RIBER SOFT

CRYSTAL XE

Developer's manual

Date (jj/mm/yyyy)	Rev	Description
22/09/2022	5	New tag type for recipe: Enumeration ; How to add ramps in a template ; update screenshots
11/04/2022	4	Added Float 32 BE type
28/09/2016	3	Added a case study in the ASCI protocol section
20/06/2016	2	Added more information for modbus read and write tags and add tasks priority section.
06/04/2016	1	Creation

CONTENTS

1 PREFACE

1.1 Audience

The Riber Crystal XE software is a very versatile and expandable environment for the control of MBE, CBE and similar growth systems.
This guide is intended for developers who want to customize Crystal XE software to communicate with new devices and/or develop new equipment (template library). Users who want to customize the user view or other part of the software should refer to the *Crystal XE User Manual*
Users who want to improve their knowledge of the Pascal scripting language should also refer to the *Crystal XE User Manual*.

1.2 Documentation accessibility

This document and its latest version can be downloaded on the Crystal XE website at :
http://www.crystalxe.com in the section My CrystalXE

You will need to create an account to access to My CrystalXE. This account will be then activated by the Riber team. Once this administrative process has been completed, you will be able to access the relevant information, as shown in the figure below. Registering will also provide you with the option of being kept informed of developments and upgrades to the software, and to other Riber systems.

2 Presentation

2.1 Getting started with customizing devices and equipment

This section provides an overview of customizing Crystal XE templates library and describes the tools and resources that are available for customization. This is a rich programming environment that has already been used successfully to interface with a wide range of growth and measurement instruments and equipment.
Development of this software will continue, and the user is encouraged to sign up for automatic notification of software releases, or to check the web site for updates and new software releases.
Feedback on the software design and features is welcome within the user community.

2.2 Terminology

Firstly, the terms used in the design environment should be defined, as these will be used throughout this document.
Riber identifies the driver to communicate with the device as a **Device** module (or **Regulator**), Such a regulator or device can be represented graphically in the system.

Figure 1: Example of graphical representation of a device (lakeshore 211)

The virtual instrument or product is known as Equipment, and can be composed of one or several **sub-equipment(s).** Again, these are clearly shown in the system representation.

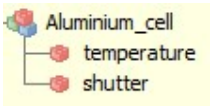


Figure 2: Example of an effusion cell (Aluminum cell) comprising of temperature and shutter sub-equipments

2.3 Available Views of the System

The **User view** is the area that user can customize to represent his system graphically. An example is shown below. It is selected by clicking on the 'UserView' tab at the top of the main window.

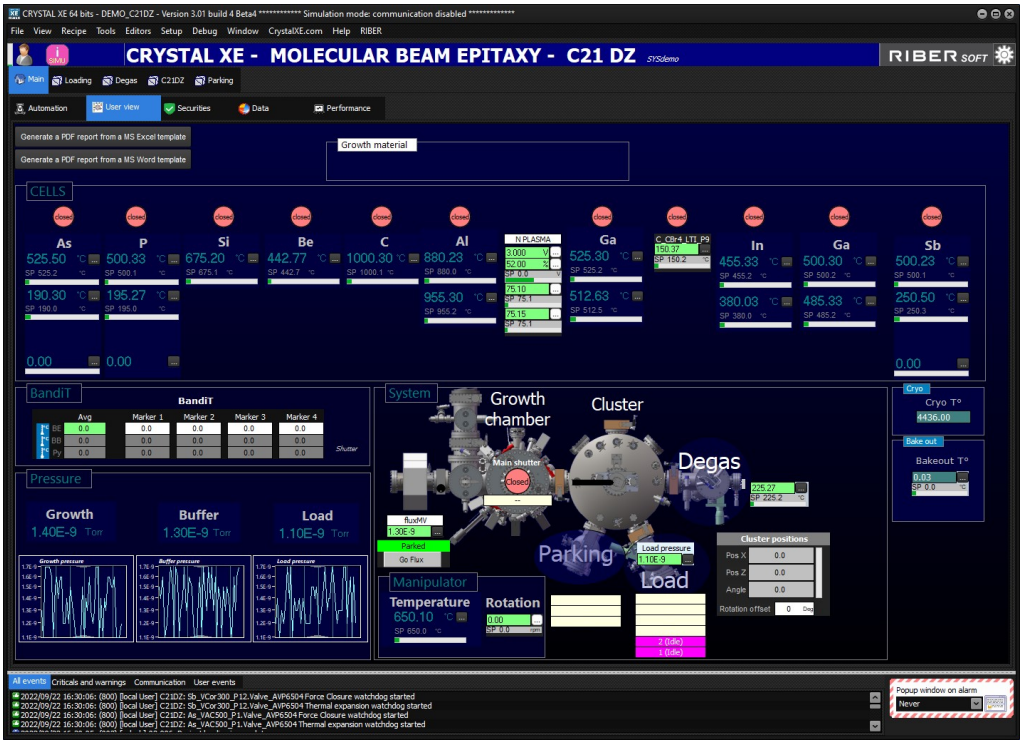


Figure 3: Example of the main user view

The **Equipment view** is separated in two parts:

- The list view or **list HMI**
- The detail view or **detail HMI**

This is selected by clicking on the 'Equipment View' tab bellow a chamber tab. As can be seen in the figures below, the two sections are shown side by side, with the detail view on the left of the main window.

