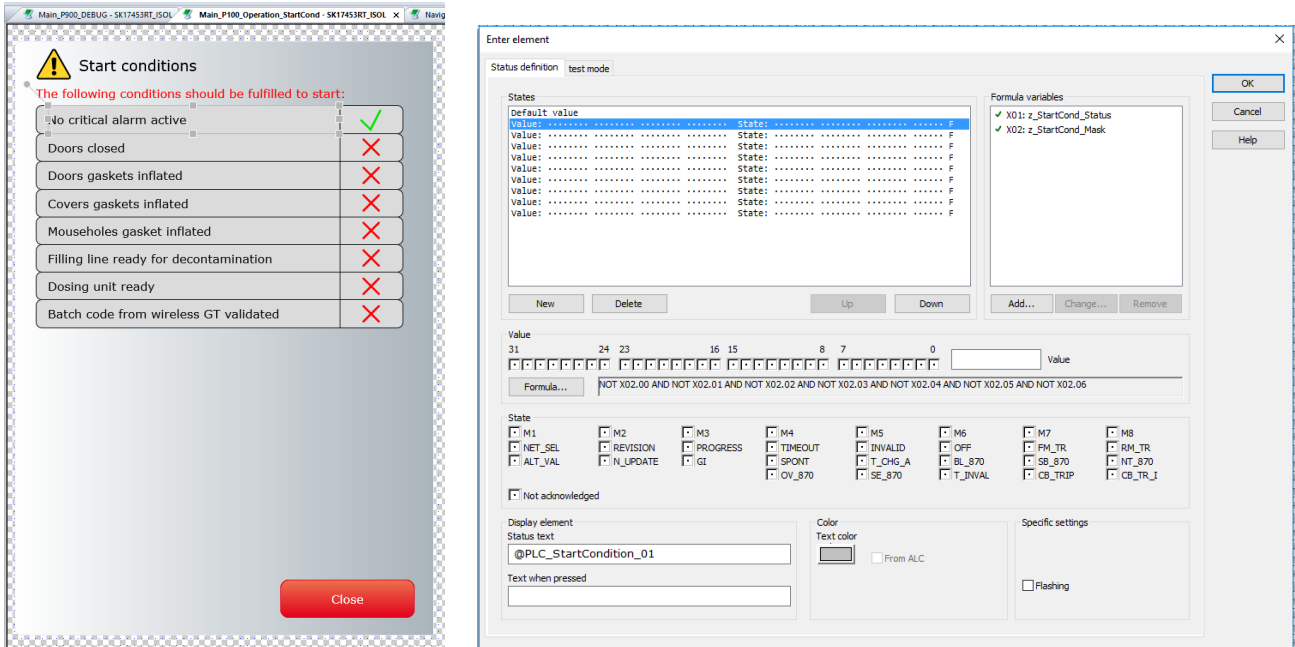


Figure 5.2: General view of operation screen

To visualize the use of this feature it is possible to take as an example the *Start conditions* screen. In Fig. 5.2 several buttons with an exclamation sign are shown. Pressing any of this buttons will call the same screen with different parameters.

In Fig. 5.3a we can check how looks the standard screen, and in Fig. 5.3b the configuration menu for a combi element representation is shown. In the case of the combi element, two variables are responsible to represent different estates. These variables will be substituted later on.



(a) Start conditions standard screen

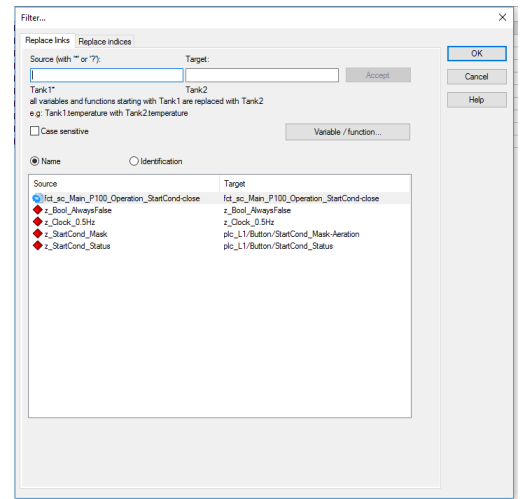
(b) Configuration of combi element representation

Figure 5.3: Example of screen configuration for substitution

Each of the buttons marked with the exclamation sign has a function linked to it. As shown in Fig. 5.4a there will be as many functions as buttons. The configuration of the possible substitution of the dynamic elements in the screen are given by a filter, Fig. 5.4b.

State	Name	Type	Parameter
	Filter text		
	ftc_sc_Main_P100_Operation_StartCond_Aeration	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_Deco-start	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_DosingRate-start	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_Leaktest-start	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_Production	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_Ready	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_Standby	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_TestModeClosed	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond_TestModeOpen	Screen switch	Main_P100_Operation_StartCond (Standard)--SUBST
	ftc_sc_Main_P100_Operation_StartCond-close	Close frame	[(F_ MAIN_StartCond)]

(a) Functions for *start conditions*



(b) Configuration of combi element representation

Figure 5.4: Functions for screen switching and substitution parameters

The only detail to take care of is the correct definition of each of the variables for *mask* and *status*.

## 5.3 Service Screen

### 5.3.1 P800 - SKAN AG info

The service screen also provides information about SKAN to the customer. Most important details namely address, telephone, fax and email are clearly visible as shown in Fig. 5.5



Figure 5.5: User's view of the service screen

To make it a little more attractive and keep the user hooked we attach several images that will

be shown sequentially. The idea is to allow the visibility of each one only for some seconds using a VBA script. An integer variable increases its value from 0 to 60 each time that the script is called, as shown in Fig. 5.6.

```

'----- SKAN SCADA SYSTEM -----'
'
' SCRIPT NAME:      fct_ASW_Watchdogs
' PURPOSE:         Managed watchdog signals from/to PLC
'
' VERSION:         1.0
' VERSION DATE:    03.08.2016
' AUTHOR:          wipfeml
'
'----- CHANGES -----'
' CHANGES: [VERSION] [DATE] [CHANGED BY] [PURPOSE]
'           1.0       03.08.2016 wipfeml   First issue
'-----'

Sub fct_ASW_Watchdogs ()

    Dim iCnt As Integer
    On Error Resume Next

    ' Update HMI watchdogs
    '-----'

    iCnt = thisProject.Variables.Item("plc_L1/Hmi/Watchdog-toPLC").Value
    If iCnt < 60 Then
        iCnt = iCnt + 1
    Else
        iCnt = 0
    End If
    thisProject.Variables.Item("plc_L1/Hmi/Watchdog-toPLC").Value = iCnt
    thisProject.Variables.Item("z_Watchdog_PLC_old").Value = thisProject.Variables.Item("plc_L1/Hmi/Watchdog-fromPLC").Value

End Sub

```

Figure 5.6: Script to provide a time counter

To be sure that the script is running at each second it is needed to have a function (to trigger the call, Fig. 5.7) and a time controller (to run the function at each round second, Fig. 5.8).

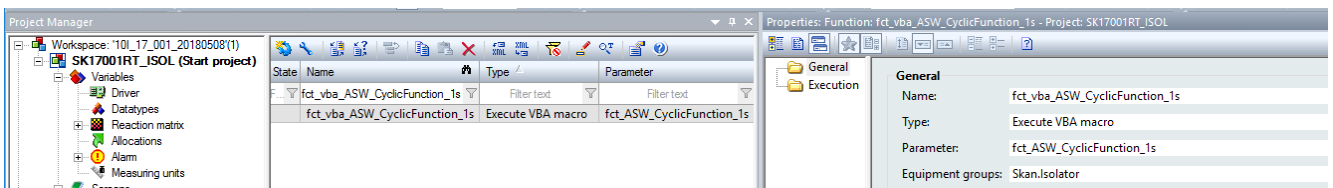


Figure 5.7: VBA trigger function

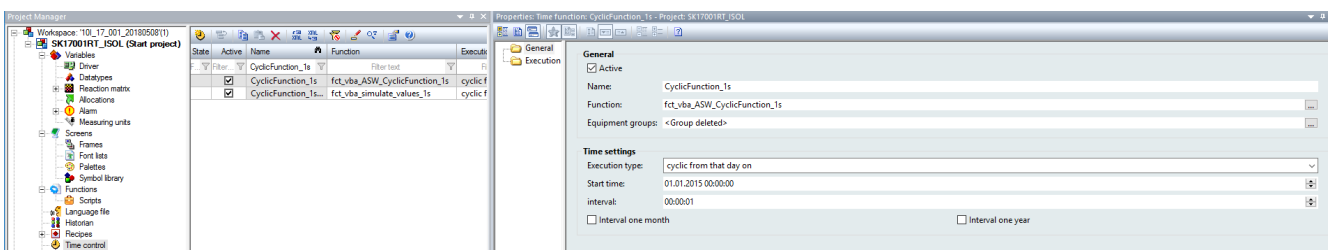


Figure 5.8: Time control function

### 5.3.2 P810 - Service functions

Service functions' screen is providing several functionalities and information. At the moment it has four different sections, as shown in 5.9

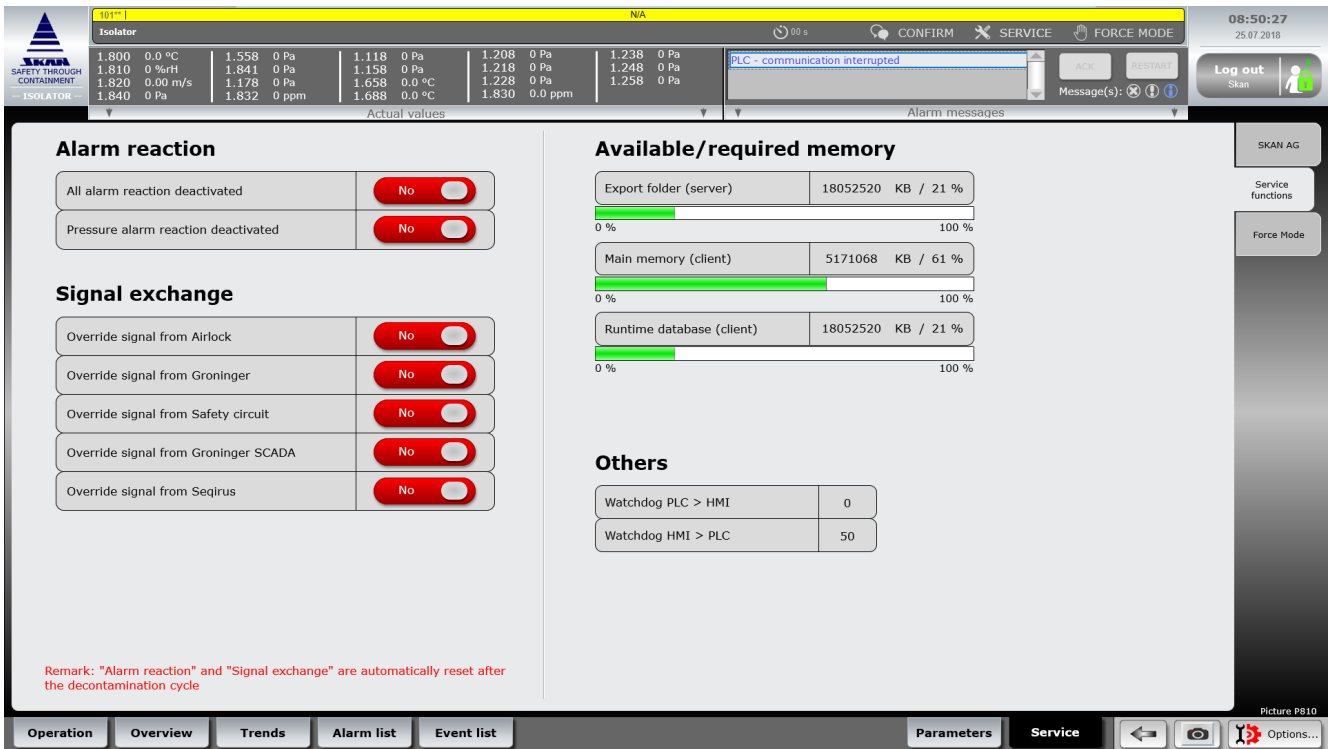


Figure 5.9: Service functions' screen

## Alarm reaction

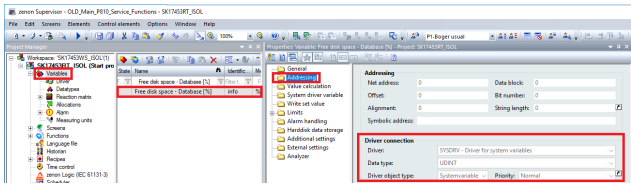
Allow the deactivation of the whole block of alarms or the pressure related alarms by sending a command to the PLC. This is very handy for cases in which specific functions should be tested, a sensor / valve should be changed, etc.

## Signal exchange

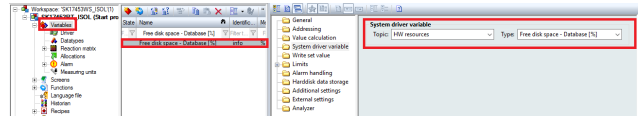
Similarly it is possible to override signals that are coming from third party devices. In testing cases these signals can limit the way isolator is behaving. It is important for installation, testing and substitution of broken / defect parts.

## Available / required memory

It is an informative part of the screen. User should know if the storing memory is enough to handle each part of the data required by ZenOn, reports, trends, chronological event lists, runtime data, etc. The default way to perform it is to create driver variables as shown in Fig. 5.10



(a) Creation of driver variables



(b) Configuration of driver variables

Figure 5.10: Obtaining computer resources' info

The problem with this kind of variables is that are not updated automatically if the SCADA is running. A script with a cyclic call should be created to that aim, Fig.5.11

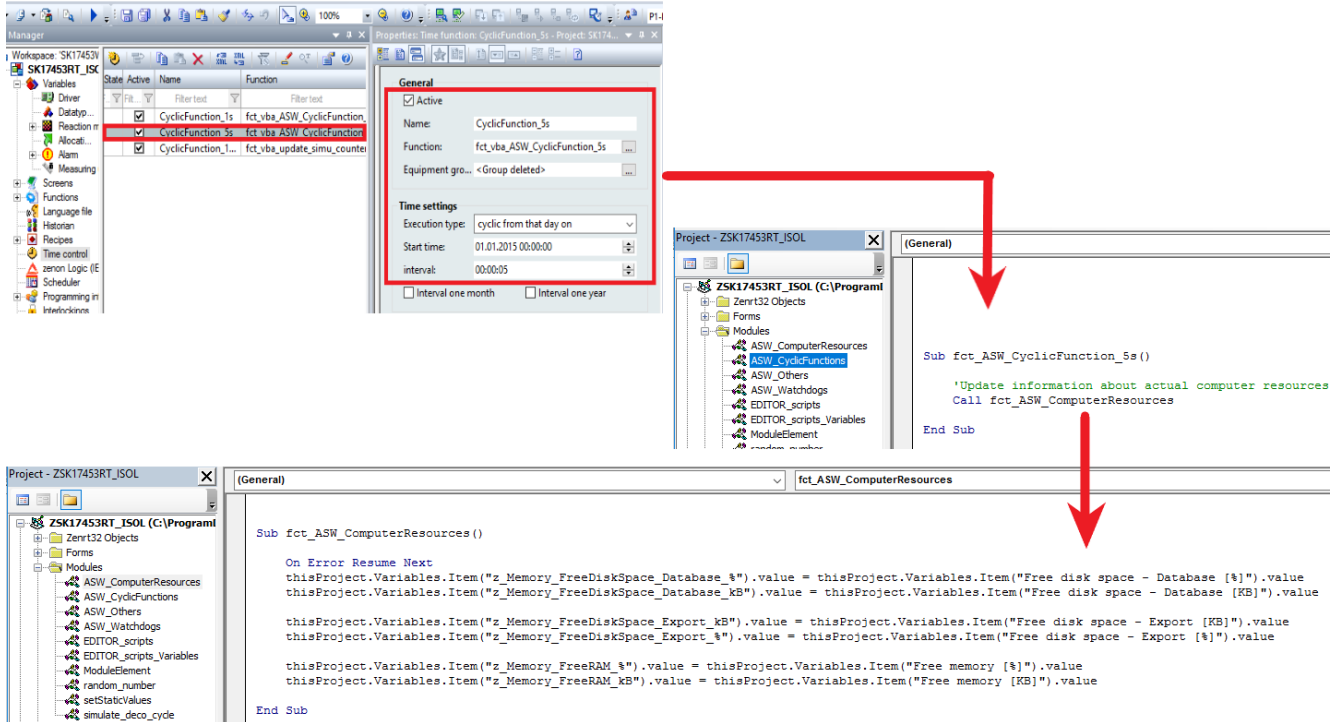


Figure 5.11: Cyclic call to update the resources' info

## Watchdog

It is important to know when the communication between PLC and SCADA system has been lost for more than a certain amount of time. The system should launch an alarm and users should not be able to change into any mode at this point.

There are several versions of the watchdog. They will be sorted chronologically, meaning that the current one will be explained first and the rest will be explained after. The reasons for not using the former versions will also be described.