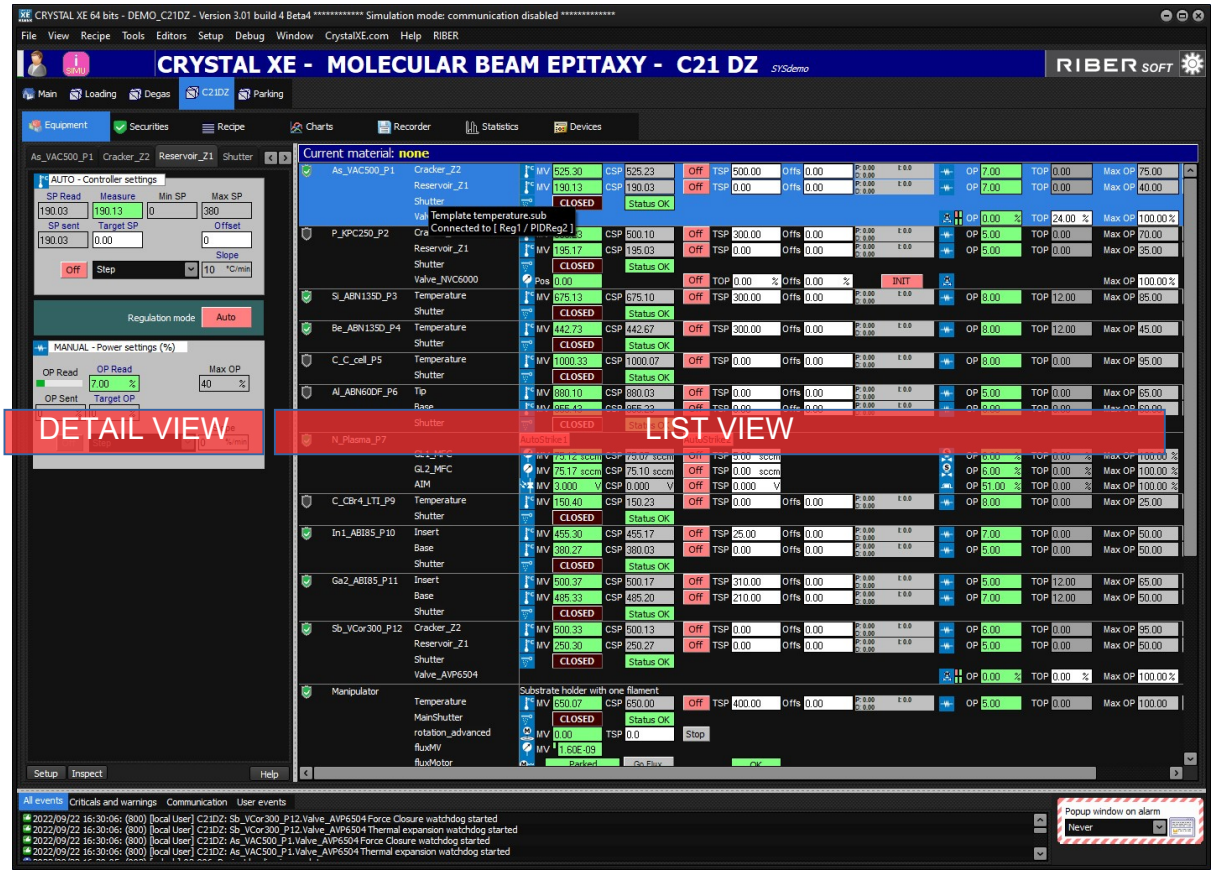


Figure 4: Equipment View

Setup HMI

The setup HMI is the window which opens on clicking on the setup button of the sub-equipment or Equipment, as shown in the figure below.

Figure 5: The setup HMI screen

2.4 File organization and file types

With Crystal XE, two main directories need to be considered:
- **The program directory**. It contains all the files needed for Crystal XE software, including executable files, templates etc.
- **The project directory** (which should **not** be in the program directory), contains all files needed for your application (configuration files, recipes, forms, recorded data etc..)

Example of disk architecture:

C:\riber\CrystalXE contains the **program** directory
D:\riber\<project_name> contains the **project** directory

The project can also be installed in the c:\riber directory like this: *c:\riber\<project_name>*

Templates file:

In this document, you will see how to create new device templates (regulators), new equipment templates and sub-equipment templates. Crystal XE is provided with a palette (library) of predefined templates. Each template is a separate file, and can be used directly or as the basis for a customized solution.

File extension of templates:

There are three kinds of template which are identified by their file extension.
- Extension of devices (regulator) is **.RGR**
- Extension of equipment is **.EQU**
- Extension of sub-equipment is **.SUB**

Location of templates:

Templates which are provided with Crystal XE are located in the directory:
<program directory of Crystal XE> \ templates\ e.g. C:\riber\templates\.

Devices templates are in *<Crystal XE program directory> \ templates\regulator*
Equipment templates are located in *<Crystal XE program directory> \ templates\equipment*
Sub-equipment templates are located in *<Crystal XE program directory> \ templates\equipment\sub-equipment*

IMPORTANT:
If you are creating your own templates, you should place them in **<project directory>\ *templates***
If not then when updating the software, your own templates won't be reachable.