### Method 1

### Method 2

Three different variables are used in ZenOn to check this communication link. Fig. 5.12

z_Watchdog_PLC_old
plc_L1/Hmi/Watchdog-fromPLC
plc_L1/Hmi/Watchdog-toPLC

Figure 5.12: Variables needed for watchdog implementation

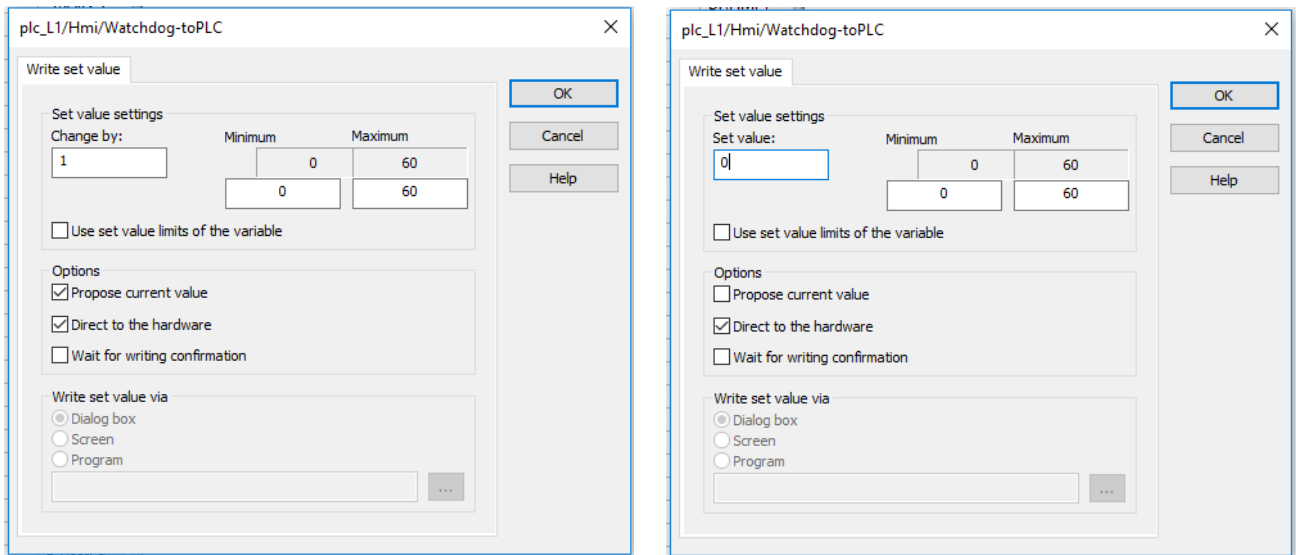
The internal variable is used to trigger an alarm if there is no change after 15 seconds in the value received by PLC variable. It is possible through the variable's configuration with a dynamic limit given and a delay. Check the details in Fig. 5.13

The screenshot shows the configuration for an internal variable. The 'Limits' section at the top has links for '{Limit new: Click here ->}' and 'Limit preview: Click here ->'. Below is the 'Reaction matrix' section with a dropdown set to '<no reaction matrix linked>'. The 'Limits[1]' section contains several fields: 'Limit active' is checked; 'Limit text' is '@HMI\_FAULT\_MESSAGE\_Watchdog\_PLC'; 'Limit' is 0; 'Minimum/Maximum' is set to 'Maximum'; 'Threshold value' is 0.000000; 'Dynamic limit active' is checked; 'Delay time [s]' is 15; and 'Variable' is 'plc\_L1/Hmi/Watchdog-fromPLC'. The 'AML/CEL[1]' section has 'In Alarm Message List' checked, and 'Alarm/event class' is set to '13 - INFORMATION'. Other options like 'In Chronological Event List', 'To acknowledge', 'To delete', 'Comment required', and 'Print' are unchecked.

Figure 5.13: Configuration of the internal variable that triggers watchdog alarm

Two functions will be needed to increase and reset the variable *plc\_L1/Hmi/Watchdog-toPLC*, Fig. 5.14





(a) Increment of the variable

(b) Reset of the variable

Figure 5.14: Functions to increment and reset a variable

The values received from the PLC are stored into an allocation variable that is triggered by any change in the current value.

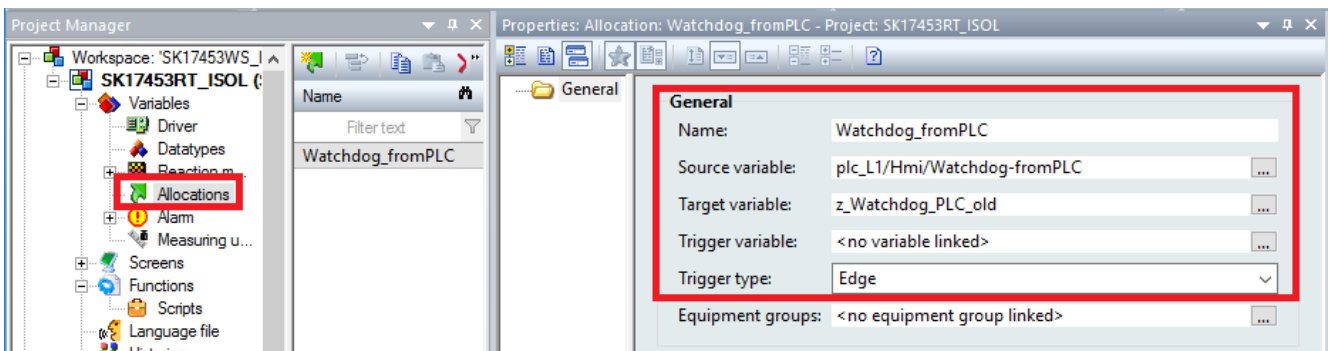


Figure 5.15: Configuration of PLC variable's allocation

The main reason to stop using this method is that some problems were found when stopping and starting the runtime. A solution for this issue is to call the function to increase the value for *plc\_L1/Hmi/Watchdog-toPLC* in the ZenOn script that is running at start-up. This should be tested to ensure that it solves the problem.

### Method 3

This implementation was done using a cyclic call to a VBA script. When the cyclic call period is too short for the SCADA system (around 2 seconds), the SCADA system could freeze or malfunction. Just for better understanding of how the whole concept was developed check Fig. 5.16

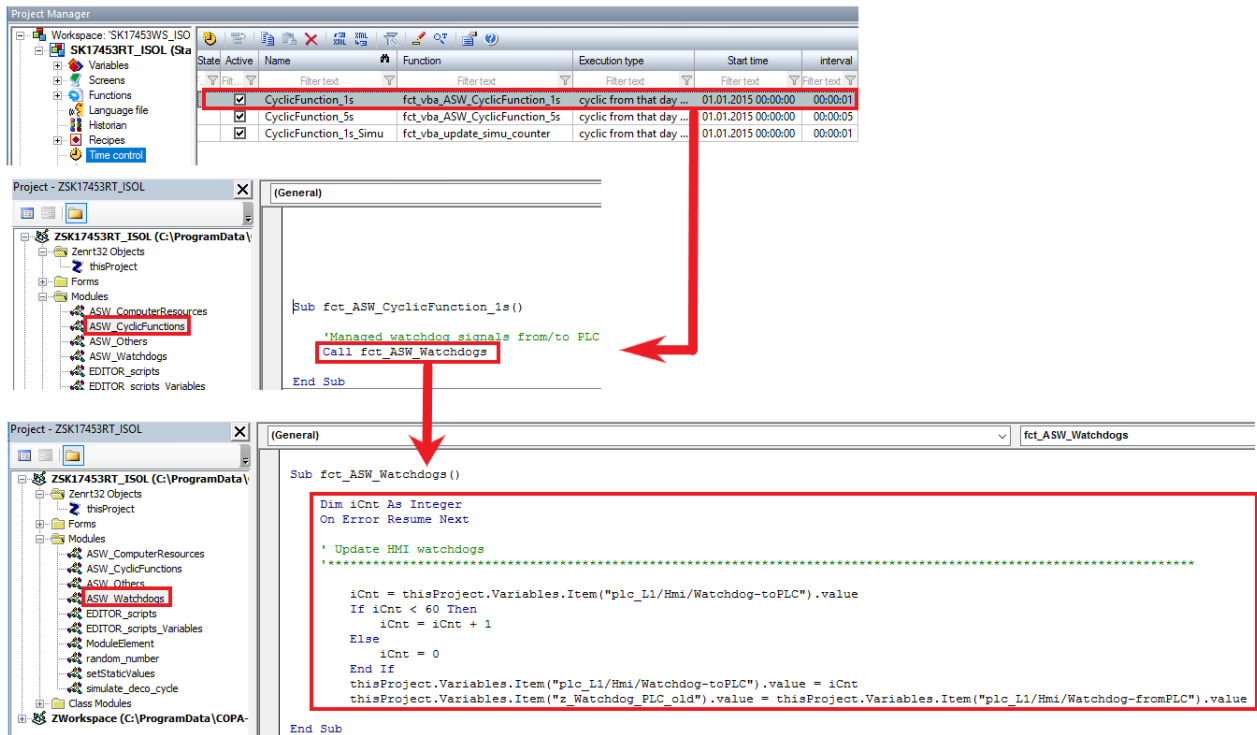


Figure 5.16: Old implementation of watchdog's *toPLC* variable

### 5.3.3 P820 - Force mode

# Chapter 6

## Trends

# Chapter 7

## Alarms

### 7.1 Alarm Matrix - ProFile Document

The list with alarms is included in the document *AM - Alarm Matrix*. There should be a document for each of the specific areas of the isolator i.e. *isolator, airlock, etc.*. This could be found in ProFile as shown in Fig. 7.1

Vorschau	Ve Sp	Proj-Nr	Profile D-ID	SKAN D-ID	Dok Titel	Ver Re	Geändert	Geändert von	WF Status	WF am	WF von	Dokumentart	Dokumenttyp	Erstellt	Erstellt von	
		10A-17-453	564648	283119	AM - Alarm Matrix Isolator	A	01	19.07.2018	jeanni1	Entwurf	19.07.2018	jeanni1	Spezifikation	AM - Alarmmatrix	24.05.2018	jeanni1
		10A-17-453	550590	279494	Auslegung Lecktest	A	01	13.07.2018	bauema1	In Bearbeitung			Spezifikation	Designspezifikation	19.03.2018	bechma2
		10A-17-453	550592	279496	Auslegung Umluft	A	01	12.07.2018	bauema1	In Bearbeitung			Spezifikation	Designspezifikation	19.03.2018	bechma2
		10A-17-453	550593	279497	Auslegung Zur-Abbluft	A	01	10.07.2018	bauema1	In Bearbeitung			Spezifikation	Designspezifikation	19.03.2018	bechma2
		10A-17-453	564737	283158	Automation concept	A	01	12.07.2018	jeanni1	Entwurf	12.07.2018	jeanni1	Zeichnung	R&I - Schema	24.05.2018	jeanni1

Figure 7.1: Alarm list document in ProFile

Once the document is locally downloaded it is possible to use it to directly import the messages, reaction matrices and level of each alarm. In order to do that a macro specially created should be loaded in Excel.

### 7.2 List manipulation with Excel

*Developer* tab must be visible to interact with macros. The way to activate it is to select *Developer* in the *Customize the Ribbon* option as shown in Fig. 7.2

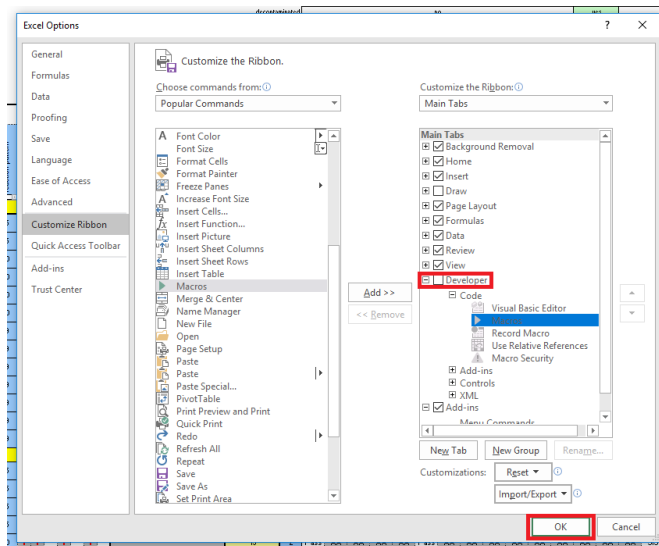


Figure 7.2: Show *Developer* tab in Excel

The macro is located in *P:\4. Copa-Data zenOn (HMI-SCADA Software)\8. Vorlagen\Alarm\_liste*. To import it a password is required "Dobs" during the process, Fig. 7.3

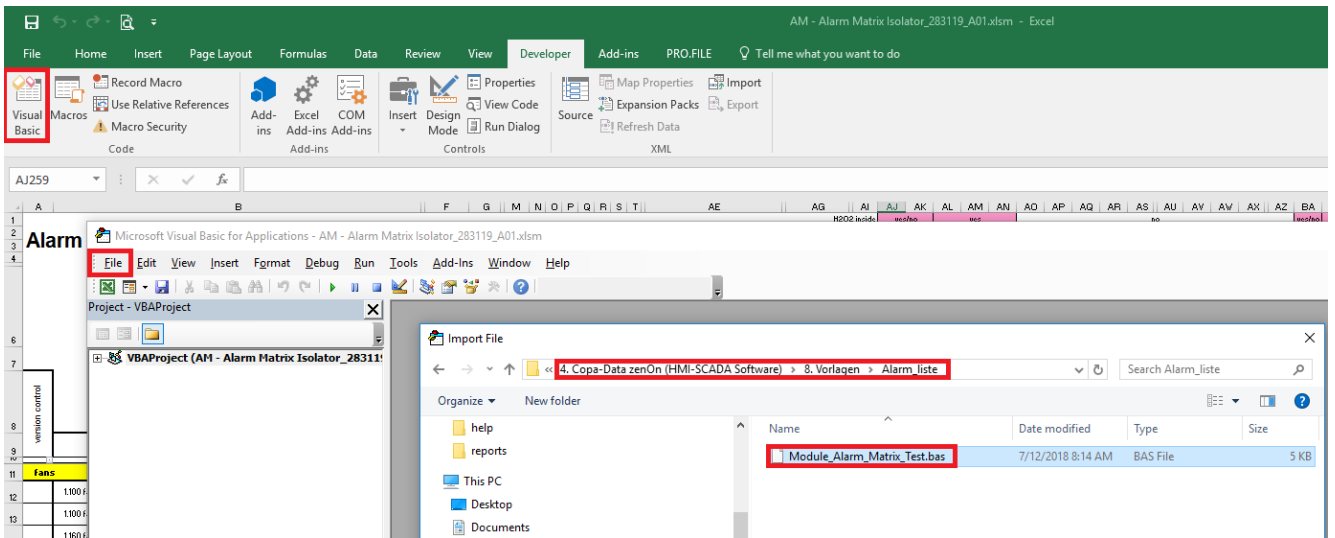


Figure 7.3: Import the macro from the automation folder

At this point it is possible to create the list of alarms using the added macro, Fig. 7.4.

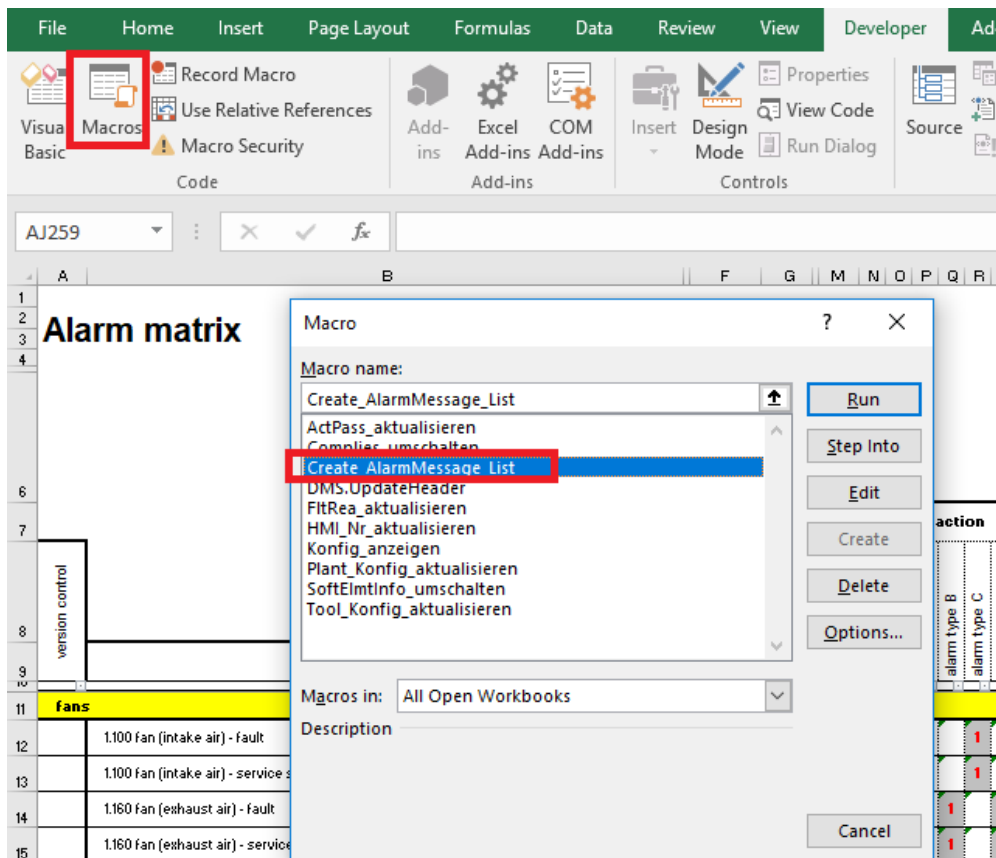
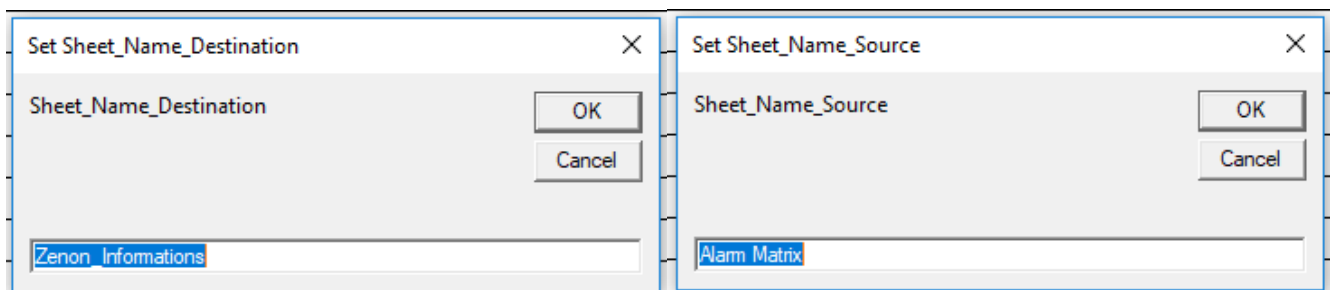


Figure 7.4: Start the alarm list's creation using the macro

The process will guide the user and will ask about the output and source sheets for the list, Fig. 7.5



(a) Destination sheet for the alarm list

(b) Source sheet for the alarm list

Figure 7.5: User input for importing alarm messages using the dedicated macro

It will be also necessary to specify the column where the alarm number (for the SCADA system) is located as shown in Fig. 7.6 . Usually is *F* but is important to set it properly, specially if the document structure is modified.