File types:

All templates are XML files and the encoding must be UTF-8. An example is shown below for information.
These files should not be modified directly. You should use one of the template editors (see next section), as these are designed for the task and will ensure consistency of data.

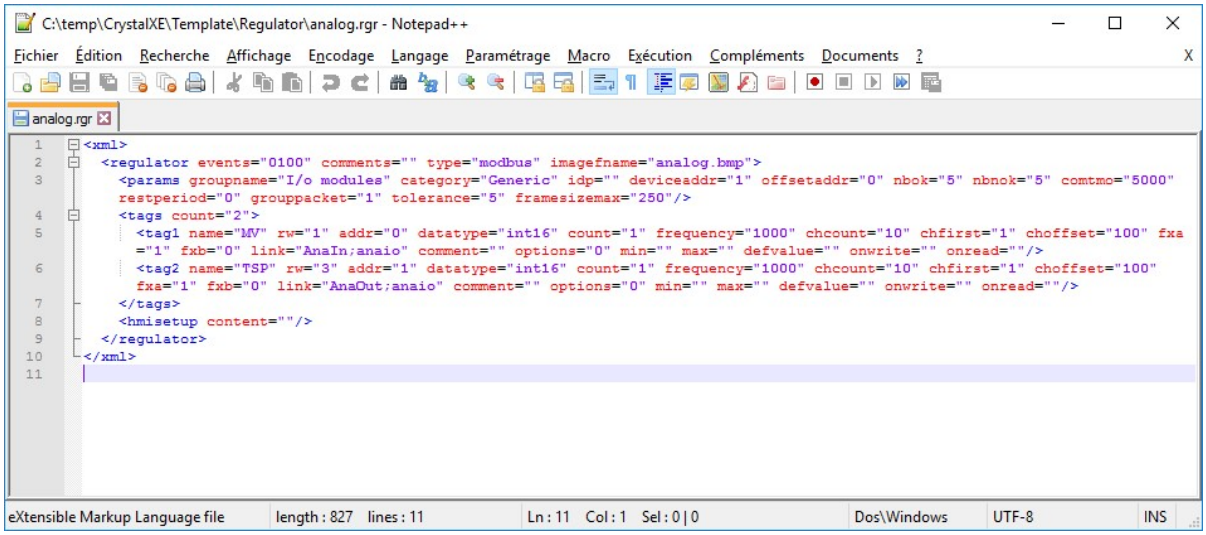


Figure 6: Example of device template analog.rgr

2.5 Templates editors - overview

Crystal XE is provided with an editor for each type of template:
- Modbus device editor
- ASCII device editor
- Equipment editor
- Sub-equipment editor

These editors can be opened in the hardware configuration windows.

Click on the top right button on the main screen to open the window.

Figure 7: Hardware Configuration button

Note: you can also access the hardware configuration through the **setup menu** then Hardware configuration.