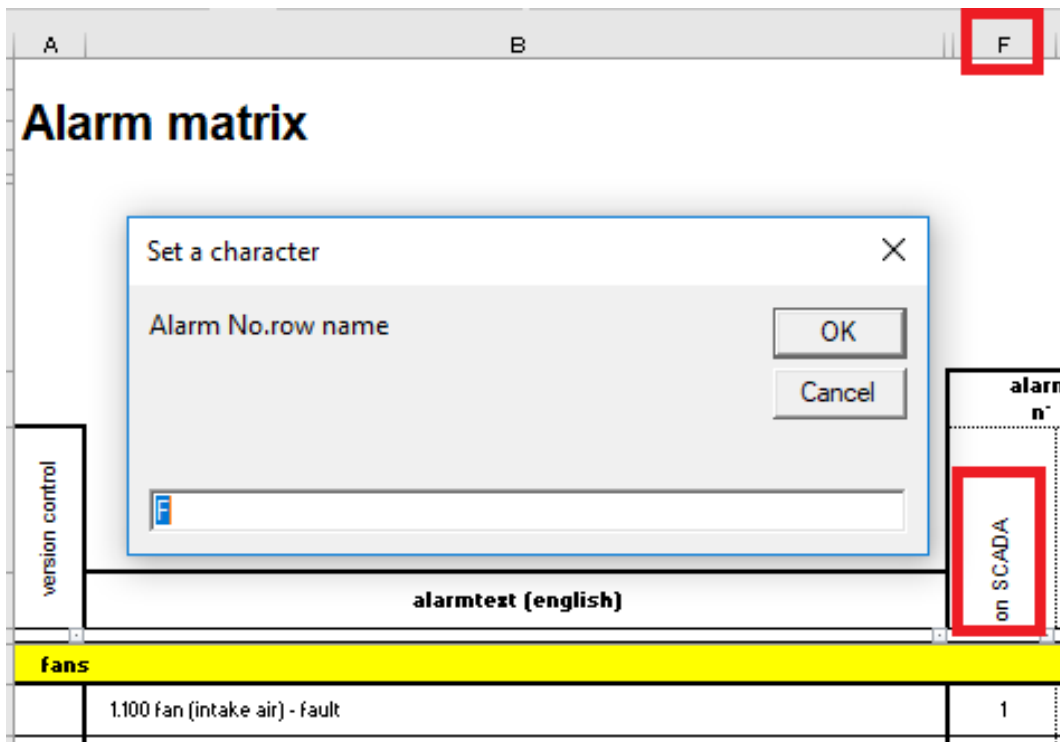
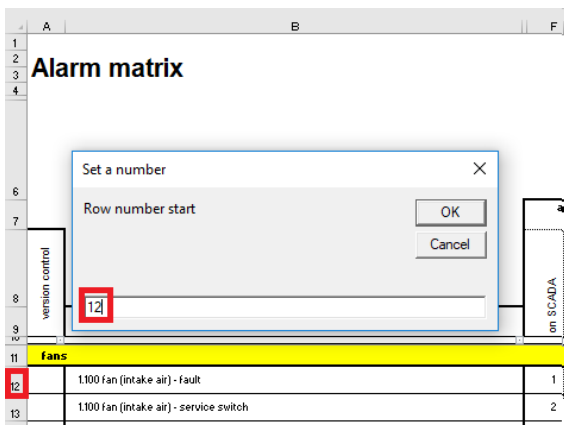
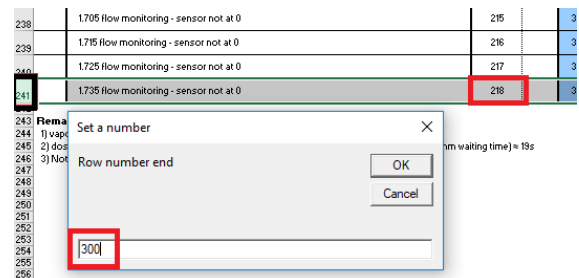


Figure 7.6: Selection of the column for alarm number

Starting and ending rows should also be defined. It is a good practice to set 50 to 80 more rows in case there are some spare alarms expected for the future, Fig. 7.7



(a) Starting row for the alarms



(b) Ending row for the alarms

Figure 7.7: User input for row selection in the dedicated macro

The resulting sheet should look like in Fig. 7.8

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	PLC_FAULT_MESSAGE_0001		1.100 fan (intake air) - fault						Faults:	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17				
2	PLC_FAULT_MESSAGE_0002		1.100 fan (intake air) - service switch						Warnings:	61, 66, 76, 83, 84, 87, 88, 91, 92, 97, 98, 101, 102,				
3	PLC_FAULT_MESSAGE_0003		1.160 fan (exhaust air) - fault						All:	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,				
4	PLC_FAULT_MESSAGE_0004		1.160 fan (exhaust air) - service switch											
5	PLC_FAULT_MESSAGE_0005		1.150 fan (exhaust regulation air) - fault											
6	PLC_FAULT_MESSAGE_0006		1.150 fan (exhaust regulation air) - service switch											
7	PLC_FAULT_MESSAGE_0007		1.200 fan (circulation air) - fault											
8	PLC_FAULT_MESSAGE_0008		1.210 fan (circulation air) - fault											
9	PLC_FAULT_MESSAGE_0009		1.220 fan (circulation air) - fault											
10	PLC_FAULT_MESSAGE_0010		1.230 fan (circulation air) - fault											
11	PLC_FAULT_MESSAGE_0011		1.240 fan (circulation air) - fault											
12	PLC_FAULT_MESSAGE_0012		1.250 fan (circulation air) - fault											
13	PLC_FAULT_MESSAGE_0013		1.260 fan (circulation air) - fault											
14	PLC_FAULT_MESSAGE_0014		1.500 flap (intake air) - not open											
15	PLC_FAULT_MESSAGE_0015		1.500 flap (intake air) - not closed											
16	PLC_FAULT_MESSAGE_0016		1.510 flap (intake air) - not open											
17	PLC_FAULT_MESSAGE_0017		1.510 flap (intake air) - not closed											
18	PLC_FAULT_MESSAGE_0018		1.550 flap (exhaust air) - not open											
19	PLC_FAULT_MESSAGE_0019		1.550 flap (exhaust air) - not closed											
20	PLC_FAULT_MESSAGE_0020		1.560 flap (exhaust air) - not open											
21	PLC_FAULT_MESSAGE_0021		1.560 flap (exhaust air) - not closed											
22	PLC_FAULT_MESSAGE_0022		1.831 valve (H2O2 LC sensor) - measuring position											
23	PLC_FAULT_MESSAGE_0023		1.831 valve (H2O2 LC sensor) - inactive position											
24	PLC_FAULT_MESSAGE_0024		1.305 valve (particle counter) - deco loop position											
25	PLC_FAULT_MESSAGE_0025		1.305 valve (particle counter) - measuring position											
26	PLC_FAULT_MESSAGE_0026		1.315 valve (particle counter) - deco loop position											
27	PLC_FAULT_MESSAGE_0027		1.315 valve (particle counter) - measuring position											
28	PLC_FAULT_MESSAGE_0028		1.325 valve (particle counter) - deco loop position											
29	PLC_FAULT_MESSAGE_0029		1.325 valve (particle counter) - measuring position											
30	PLC_FAULT_MESSAGE_0030		1.335 valve (particle counter) - deco loop position											
31	PLC_FAULT_MESSAGE_0031		1.335 valve (particle counter) - measuring position											
32	PLC_FAULT_MESSAGE_0032		1.355 valve (air sampler) - deco loop position											
33	PLC_FAULT_MESSAGE_0033		1.355 valve (air sampler) - measuring position											
34	PLC_FAULT_MESSAGE_0034		1.365 valve (air sampler) - deco loop position											
35	PLC_FAULT_MESSAGE_0035		1.365 valve (air sampler) - measuring position											
36	PLC_FAULT_MESSAGE_0036		1.375 valve (air sampler) - deco loop position											
37	PLC_FAULT_MESSAGE_0037		1.375 valve (air sampler) - measuring position											
38	PLC_FAULT_MESSAGE_0038		1.385 valve (air sampler) - deco loop position											
39	PLC_FAULT_MESSAGE_0039		1.385 valve (air sampler) - measuring position											
40	PLC_FAULT_MESSAGE_0040		1.308 vacuum pump (particle counter) - no flow											
41	PLC_FAULT_MESSAGE_0041		1.318 vacuum pump (particle counter) - no flow											
42	PLC_FAULT_MESSAGE_0042		1.328 vacuum pump (particle counter) - no flow											
43	PLC_FAULT_MESSAGE_0043		1.338 vacuum pump (particle counter) - no flow											
44	PLC_FAULT_MESSAGE_0044		1.300 particle counter PC01 (working chamber) - fault											
45	PLC_FAULT_MESSAGE_0045		1.310 particle counter PC01 (working chamber) - fault											
46	PLC_FAULT_MESSAGE_0046		1.320 particle counter PC01 (working chamber) - fault											
47	PLC_FAULT_MESSAGE_0047		1.330 particle counter PC01 (working chamber) - fault											
48	PLC_FAULT_MESSAGE_0048		1.350 viable air sampler - fault											
49	PLC_FAULT_MESSAGE_0049		1.360 viable air sampler - fault											
50	PLC_FAULT_MESSAGE_0050		1.370 viable air sampler - fault											
51	PLC_FAULT_MESSAGE_0051		1.380 viable air sampler - fault											
52	PLC_FAULT_MESSAGE_0052		1.830 H2O2 LC - fault											

Figure 7.8: *ZenOn\_Information* sheet.

Last step before interacting with ZenOn will be to delete the column *B* of the new sheet, Fig. 7.9



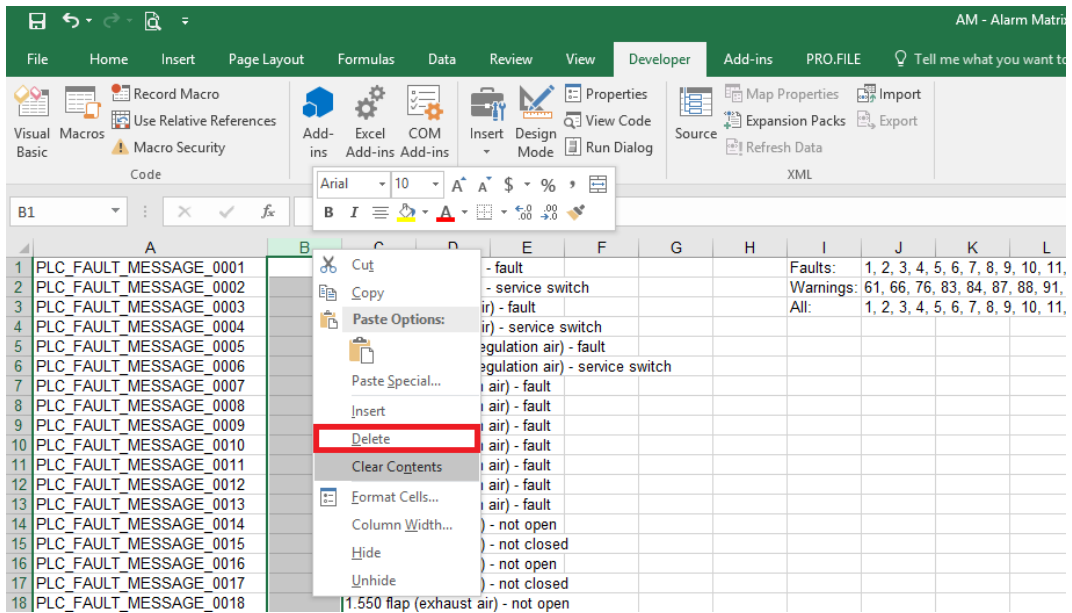


Figure 7.9: Delete column B before interacting with ZenOn

In ZenOn project it will be necessary to create two different divisions:

1. **Alarm areas** that will correspond to each part of the installation (i.e. *ISOLATOR*, *AIR-LOCK*, etc.). Fig. 7.10
2. **Alarm classes** For us will be divided into *ALARM*, *WARNING* and *INFORMATION*. Fig. 7.11

### 7.3 Importing into ZenOn

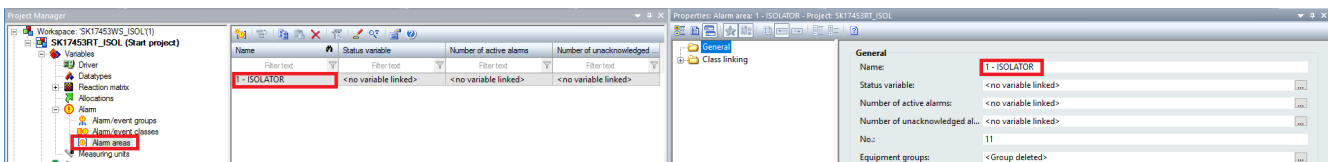


Figure 7.10: Creation of the alarms for each part of the installation

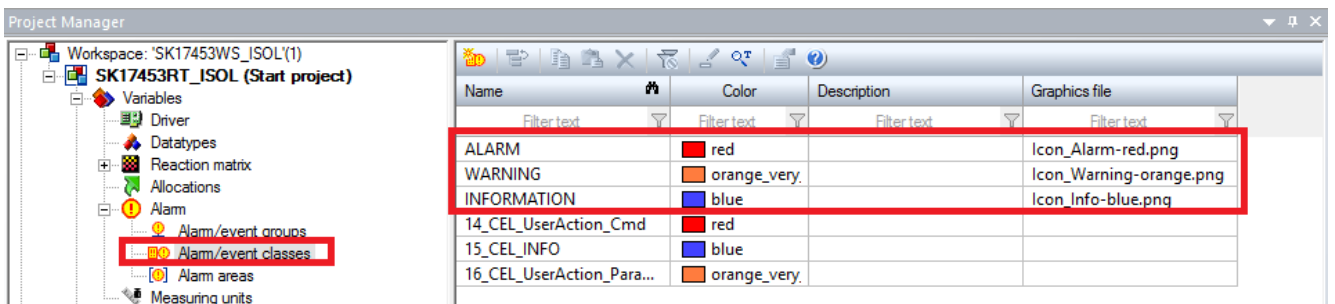


Figure 7.11: Creation of the classes for each part of the installation

Before importing the alarms into a variable array it is important to check that there are no other alarms messages already in the system. This is the case for those projects that are based on former ones. To remove the alarm messages select the *Language file* in the lateral menu and search for *PLC\_FAULT\_MESSAGE\_\** in the Keyword filter bar, as in Fig. 7.12

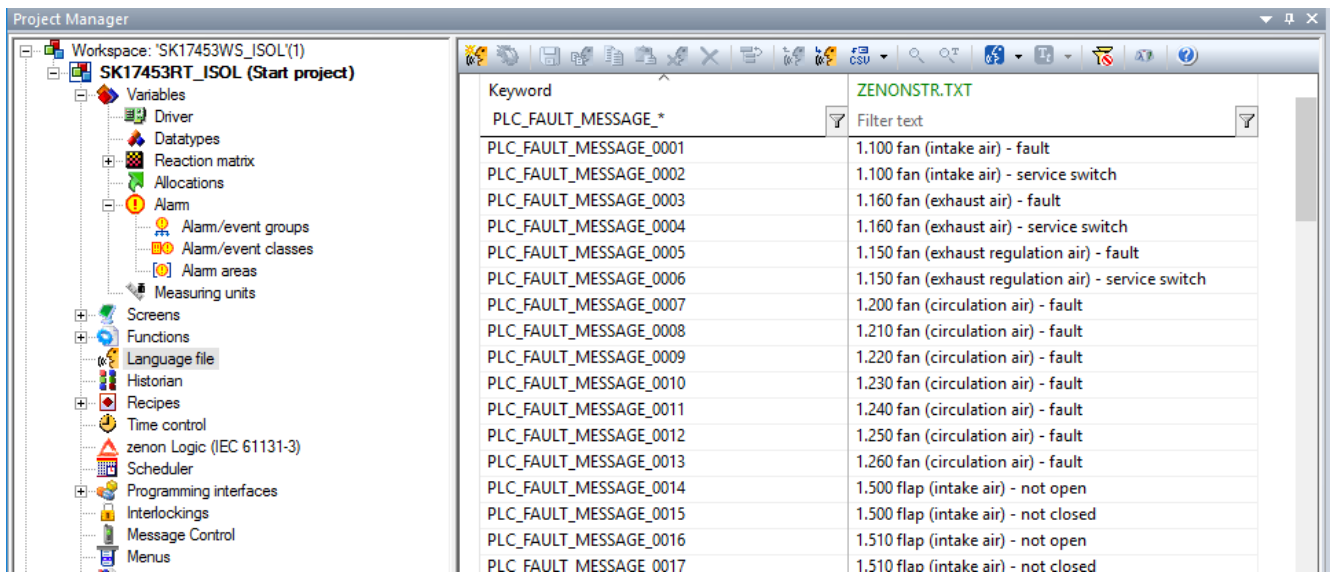


Figure 7.12: Remove alarm messages from previous projects

It is needed to export the complete list of messages to a .csv file. This file initially contains all the messages but the ones related with alarms that will be added soon. Right click over any message in the *Language file* (Remember to remove the filter), click *Extended import/export* and choose *Export CSV all*, Fig. 7.13. It will be saved by default as *Language.csv*. Sometimes there are errors when saving with regard to the formatting of the characters. When saving, press *No* and *Cancel*. The file will be saved anyway.