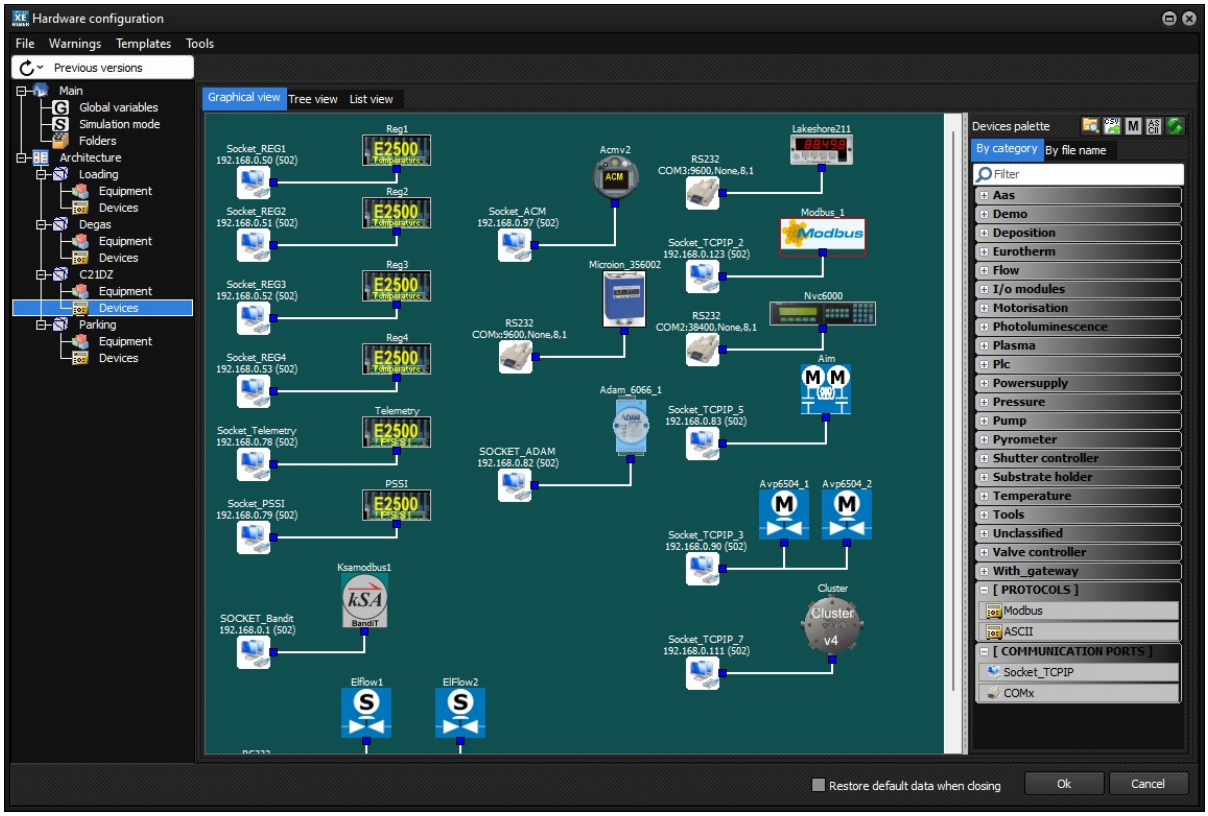


Figure 8: Example of project configuration screen

All the chambers in the system are listed in the "Architecture."
You can add/remove or rename chambers at your convenience.
All chambers include two sub-elements which are Equipment and Devices.

2.5.1 Modbus device template editor

To open the modbus device template editor, click on the device item of a chamber, move the modbus protocol module on the Graphic view area (drag and drop). Then double click on this module or right click and select Edit in the popup menu.

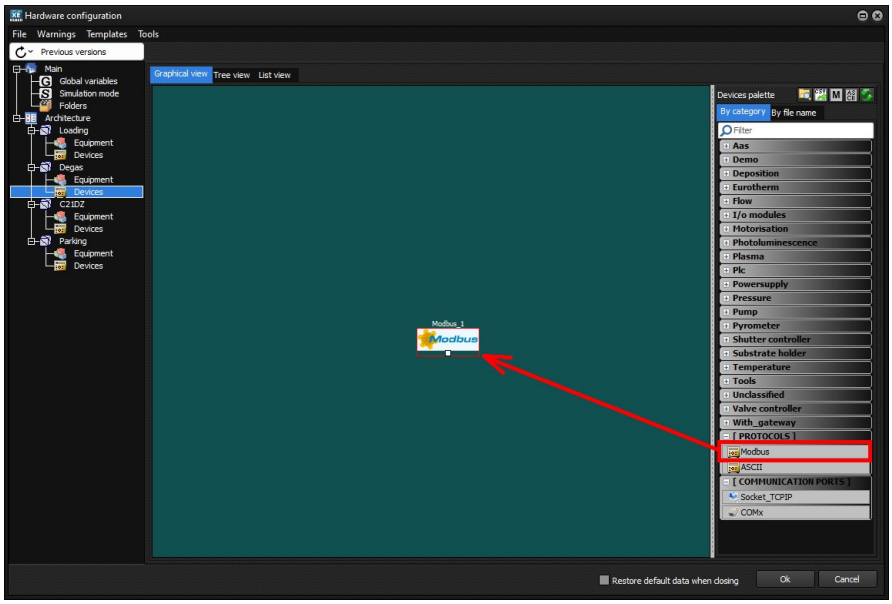


Figure 9: Drag and drop modbus module

The other approach is to edit an existing template.

For example, in the Device palette (the column on the right) open the category Eurotherm (click on “+” to open the category) and right click on the device E3508_v330. Then select “Edit template : E3508_v330.rgr”