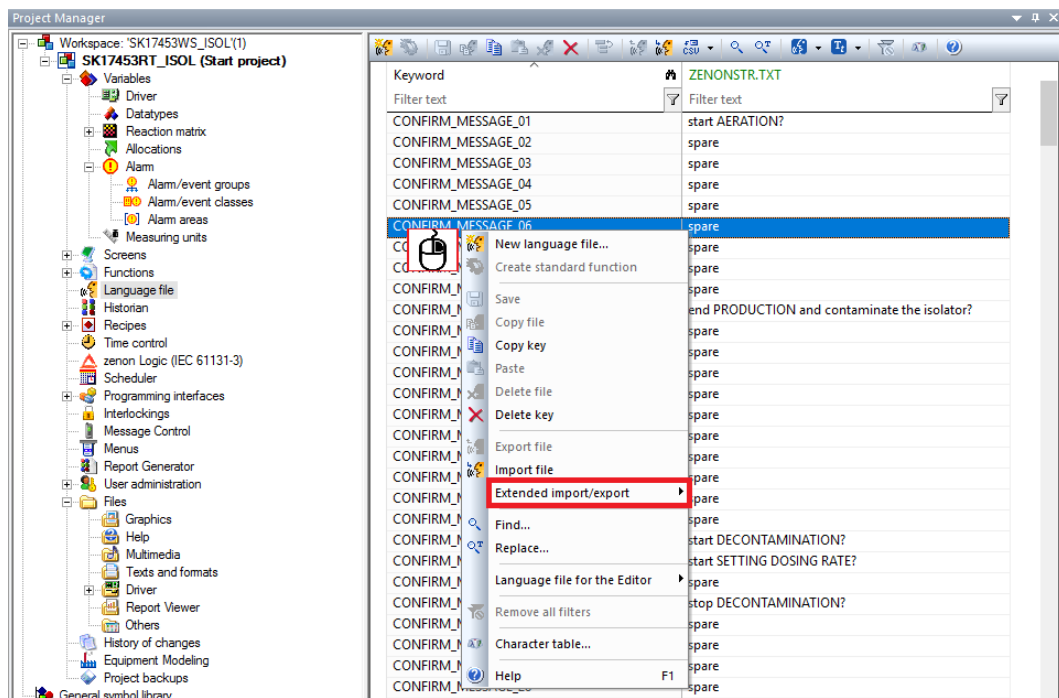


Figure 7.13: Remove alarm messages from previous projects

The messages of the alarms should be added to the file *Language.csv*. Copy just the number of rows the sheet *ZenOn\_Informations* that have a message and paste them at the bottom of the file, Fig. 7.14

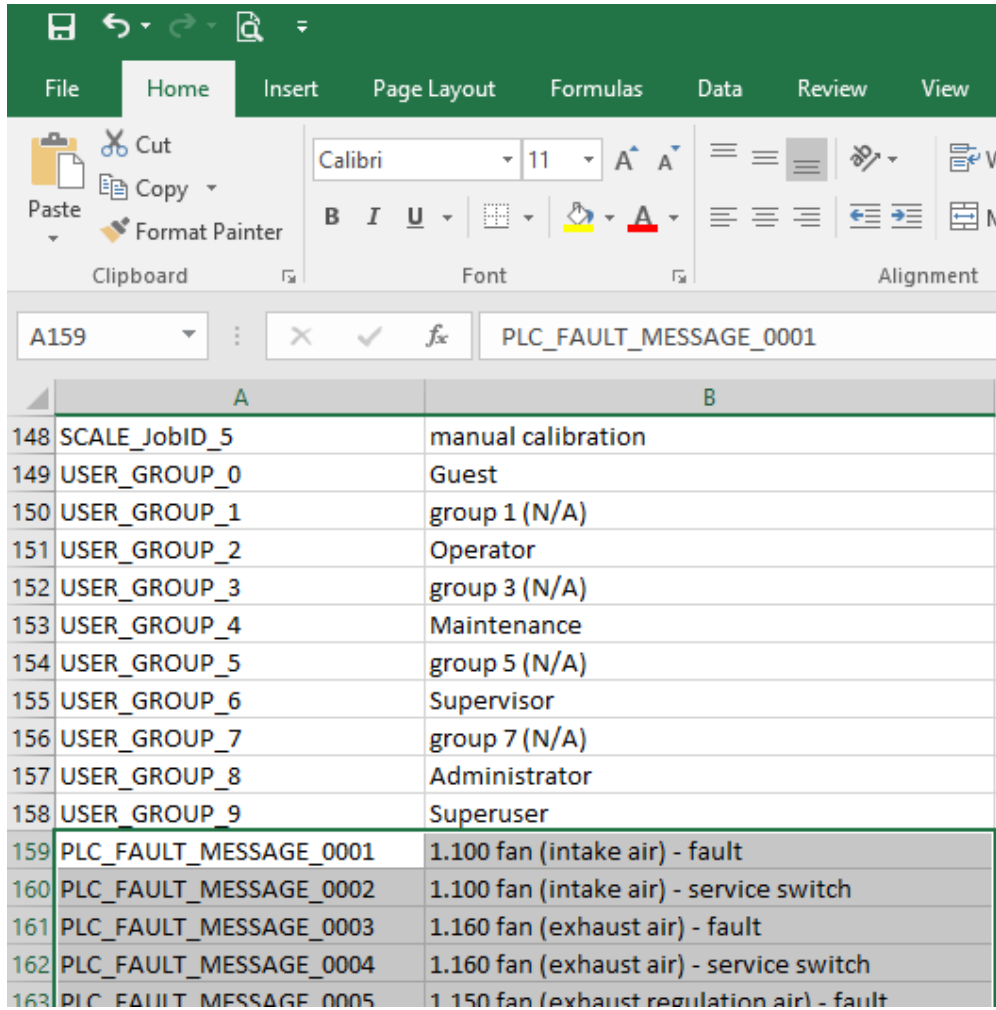


Figure 7.14: Create the new *Language.csv* file before importing it

After saving the file it should be imported into *Language file* in ZenOn. It is done similarly to 7.13 but clicking *Import CSV* inside the *Extended import/export* menu.

Now all the messages are inside our SCADA system but they should be classified as *Faults*, *Warnings* or *Info* messages. In order to do that macros in ZenOn Editor are used. Open an random screen in ZenOn Editor and double click over any point of the screen to open a popup with the list of macros. The macros that will be used are highlighted in Fig. 7.15

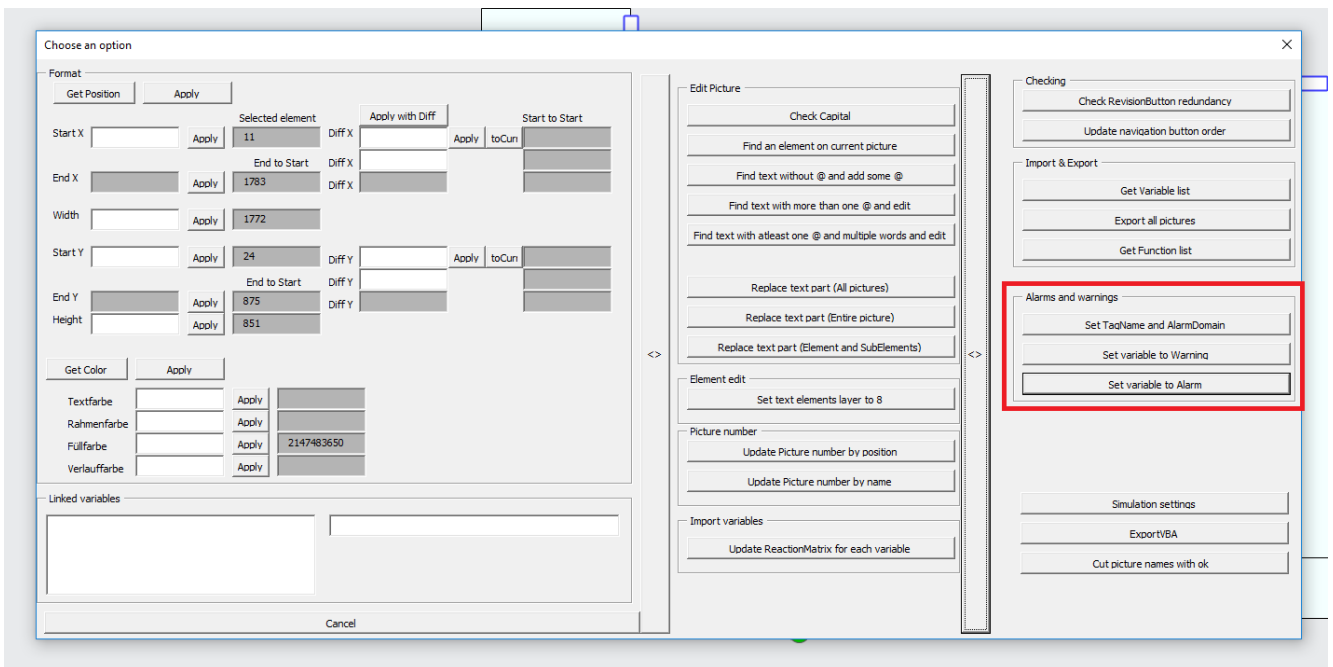


Figure 7.15: List of macros in ZenOn Editor. Highlighted the ones to set the alarm/event class

The order to run the different scripts is given from top to down. After clicking over *Set TagName and AlarmDomain* a popup screen will appear. Here we should copy the values given in cell I3 (All) of *ZenOn\_Information* sheet. Fig. 7.16

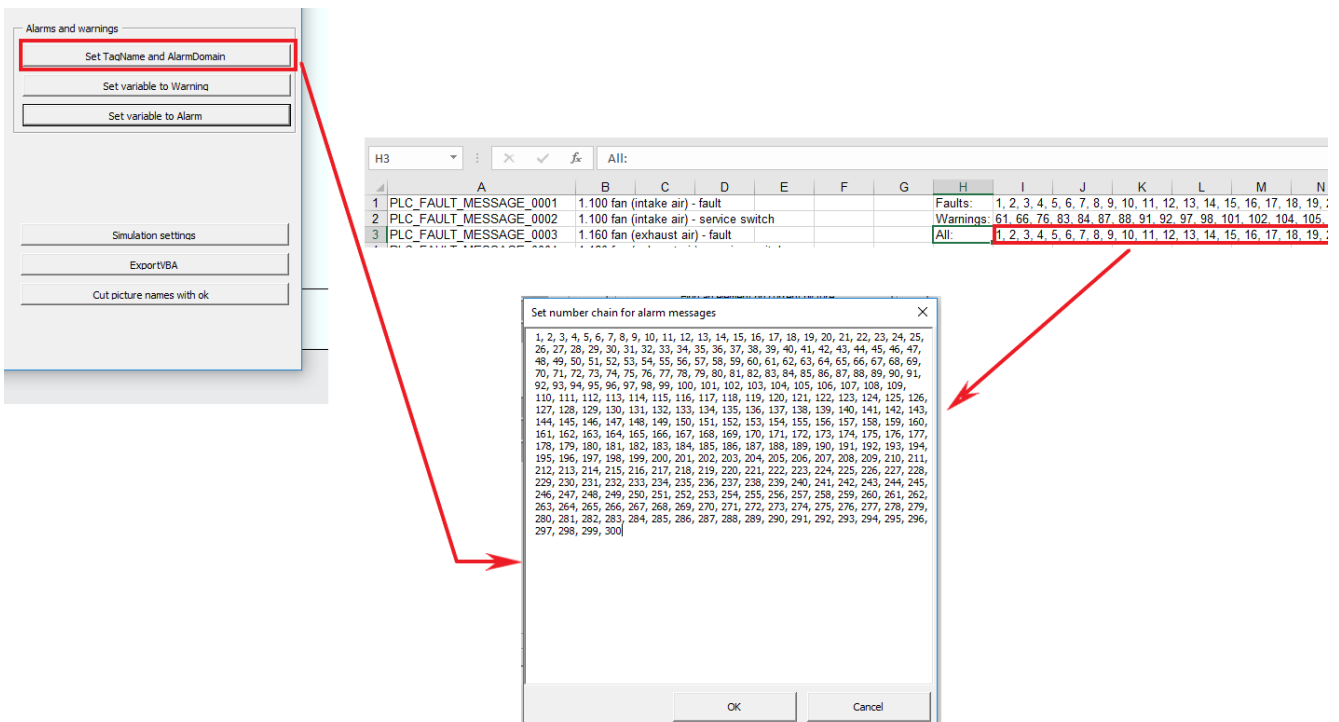
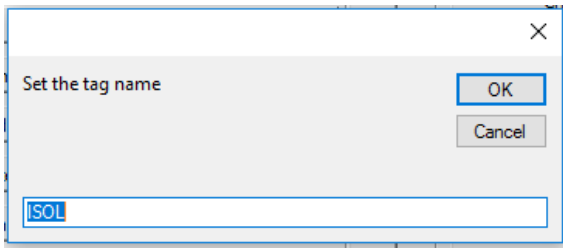
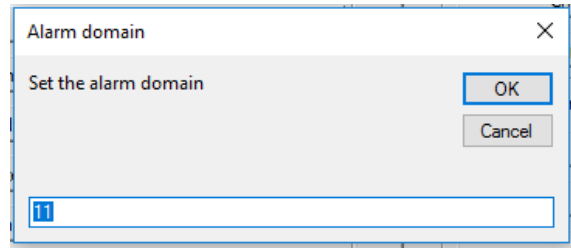


Figure 7.16: List of macros in ZenOn Editor. Highlighted the ones to set the alarm/event class

The process will follow asking the user for a tag name for the messages and the alarm domain (by default is 11 for faults, 12 for warnings and 13 for info messages) as shown in Fig. 7.17



(a) Setting tag name to alarm messages



(b) Setting alarm domain to alarm messages

Figure 7.17: User input for tag name and alarm domain

It will take around half a minute to run the script. Next macro to run will be *Set variable to Warning*, Fig. 7.18

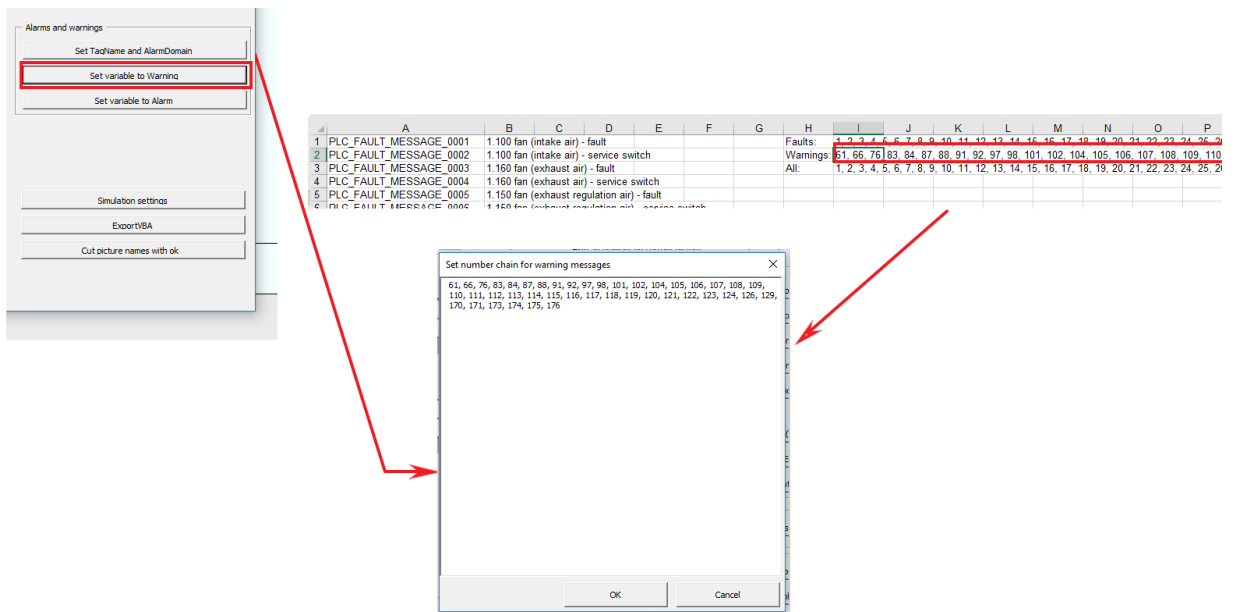


Figure 7.18: Set variables to warning

Last step would be to run the macro *Set variable to Alarm* as shown in Fig. 7.19

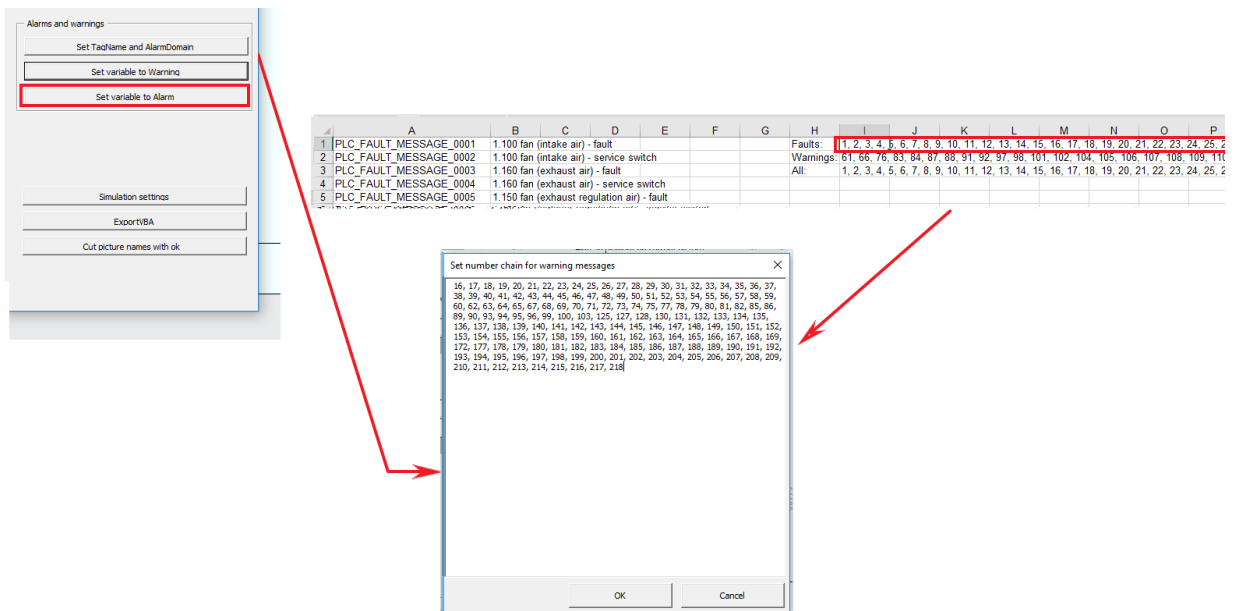
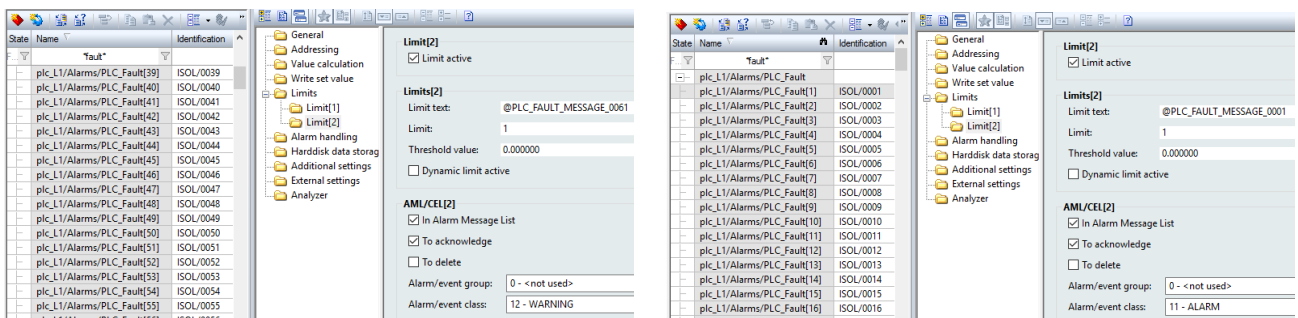


Figure 7.19: Set variables to alarm

It is possible to check if the scripts performed as expected by checking the alarm domain for each fault variable. An example of each one can be found in Fig. 7.20



(a) Example of warning configuration

(b) Example of alarm configuration

Figure 7.20: Result examples after the whole process

## 7.4 Special requirements

### 7.4.1 Single alarm acknowledgment

### 7.4.2 Acknowledgment class E alarm only for some users

# Chapter 8

## CEL: Chronological Event List

# Chapter 9

## Reports

### 9.1 Introduction

The reports are a fundamental part of what customers are expecting from a SCADA. They are basically documents with specific data. In our case it will collect information about leak tests, decontamination cycles or batch production.

In ZenOn there are two different ways to create reports. The default one is given over the navigation menu as *Report generator*, Fig. 9.1.

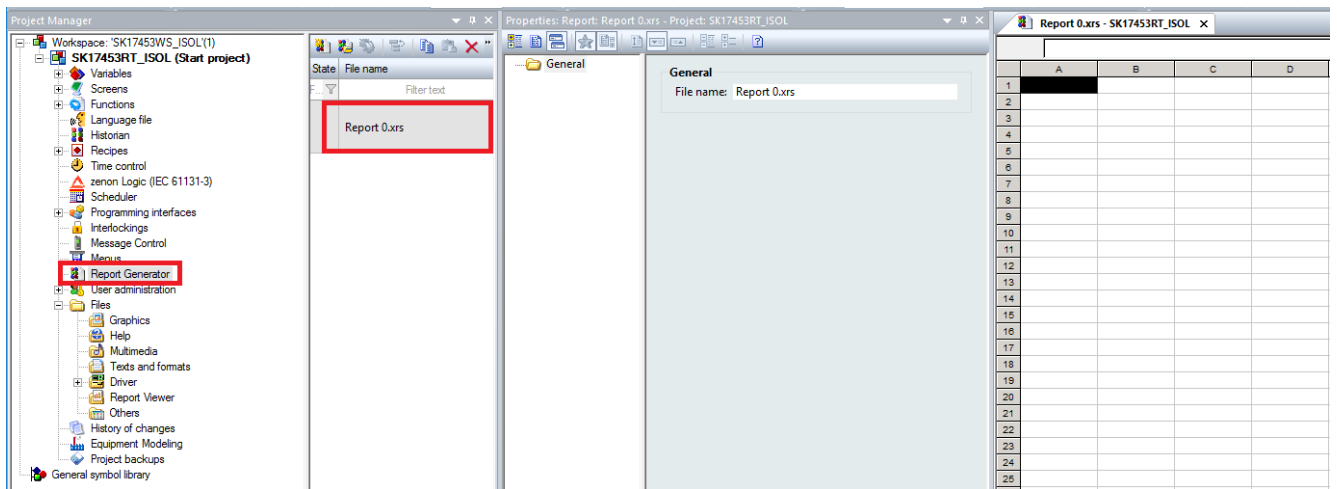


Figure 9.1: Default report generator

There is a much more flexible way to generate reports. The creation of *.rdl* files allow to use Visual Studio to generate the reports in a GUI with more options, like the link to different signals belonging to the variables, internal Windows' signals, etc.



## 9.2 Report Definition Files (RDL)

The use of Microsoft Visual Studio for the development of reports will make the process simpler and easy to use the already created templates for future projects.

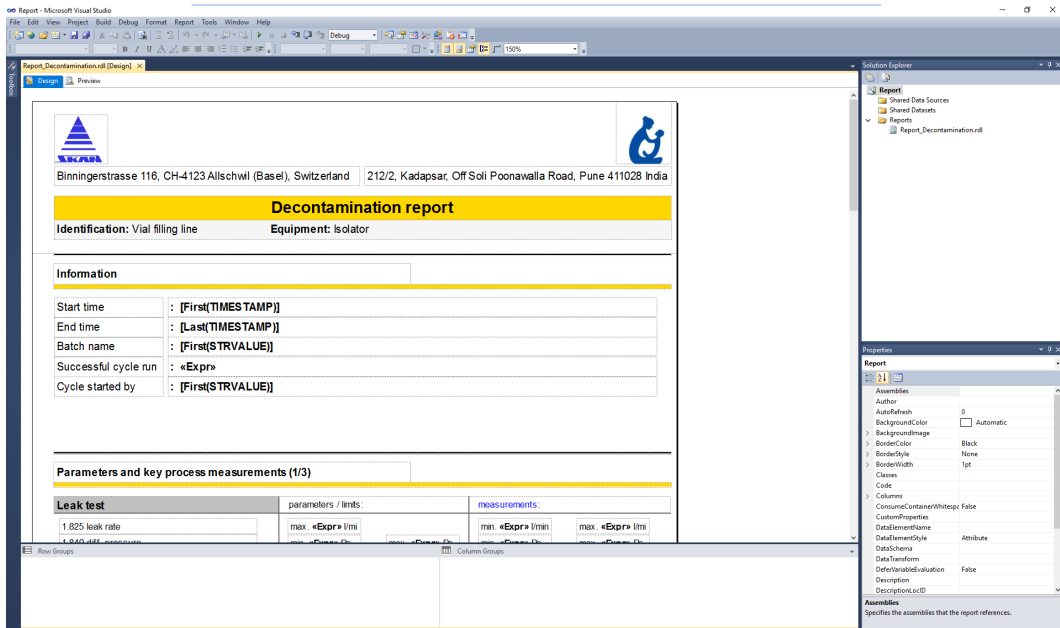


Figure 9.2: Example of report design using Visual Studio

### 9.2.1 RDL creation and dataset configuration

To create a new report definition file it is needed to click over *Files* in the navigation menu and then right-click over *Report Viewer* to add a new file. A menu will pop-up to select among the different datasets that can interact with the report, Fig. 9.3.

The variables available for the connection with the reports could be runtime, CEL, historian or alarm variables. It will be important to have a view of which variables could be chosen and for that reason *Report data* menu should be active, Fig. 9.4.

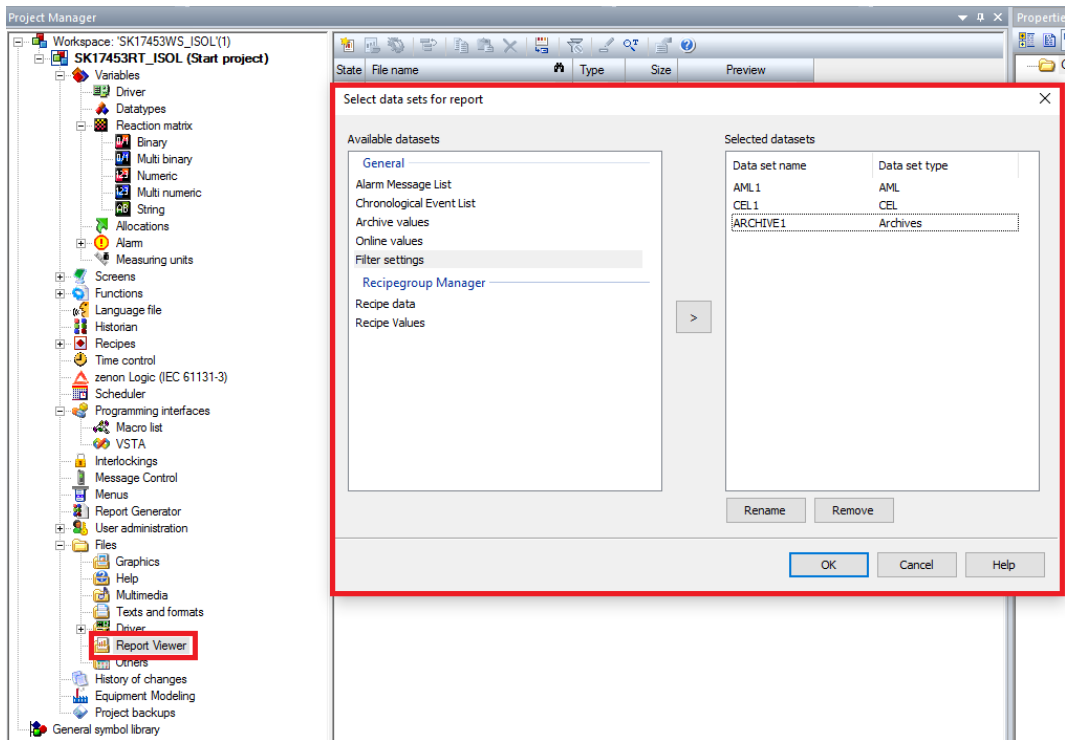


Figure 9.3: Creation of a new RDL file and configuration of its datasets

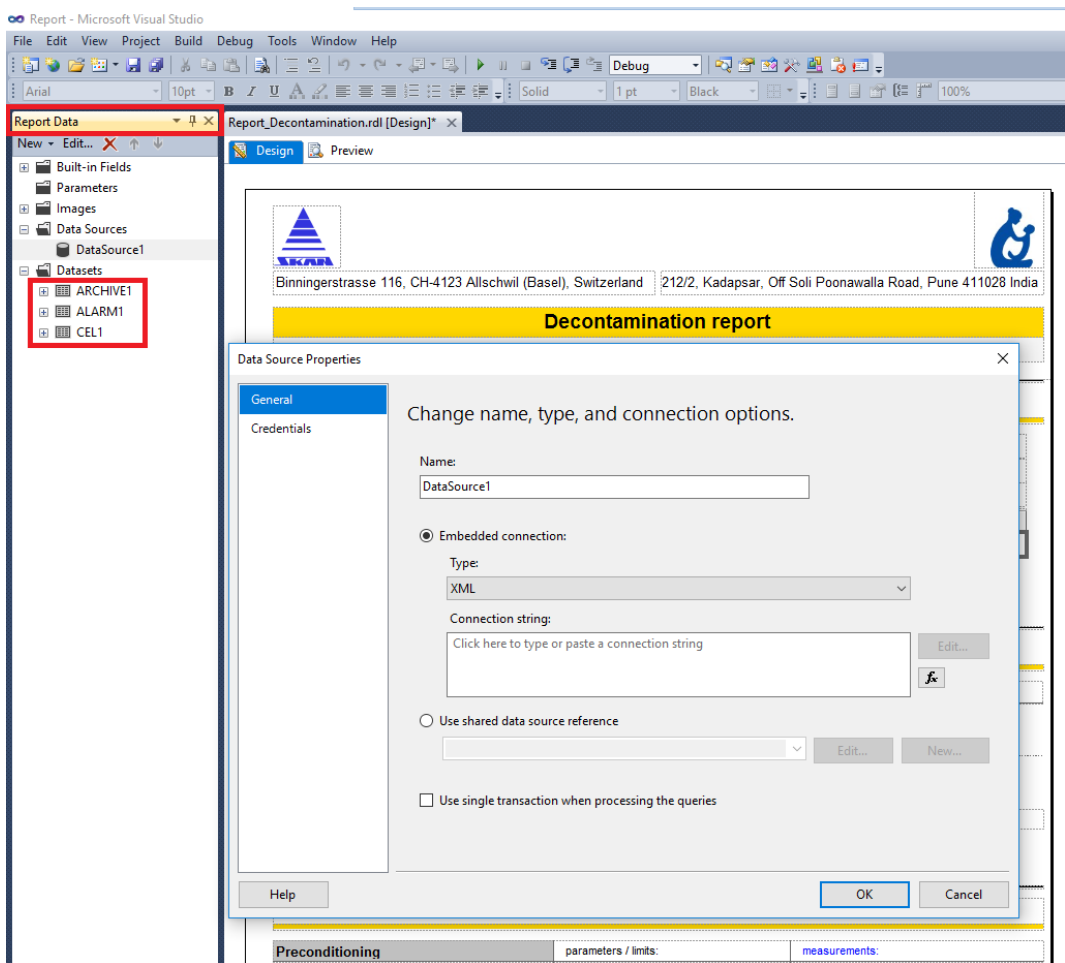


Figure 9.4: View of the report data

## 9.2.2 Archives for reports

At this point we should already know what values will be shown in the report. There are several ways to link data with the reports.

Dedicated archives will be created for each report type (i.e. *deco*, *production*, etc.).

The main reason to do this is to keep the idea of **having the less logic as possible in the SCADA and the more in the PLC**. There are pros and cons for this choice and it can be listed below.

### *Pros*

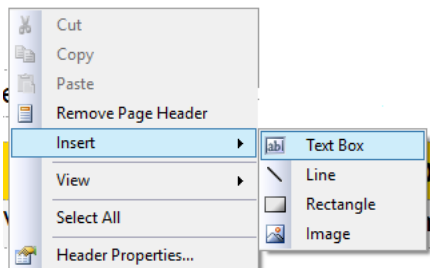
1. **Logic** More logic set into PLC and less to the SCADA system.
2. **SQL** RDL data retrieval is based in SQL queries. The simplification of these queries is achieved by the creation of explicit variables for each report. Very light queries. Easy to debug.
3. **Value limits** Possible to overcome the limit of 5000 values per variable by reducing the sampling rate for the variable in the new archive.

### *Cons*

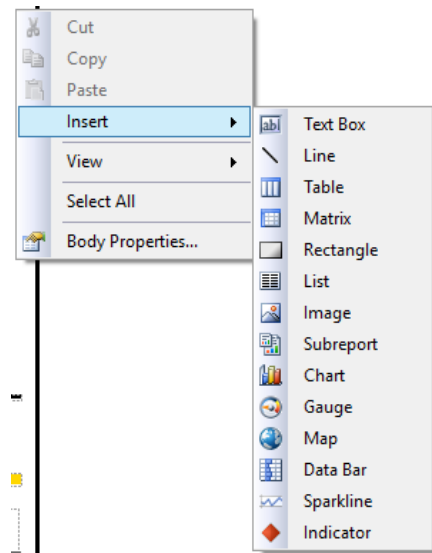
1. **Variables** Creation of redundant variables.
2. **Memory** Memory management of these archives should be treated separately.

## 9.2.3 Report layout and configuration

The layout could be configured in such a way that the report is easily printed into paper, *.pdf* or seen into the screen. It will have static fields like images, lines, descriptors or sentences and be able to show the values of different variables at specific times. In Fig. 9.5 all the elements for the report's construction are listed.



(a) Layout elements for header



(b) Layout elements for body

Figure 9.5: Layout elements for a RDL

The most used elements in our projects are *Text Box* and *Table*. In *Text Box* only static text is introduced, while in *Table* values related with the variables could be displayed. In our case we only display one value, i.e. *max*, *min*, *total*, *length*, but in general it could be used to show more than one value linked to an specific variable or combination of several.

It is possible to add a new table into the report and link the data to defined variables of the different sources. This is described in Fig. 9.6

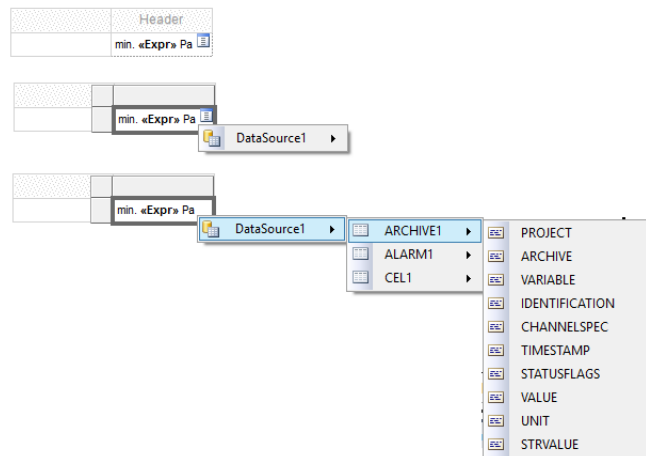


Figure 9.6: Add a table with a new variable

In the previous example it was introduced a variable into a column of the table. In most of the cases we introduce only one variable per table and the table is configured to show only one variable with only one value. In the example on Fig. 9.7 could be found which variable is set in a table. As mentioned above the most of the variables used in tables are internal variables that store only one value.