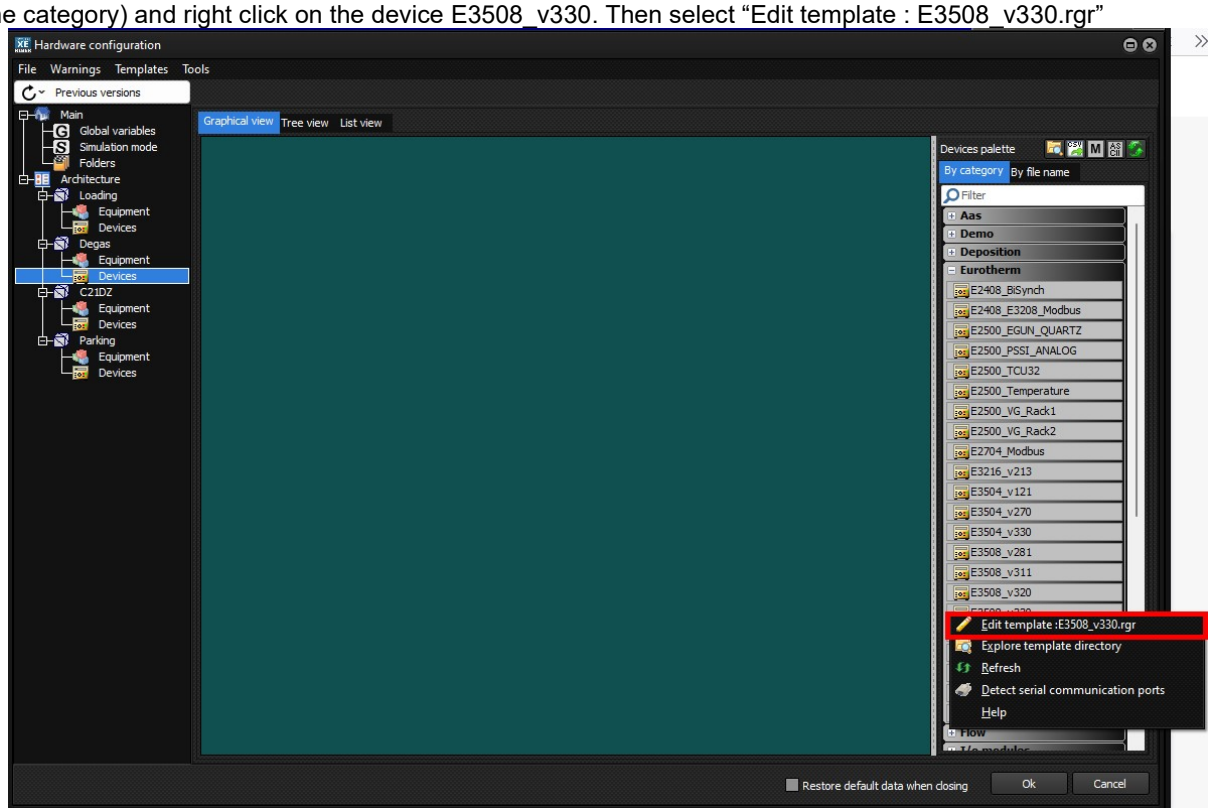


Figure 10: This picture shows how to edit an existing template

→ See later in this document for more details about the modbus template editor.

2.5.2 ASCII device template editor

To open the ASCII device template editor, click on the device item of a chamber, move the ASCII protocol module on the Graphic view area (drag and drop). Double click on this module or right click to display to popup menu and select Edit Template.

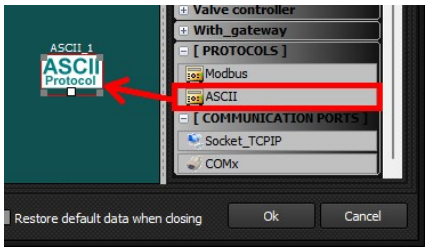


Figure 11: Drag and drop ASCII module

The other approach is to edit an existing template.
For example, in the Device palette (The column on the right) open the category Thickness (click on “+” to open the category) and right click on the device STM_2_Inficon. Then select “Edit template : STM_2_Inficon”

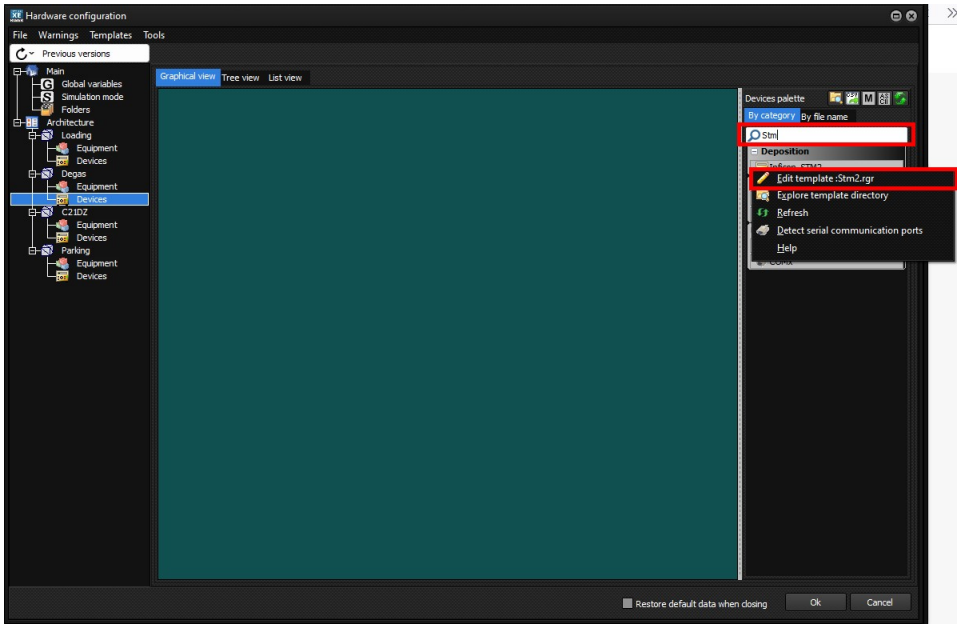


Figure 12: This picture shows how to edit an existing template by searching its name, then right click on it.

→ See later in this document for more details about the ASCII template editor.

2.5.3 Equipment template editor

To open the Equipment template editor, click on the equipment tag of a chamber and you will then see buttons in the tool bar of the Equipment palette. Click on the button highlighted in the figure below (a the top right).