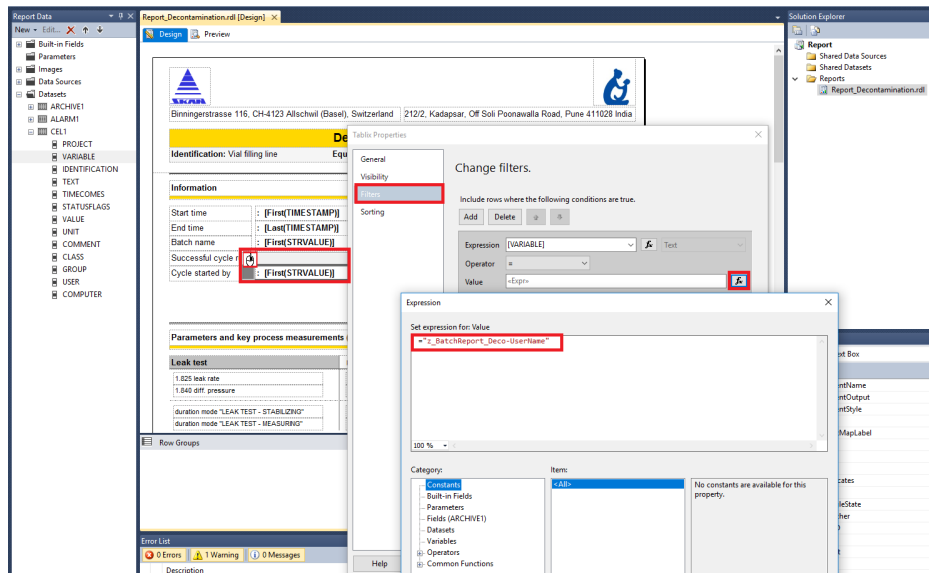


Figure 9.7: How to find the variable set in a table

## 9.2.4 Customized fields

In some cases it is needed to customized some fields, like for example if a deco cycle has ended successfully or set different colors for each alarm type.

### Alarm entries color

It is important to understand and differentiate between two different parts involved in the process:

1. **Data** - Contains the values that are provided by ZenOn and are stored in a database (with tables, properties, etc.)
2. **Report elements** - Text boxes, lines, images that are created/imported.

Check Fig. 9.8 to visualize these two main parts, data with all sources with their fields and report elements.

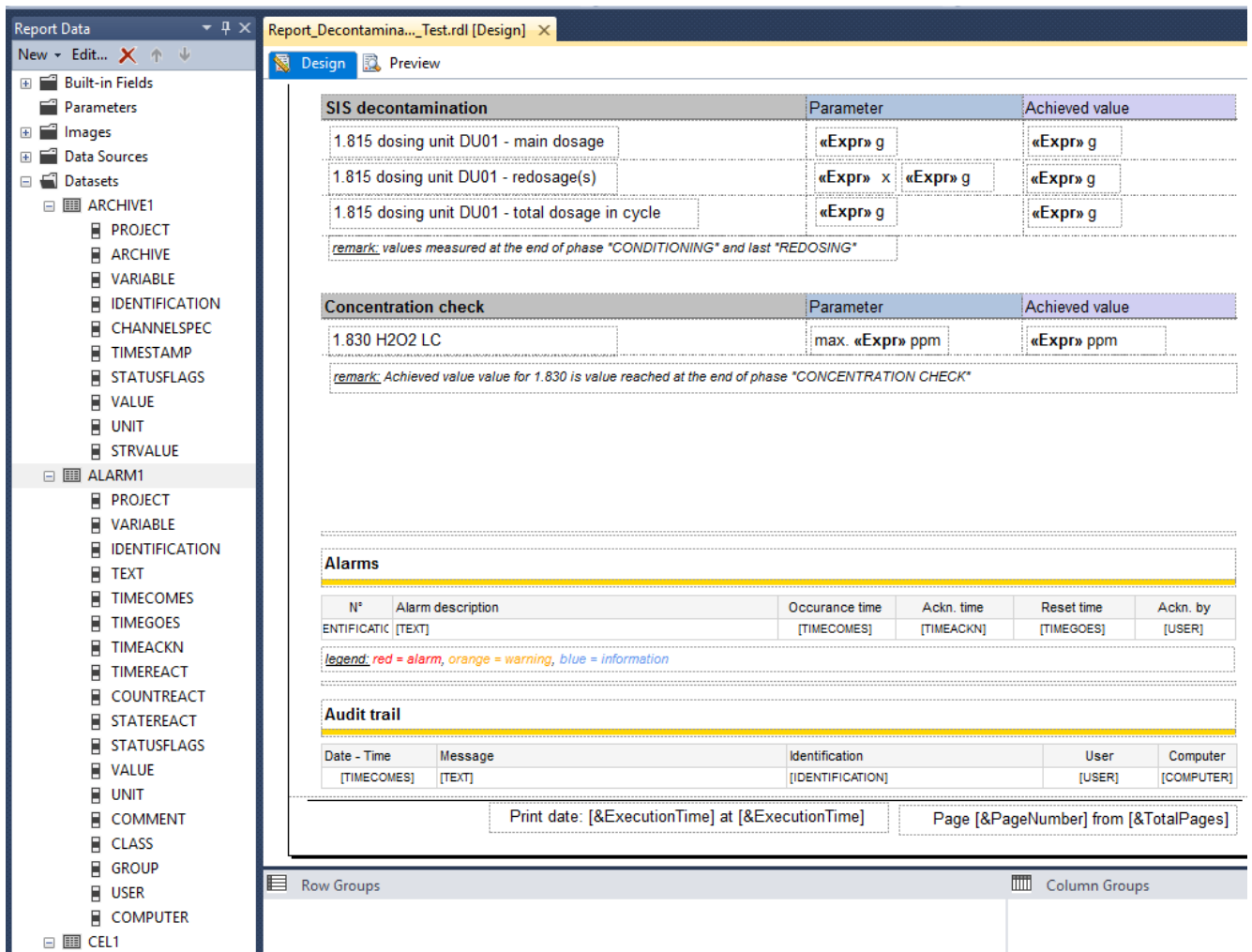


Figure 9.8: Report Data and RDL editor

To change the text's color in the alarm list, it will be necessary to know explicitly what it is wanted to be changed, how to find it and how to change it.

Fig. 9.9 shows the way to reach the menu in which the fields' font could be found.

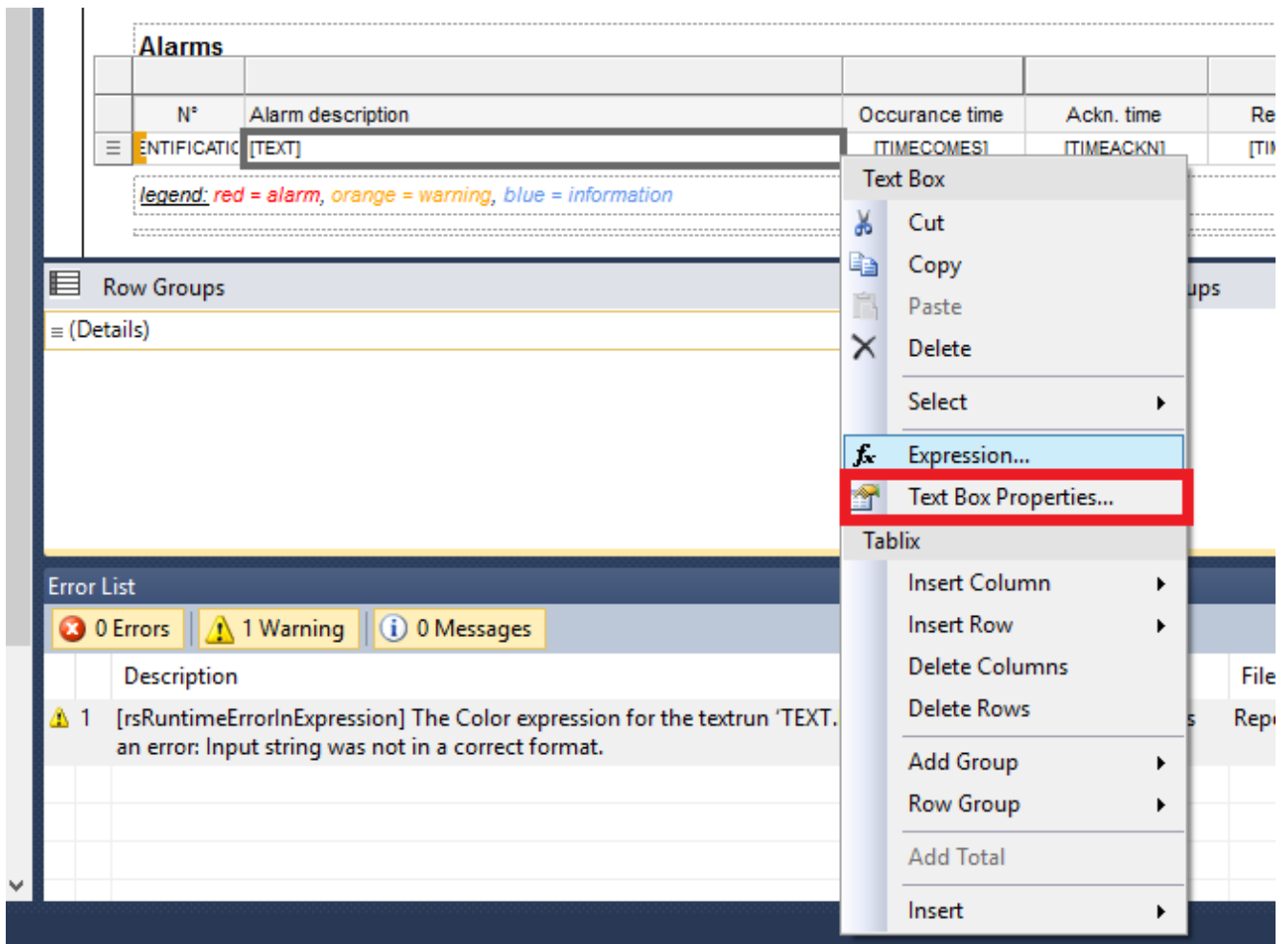


Figure 9.9: How to get to the font in the alarm fields

To check if the alarm is of type *alarm*, *warning* or *info* it will be needed to interact with the data provided by the alarm source. Then the option of expression should be select as shown in Fig. 9.10

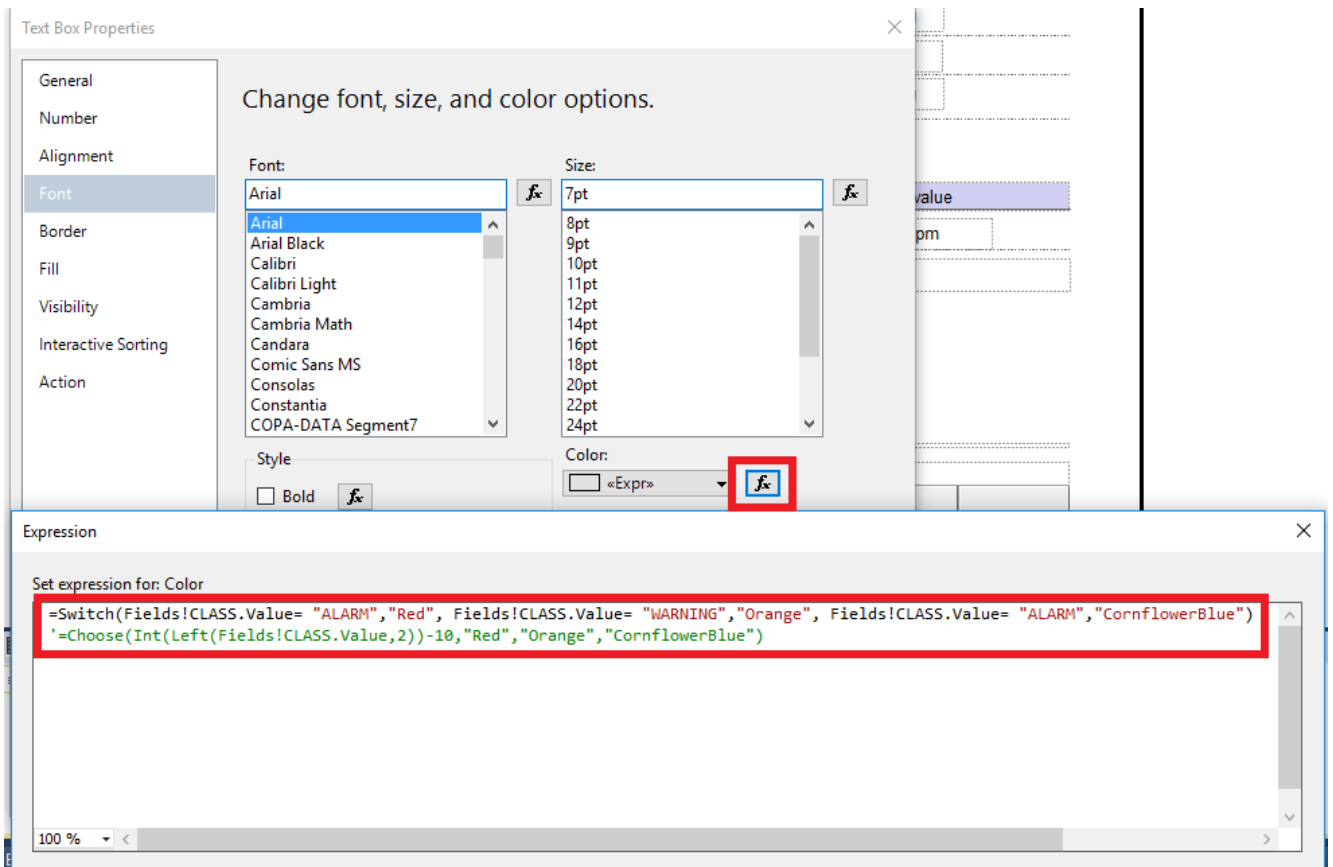


Figure 9.10: Selection of *Expression* and its value

There are two different expressions in the previous figure. First one is the one used currently with the alarm classes defined in the importing script. Second one is using an old definition of the alarm groups and taking advantage that the first characters of each group were two digits. To understand better why there are two expressions take a look to Fig. 9.11

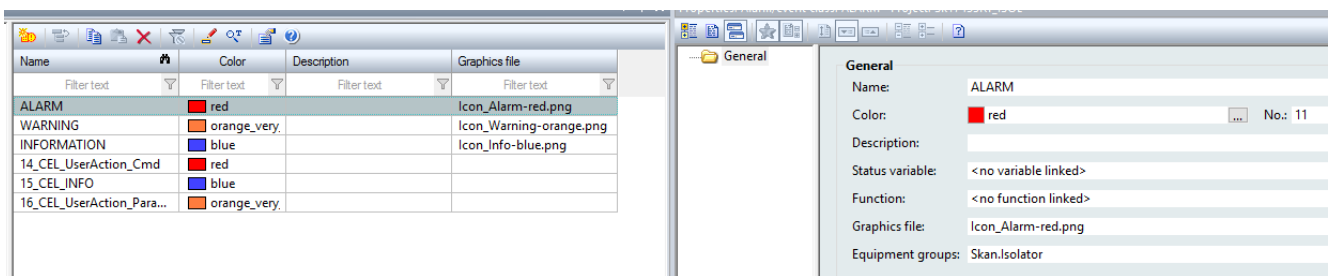


Figure 9.11: Alarm classes in the example project

At the beginning, each group used to have two digits and a dash (XX\_) to begin the name (like the three last groups in Fig. 9.11). Using the new importing script (as described in this manual) to create the alarms variables, alarm groups would not have this naming convention.



## 9.3 Models for report generation

### 9.3.1 Lot archive with allocations

#### Summary

The model will have a dedicated archive for each kind of report (decontamination, production, etc.). A lot is defined as the period to complete one of these activities, with no specific time length. For each variable in the report, two entries will be stored in the archive: one at the start and one at the end. If values like *maximum* or *minimum* should be on the report, PLC should retrieve them as final entries or internal variables should be created on ZenOn to this end.

**PLC variables will not be used directly on the archives or on the reports**, which allows to work independently on RDL files and SCADA.

#### Pros

- Possibility to reuse RDL files without changing any variable or configuration on it.
- Very small memory required for a report generation.
- Constant size for the memory per report for archive and *.pdf* files.
- As it uses lots, possibility to view or regenerate the report without data corruption or loss.

#### Cons

- Complexity: Special configuration for the archive, extra variables, interlocks, reaction matrices and allocations.
- Need of two internal variables per variable in the report: Variable itself and a check boolean.
- Possibility to lose around 0.3 seconds at the moment that start command is sent.

#### Description

The idea behind the development of current model is to be able to deliver a report without expending engineering time on RDL files, with a very light memory consumption and reusable by changing the allocations as the unique SCADA need.

Allocations are internal variables that will be storing another variable content whenever an event happens. Our archive will only contain these variables, and new entries will only be stored *on change* (if the archive is open). If we trigger this changes just after starting the archive and before opening it the result is a model as expected.

There is a main issue. The archive should be opened and ready before storing any data. Allocation of the variables takes more time than opening the archive and saving the variable values on it (even using ZenOn scripts). This means that some variables will be easily saved in the archive while other will change before the archive is ready to read them (so nothing will be stored for them).

The issue was solved by using boolean variables that will be activated/deactivated with each change in the values of report variables.

In Fig. 9.12 an schema of the current model could be seen.