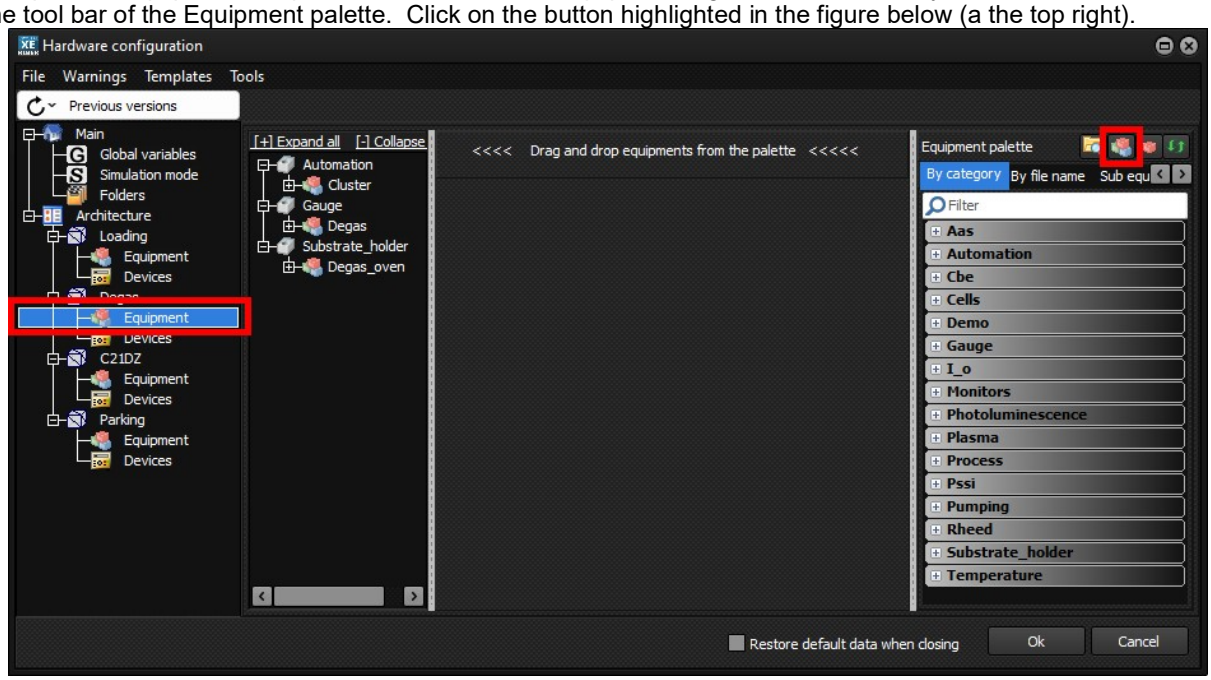


Figure 13: Opening the equipment template editor

Edit a template in a category of the palette:
For example, in the Equipment palette open the Cells category (click on “+” to open the category) and right click on the item “EffusionCell | SimpleFilament”. Then select “Edit template: EffusionCell | SimpleFilament”

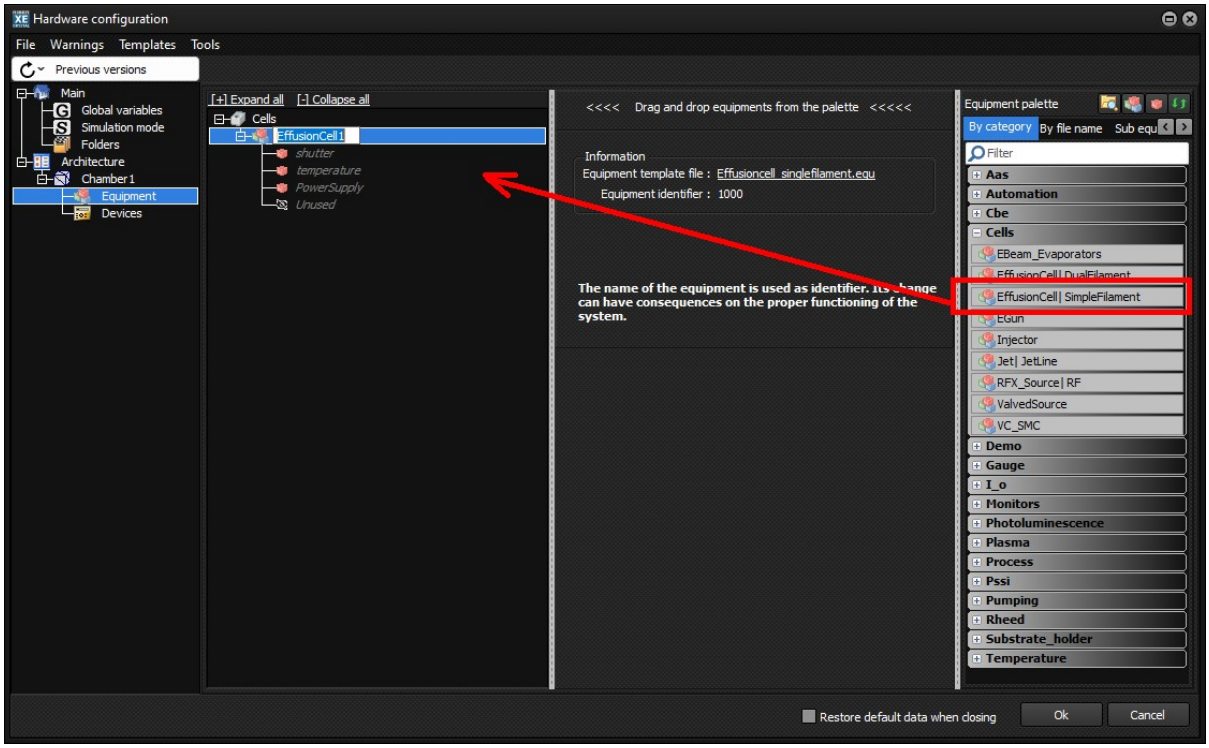


Figure 14: Drag and drop equipment

Edit a template in the tab "By file name" of the palette:
For example, click on the tab "By file name" in the palette, right click on an item and select "Edit template…"