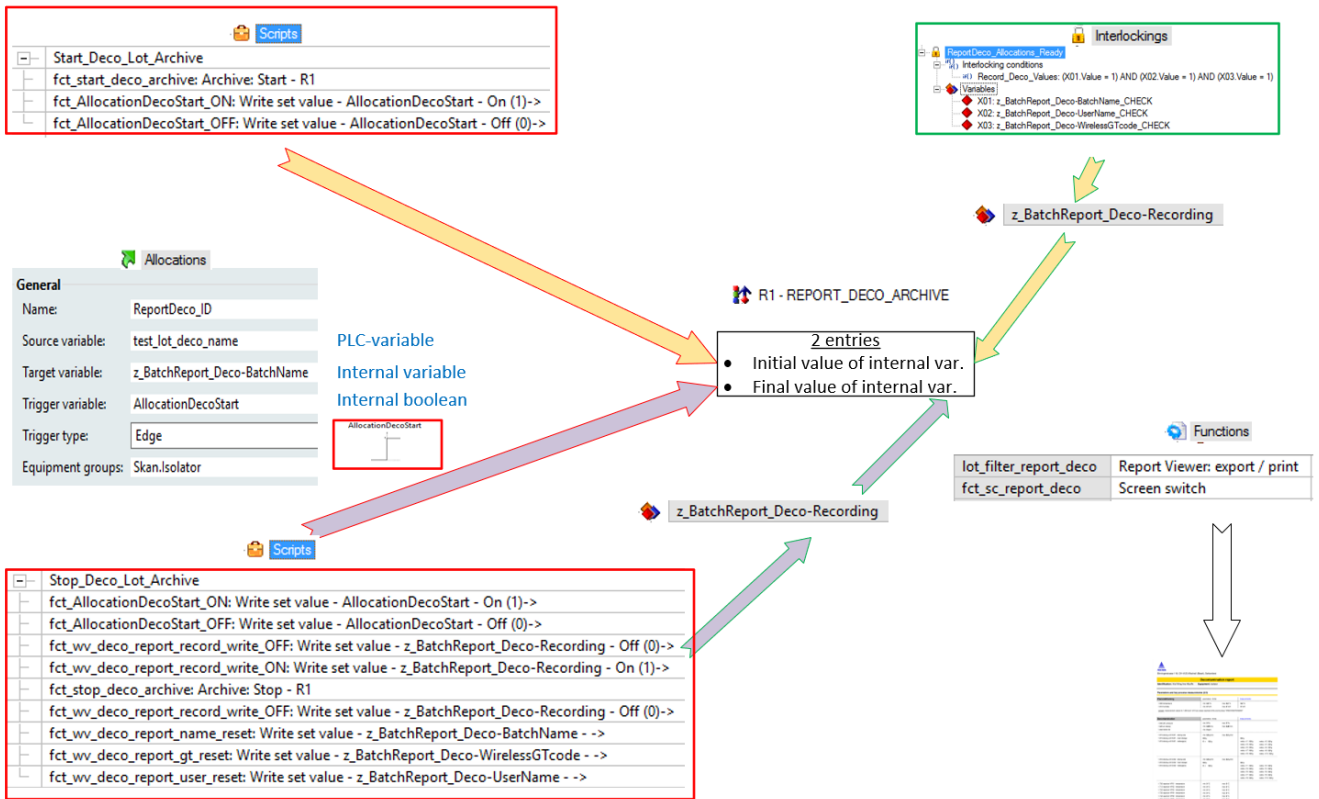


Figure 9.12: Model schema for reports using allocations and lots

The archive for each report should be configured as a lot archive in which the records will be made by triggering using a boolean variable. The variables inside this archive will be only internal variables that are written through allocations. The complete description of the archive configuration could be seen in Fig. 9.13

The reaction matrices associated with each of the variables in the report should be created. This is necessary to control our control variables. The drawback is that there will be too many reaction matrices. The reason is to have independent feedback for each variable. To check the two states of the reaction matrices look at Fig. 9.14

In order to write the initial entry for each variable in the archive should happen two things:

1. **Start the archive.** The archive should be properly activated with the lot name.
2. **Interlocking.** All the allocations have been performed and the check variables are set to 1.

Please check the yellow arrows in Fig. 9.12

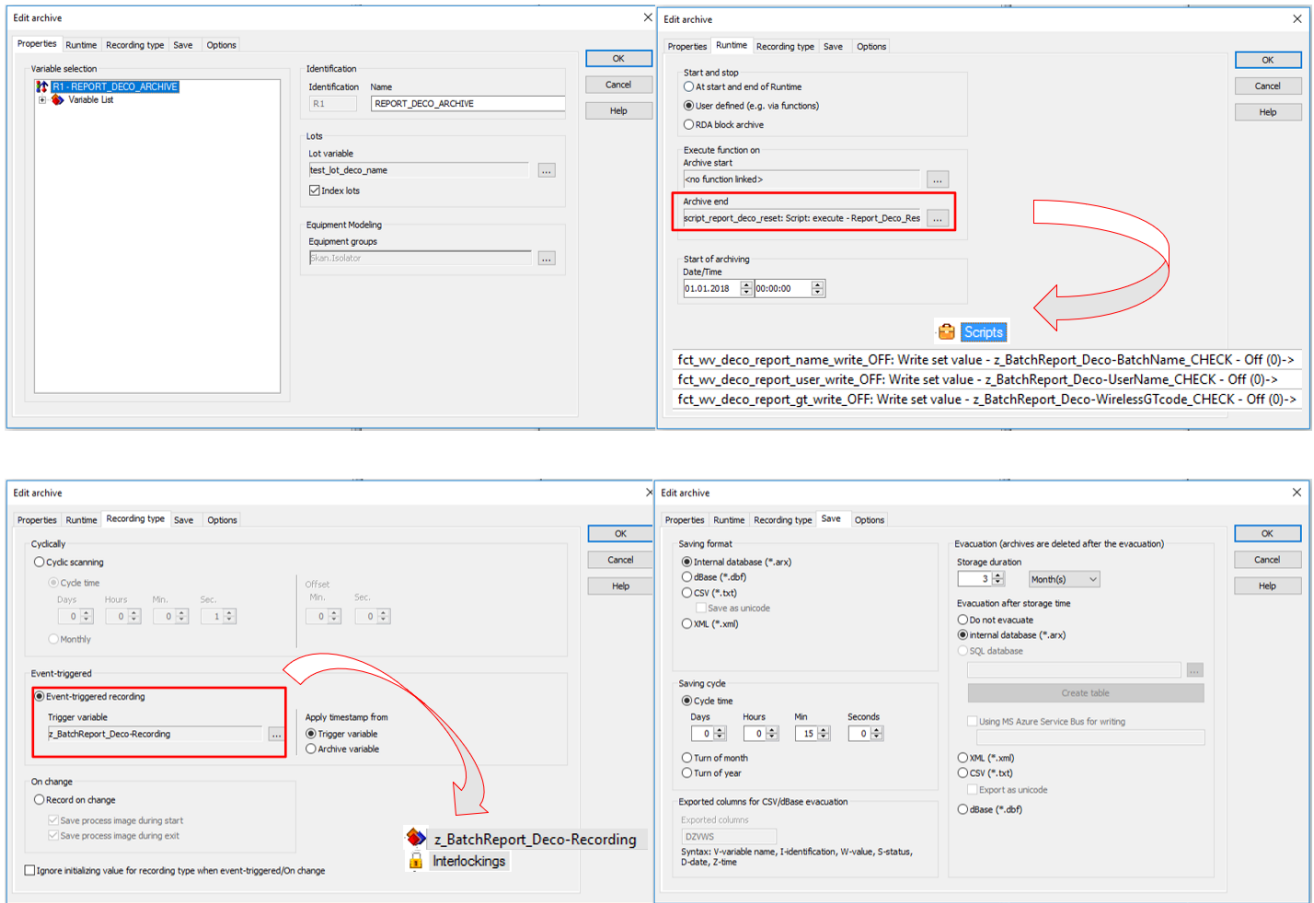


Figure 9.13: Archive configuration example

A result example of the current model configuring only 3 variables (decoID, user name and wireless GT code) can be seen in Fig.9.15

**Note:** Working with lots it is mandatory to have unique names for each lot. If the lot name is repeated the times will be mixed up and it will not be possible to retrieve reliable data.

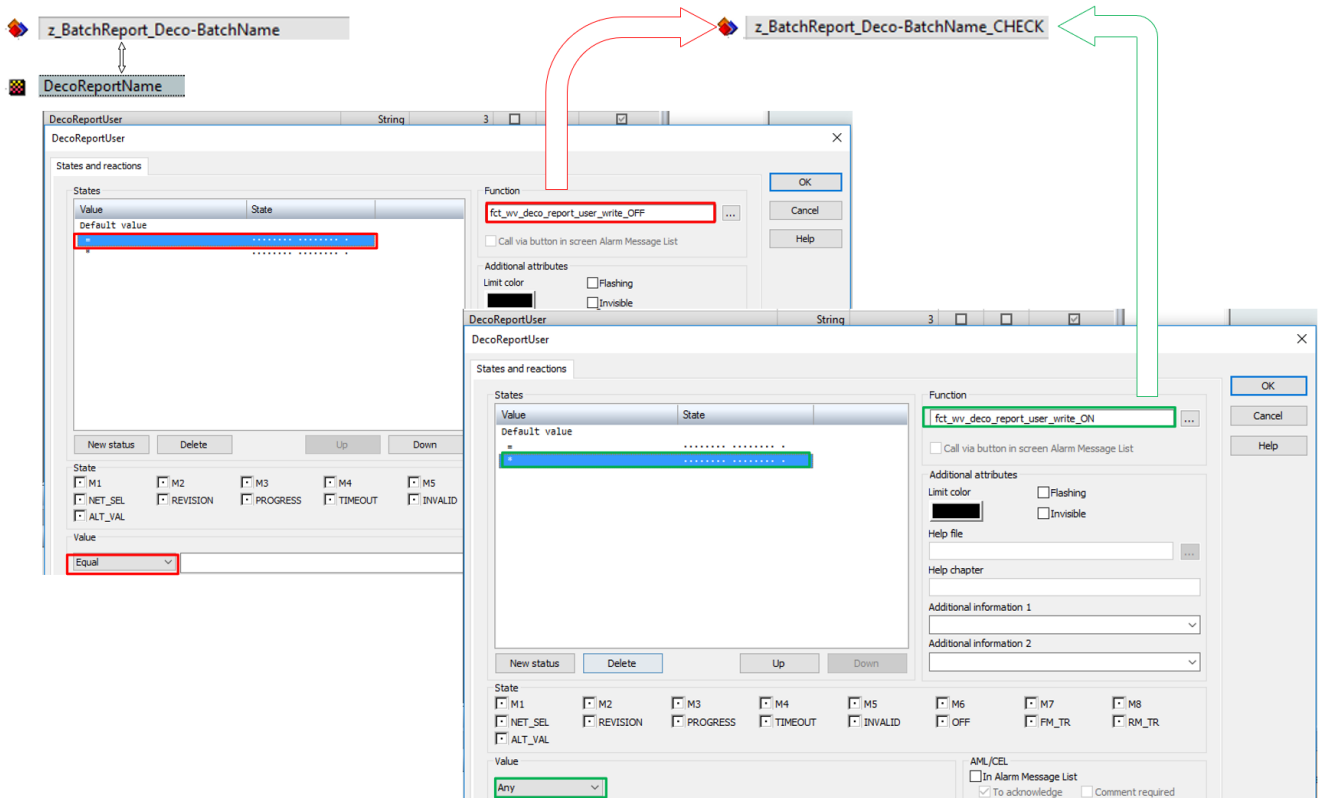




Figure 9.14: States of a variable's reaction matrix

1 of 4 100% Find | Next



SKAN AG  
Binningerstrasse 116, CH-4123 Allschwil, Switzerland



SERUM INSTITUTE OF INDIA PVT. LTD.  
(SEZ UNIT NO.:PBP-1)  
Poonawalla Biotechnology Park SEZ  
(Developed by SEZ Biotech services Pvt.Ltd)  
Manjari Bk., Tal. - Haveli, Pune 412207 (India)

---

**Decontamination report**

---

**Identification:** Vial filling line      **Equipment:** MSEZ-3-GF

---

**Information**

---

Start time : 2018-08-29 16:47:05  
 End time : 2018-08-29 16:47:17  
 Decontamination ID : d0000001  
 Cycle started by : Skan  
 Wireless GT code : g0000001  
 Successful cycle run : **NO**

Done by	Checked by

---

Print date: 2018-08-29 at 16:49:29
Page 1 from 4

Figure 9.15: Example of result on ZenOn report viewer screen

# Chapter 10

## VBA scripts

### 10.1 Introduction

The use of VBA scripts help us with repetitive or customized tasks that could not be done so easily or in such short amount of time with ZenOn standard functionalities. It makes no sense to write each line of code for every script. The normal behavior and the configuration options will be described for proper use.

### 10.2 Decontamination cycle simulation

This is specially interesting when reports are needed in the project. It is always difficult to test how they look and if the data provided by them is displayed correctly. Even more if we are not familiar with WinMode or we do not have a pre-configured PLC module to simulate the signals from isolator's sensors and measurement devices.

In order to ease the procedure and make it independent from PLC devices or external software, a simulation of the decontamination cycle has been developed.

The time for a decontamination cycle is usually quite long and waiting for it to finish to check the results in a report leads to a waste of developer's time. To avoid that the only parameter that will be provided by the developer to test the tool is deco-cycle's length in minutes.

The result for a 15-minutes simulation cycle could be seen in Fig. [10.1](#)

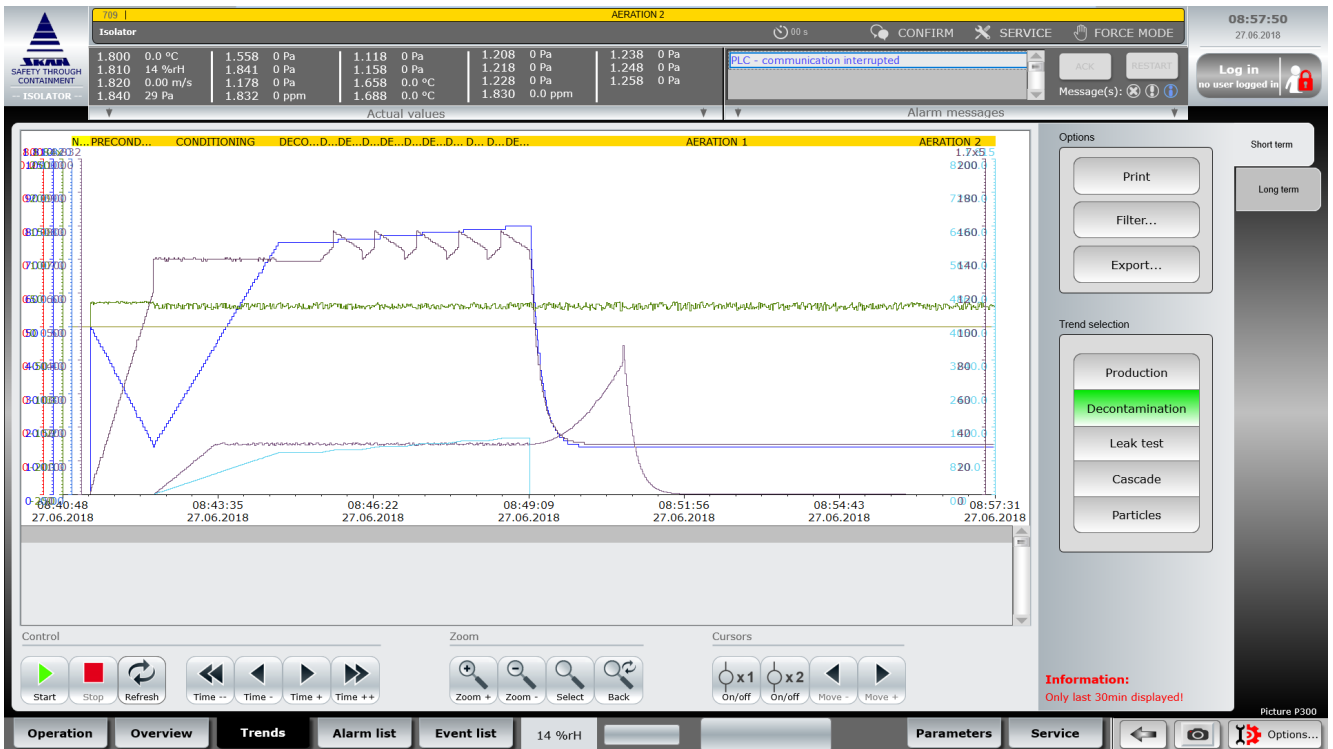


Figure 10.1: Resulting trends of the deco-cycle

## 10.2.1 How does it work?

The diagram in Fig. 10.2 represents the flow chart of the simulation.

The initialization of the simulation requires two initial conditions to be triggered:

- Set the number of minutes to simulate the cycle.
- Accept the chosen conditions.

After clicking a button which will bring up a pop-up screen with the time-selection menu an integer should be introduced. This value is the number of buttons that will last the simulation. By clicking 'Ok' this minutes will be converted into seconds and the lengths of different phases during the deco-cycle will be set proportionally to the total length.

To meet the second condition it will be needed to accept the changes set by the first condition. This can be done in the pop-up screen by clicking 'Run !'. It will trigger a variable, 'z\_Simulation\_Deco\_Flag', which helps a script that is running each second to know when to increase a counter or when to set it to -1.

Different scripts will be calling regarding the value of this counter variable. The limit values (those for which the counter triggers a different script) are just the addition of the length of the previous phases' length.