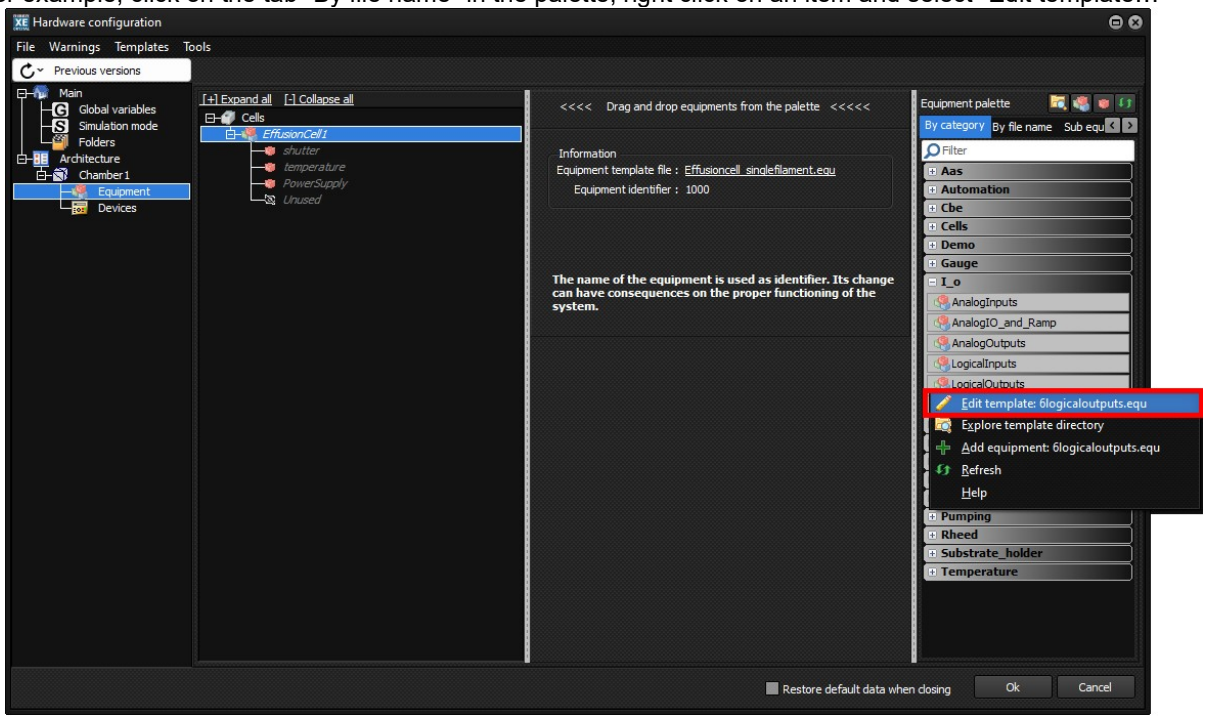


Figure 15: Edit equipment by filename

Other approach:
In order to edit a template which is already in the tree view, you can click on the link as indicated in this picture:

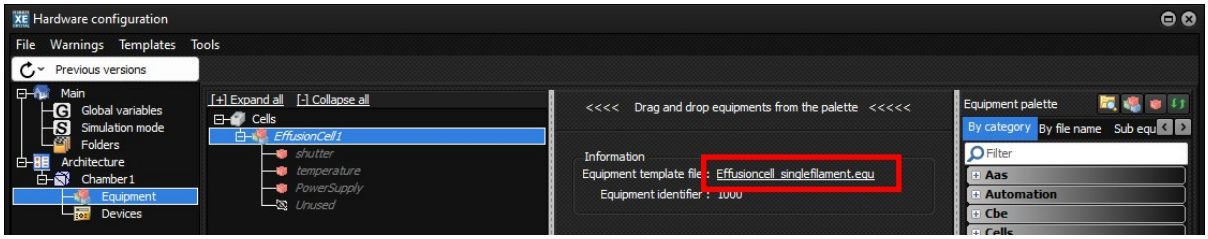


Figure 16: Opening the equipment template editor

→ See later in this document for more details about the equipment template editor.

2.5.4 Sub-equipment template editor

To open the sub-equipment template editor, click on the equipment item of a chamber and you will see buttons in the tool bar of the Equipment palette.