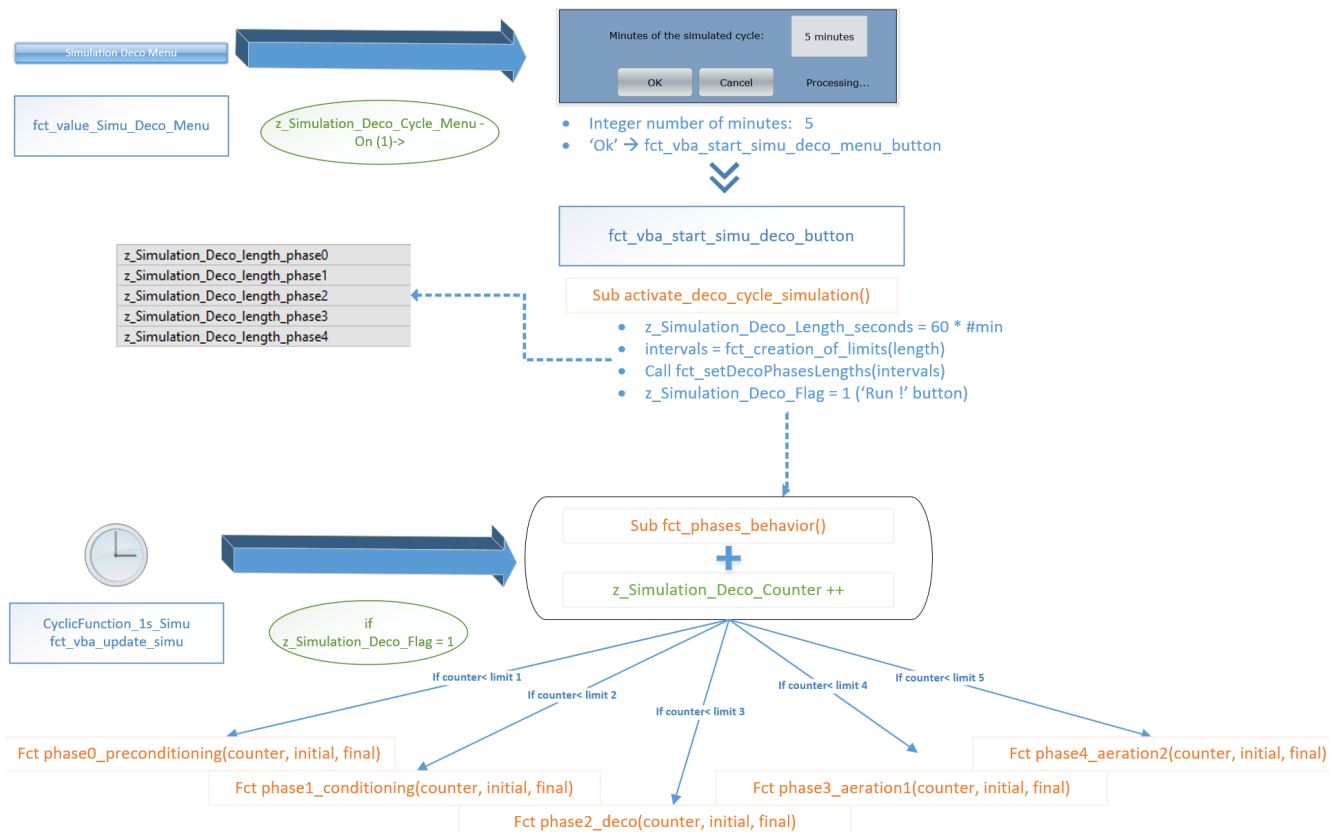


Figure 10.2: Functional schema of the simulation of the deco-cycle

## 10.2.2 How can we use it?

It is recommended to use it only when it is needed as it will take a few time to set up the simulation conditions, and a little fewer to set them back.

Anyhow, all the variables, functions, editor scripts, runtime scripts, frames and screens can be loaded as they are saved as *.XML* files. If we want to import these files **the order is important**. Follow the order to import the files:

1. Variables
2. Frames
3. Screens
4. Editor scripts
5. RT scripts
6. Functions
7. Reaction matrix
8. Time control



## Driver and PLC variables

The PLC driver should be set to *Mode: Simulation Static*. This will allow to change the variables linked to PLC addresses without losing the configuration. Once the simulation is not needed anymore the original state. Check Fig. 10.3 for more details.

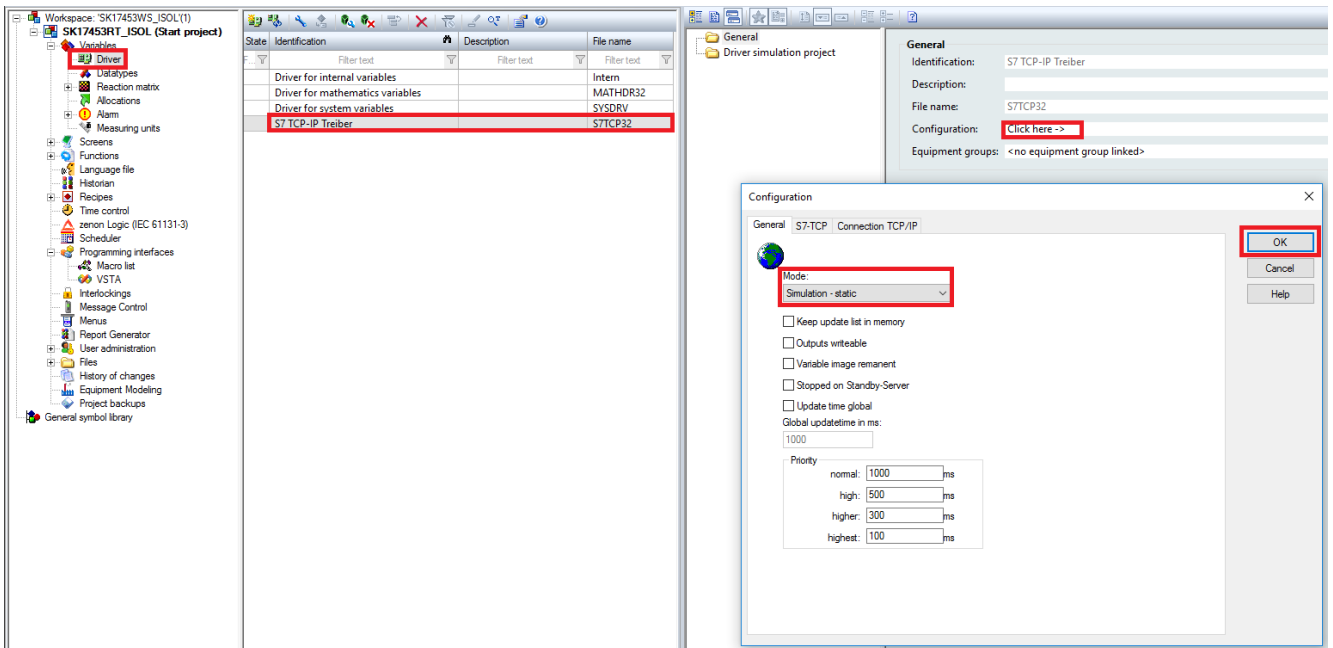


Figure 10.3: Changing the driver for simulation

## Scripts (Editor Mode)

It is important to set 'Write set value' property to 1 in order to be able to change their values via script. If records in the CEL want to be avoided due to changes in these variables also remove them using the properties Fig. 10.4.

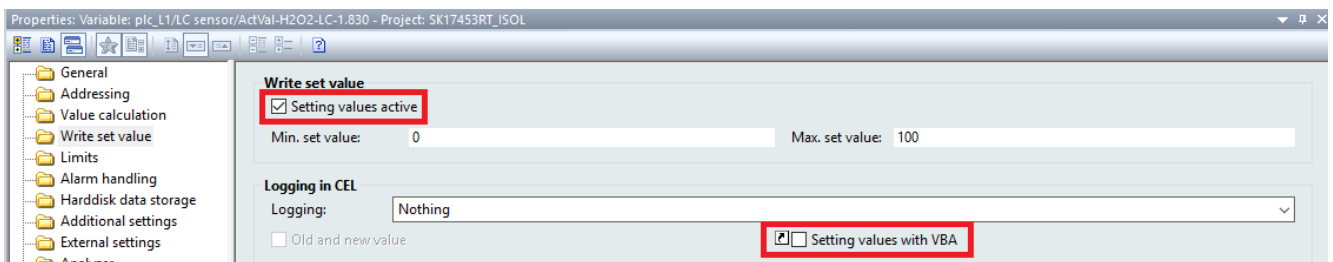


Figure 10.4: Internal variables should be read constantly

To do it manually it can take some time. We wrote a script that could be run from the editor to avoid repeating this operation as many times as variables we would like to simulate. The list of variables to simulate should be introduced in this script.

```

Sub fct_allowToWriteVariablesForDecoSimulation(ByVal obElement As Element)
  Dim varVariableNamesList As Variant
  Dim varString As Variant

  '
  'ALLOW TO WRITE THE FOLLOWING VARIABLES
  '
  '-----

  Set myProject = obElement.Parent.Parent.Parent.Parent

  varVariableNamesList = Array( _
  "plc_L1/SIS/ActVal_Vap01-1.815", _
  "plc_L1/Sensor/ActVal_Hum-1.810", _
  "plc_L1/ActVal_Deco/ActualDosing", _
  "plc_L1/ActVal_Deco/CumulatedDosing", _
  "plc_L1/Sensor/ActVal_dPressure-1.840", _
  "plc_L1/Sensor/ActVal_HC-1.832", _
  "plc_L1/LC sensor/ActVal-H2O2-LC-1.830", _
  "plc_L1/Line/ModePhase" _
  )
  For Each VarName In varVariableNamesList
    myProject.Variables.Item(CStr(VarName)).DynProperties("InOut") = True
    myProject.Variables.Item(CStr(VarName)).DynProperties("SV_VBA") = False

  Next VarName
End Sub

```

Figure 10.5: Script in the editor to allow to write and read the variables

After finishing all the test with the simulation tool, the original variables' properties should be set as in the beginning. Script *fct\_forbideToWriteVariablesForDecoSimulation* rolls back these properties. It is important to use the same variable list as it was used for setting the variables to writable.

## Scripts (RT)

The scripts used in subroutines and functions are summarized in the table Tab. 10.1:

Table 10.1: Table with functions / subroutines in RT

Ref.	Function/Subroutine	Name	Triggered by	Parameters	Return	Description
1	Subroutine	activate_deco_cycle_simulation	Positive Edge	z.Simulation_Deco_Length_minutes > 0	-	Create limits for each phase Set the simulation flag to 1
2	Function	fct_creation_of_limits	Call from 1	z.Simulation_Deco_Length_seconds > 0	Intervals	Create interval objects for each phase Set the total length in seconds for each phase
3	Subroutine	fct_setDecoPhaesLengths	Call from 1	Intervals	-	Save the length of each interval in variables
4	Subroutine	fct_phases_behavior	Reaction Matrix (counter.value) counter.value > -1	z.Simulation_Deco_Counter	-	Read value of the counter and limits for each phase Check current phase & call the phase function
5	Subroutine	phase0_preconditioning	Call from 4	Counter, initial, end	-	Get the values for variables in phase 0
6	Subroutine	phase1_conditioning	Call from 4	Counter, initial, end	-	Get the values for variables in phase 1
7	Subroutine	phase2_deco	Call from 4	Counter, initial, end	-	Get the values for variables in phase 2
8	Subroutine	phase3_aeration1	Call from 4	Counter, initial, end	-	Get the values for variables in phase 3
9	Subroutine	phase4_aeration2	Call from 4	Counter, initial, end	-	Get the values for variables in phase 4
10	Subroutine	fct_update_counter	Time control: Each second CyclicFunction_Is_Simu	-	-	If flag = 1 , increase counter If counter = length, counter = 0, flag = 0

## Internal variables

Several internal variables should be created in order to perform the simulation. Check Fig. 10.6 to understand the types and units of each variable.

z_Simulation_Deco_Counter	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_Cycle_Menu	simu		0	0	0	0	0	0	Intern - Driver for internal vari...	BOOL	0
z_Simulation_Deco_Debug_Msg	simu		0	0	0	0	0	0	Intern - Driver for internal vari...	STRING	0
z_Simulation_Deco_Flag	simu		0	0	0	0	0	0	Intern - Driver for internal vari...	BOOL	0
z_Simulation_Deco_Length_minutes	simu	minutes	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_length_phase0	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_length_phase1	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_length_phase2	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_length_phase3	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_length_phase4	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0
z_Simulation_Deco_Length_seconds	simu	seconds	0	0	0	0	0	0	Intern - Driver for internal vari...	INT	0

Figure 10.6: Internal variables to create and configure for deco-cycle simulation

It is important to be able to read all of them all the time. Otherwise the information will be lost when they are not shown in the screen (Fig. 10.7)

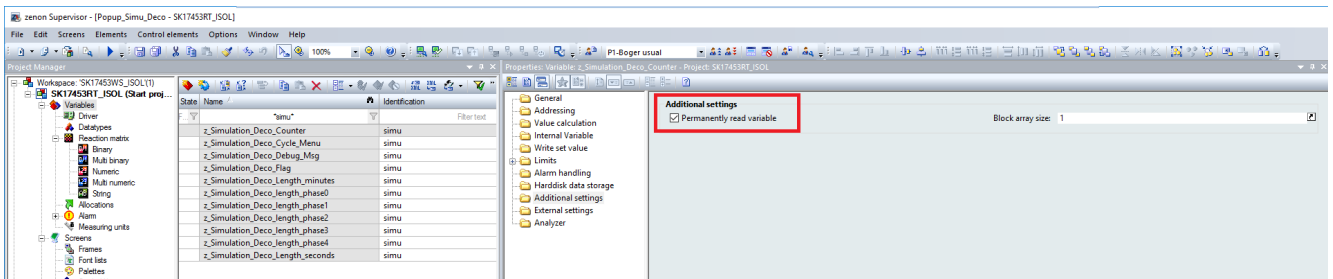


Figure 10.7: Property to allow the variables to be readable all the time

## Pop-up screen

A pop-up screen will be triggered by a button that sets *z\_Simulation\_Deco\_Cycle\_Menu* to 1. This pop-up screen has a writable window to set the number of minutes of the whole cycle, the length of each period, a dynamic text window to check at which moment of the simulation we are and controls to exit the window. Fig. 10.8

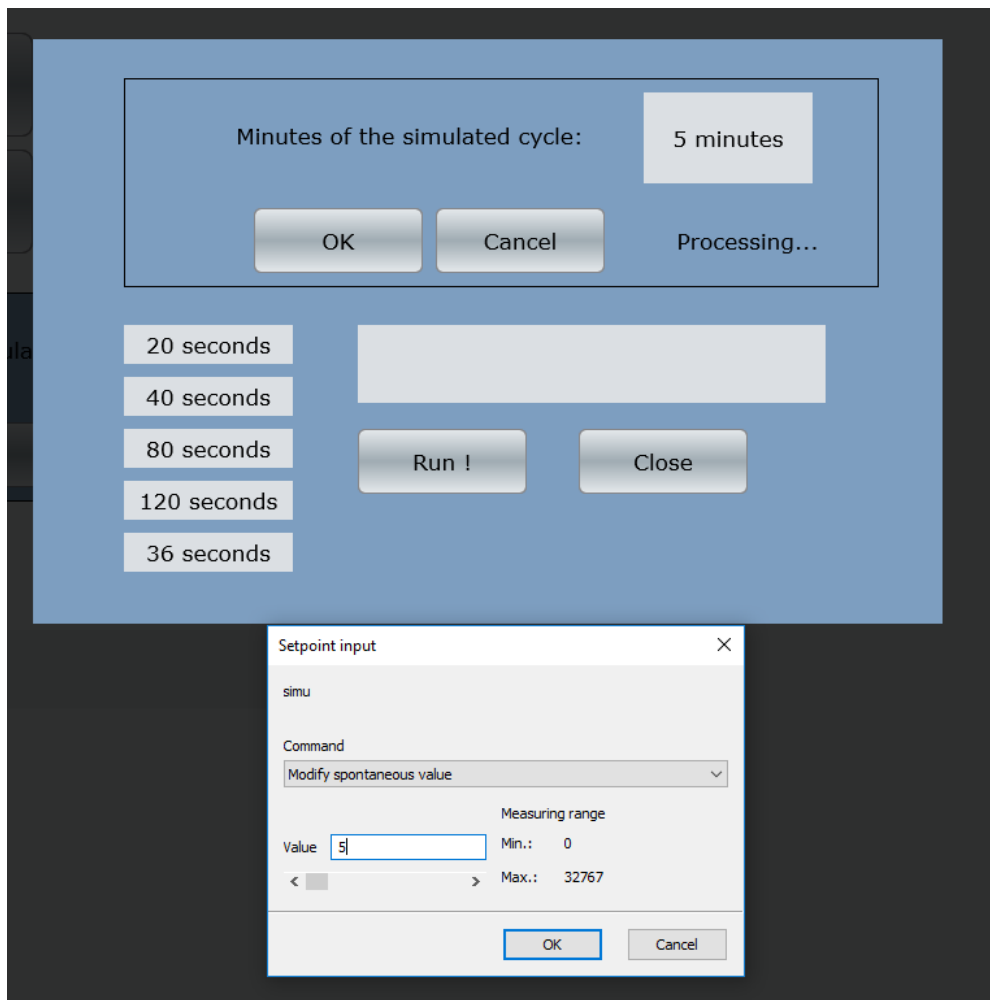


Figure 10.8: Pop-up screen and time selection window

## Functions

The functions needed to use the deco-simulation tool are listed in Fig.10.9.

fct_vba_select_simu_behaviour	Execute VBA macro	fct_phases_behavior
fct_vba_update_simu_counter	Execute VBA macro	fct_update_counter
fct_vba_start_simu_deco_menu_button	Execute VBA macro	activate_deco_cycle_simulation
fct_sc_Popup_SimuDeco-open	Screen switch	Popup_Simu_Deco (Standard)
fct_run_simu_deco	Write set value	z_Simulation_Deco_Flag - On (1)->
fct_value_Simu_Deco_Menu	Write set value	z_Simulation_Deco_Cycle_Menu - On (1)->

Figure 10.9: Functions needed to use the simulation tool

The chronological order to use the functions would be as follows:

1. **fct\_sc\_Popup\_SimuDeco\_opens** It opens the popup menu to set up the time. Should be linked to a button created specifically for the simulation.
2. **fct\_start\_simu\_deco\_menu\_button** After changing the number of minutes expected for

the simulation we click *Ok* button. It will be linked to this function that will change the length in seconds for each phase and store it in an internal variable.

3. **fct\_run\_simu\_deco** Linked to the button *Run !*. This will start the simulation itself by setting a flag to one.
  - (a) **fct\_vba\_update\_simu\_counter** This function is called every second but only increase the counter and call the next function if the flag is active.
  - (b) **fct\_vba\_select\_simu\_behaviour** Depending on the length of each phase and on counter's value a different function can be called. Check the diagram [10.2](#)