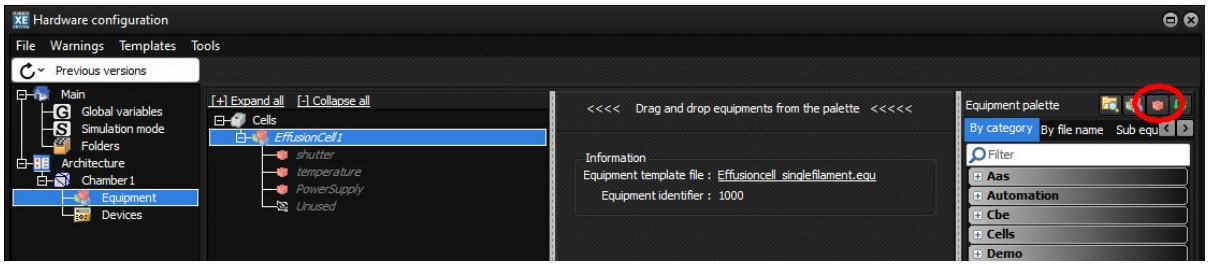


Figure 17: Opening the sub-equipment template editor

Edit a template in the tab "Sub-equipment" of the palette:
For example, click on the tab "Sub-equipment" in the palette, right click on an item and select "Edit template…"

Alternative Access to a sub-equipment template:
In order to edit a template which is already in the tree view, you can click on the link as indicated in this picture:

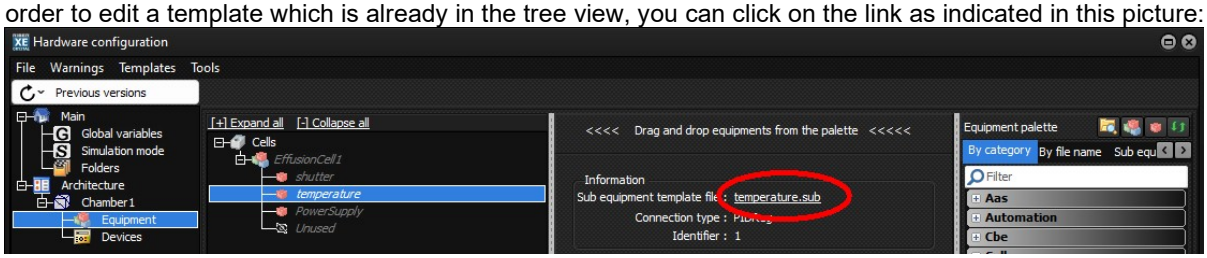


Figure 18: Alternative way to access sub-equipment template

→ See later in this document for more details about the sub-equipment template editor.

2.6 Communication protocols

Crystal XE can communicate with devices in **Modbus protocol** or **ASCII protocol**.

2.6.1 Modbus protocol

The Modbus protocol is described in full at http://modbus.org/docs/PI_MBUS_300.pdf
The following functions are used by Crystal XE:

For writing:
- function 5 : write single coils.
- function 15 : write multiple coils.
- function 16 : write multiple registers.

For reading:
- function 1: read coil.
- function 2: read input register.
- function 3: read multiple registers.

For each Modbus device, a memory of 65536 words of 16 bits is allocated. This is shared between external (equipment) Modbus registers and internal memory locations. Note that internal memory locations need to be allocated a specific address within this memory space.
It is possible to read or write directly in this memory by using the script functions ReadTable, WriteTable and derived functions. See the Pascal functions reference section in the user manual for more details. You can also access on-line help by pressing the F1 key in the script editor at any time.

When a tag is updated (when a read frame answer is received from a device), the memory is automatically updated. But when using the function WriteTable, the value is not sent to the device. To send the value to the device, use the functions beginning by WriteTag, with the appropriate parameters set

2.6.2 ASCII / Eibisync / binary protocol

The ASCII module can be configured to communicate with **EibiSync** devices or other ASCII protocols.
Using the scripting facility, the ASCII module can also be used to communicate in **binary protocol**.

3 System Architecture

A system is composed of different equipment.
For example, equipment can be a cell, a manipulator (substrate holder), a pressure gauge, a motorized flux gauge, a gas source system, a pyrometer etc.
Equipment have specific functions and one or more HMI (human man interfaces).

Equipment is composed of one or several sub-equipment, and each sub-equipment can communicate with only one electronic device.

3.1 Rules

Tags
A tag is the smallest item which belongs to a device or a sub-equipment or equipment.
A tag is like a variable in a programming language.
A tag contains a value or is a shortcut to another tag and has a type (int16, float etc..)
A tag can be read and/or written from anywhere in Crystal XE using the full path (for example: C21DZ.In1_ABI85_P10.Insert.MV, or only the tag name (MV) if it is used inside the template itself)

Devices module rules
Each device module (regulator) communicates with only one physical device or with only one modbus memory map (for the modbus protocol).

Each device (regulator) may have several channels. By example, the Eurotherm E2500_Temperature has 8 channels of temperature regulator.
Only tags of used channels will communicate with the device (regulator). Channels become 'used' when they are linked to sub-equipment at the system level. Unused channels do not become active and do not generate bus traffic.

Equipment rules
Equipment is composed of one or several sub-equipments.
For example, a cell can be composed of on temperature sub-equipment and one shutter sub-equipment:

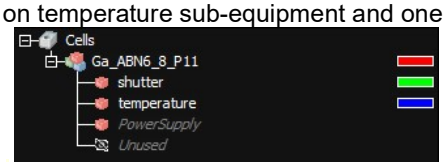


Figure 19: Equipment structure with sub-equipment

Sub-equipment rules
Each sub-equipment can be linked to only one device and channel. The channel to be allocated to the sub-equipment can be selected when it is linked with an equipment.

3.2 Links

This section is important and your attention is requested.
This section explains how the links between the devices and the sub-equipment are implemented.

When the user prepares a configuration, he must define a link between all sub-equipment and the devices (regulators).

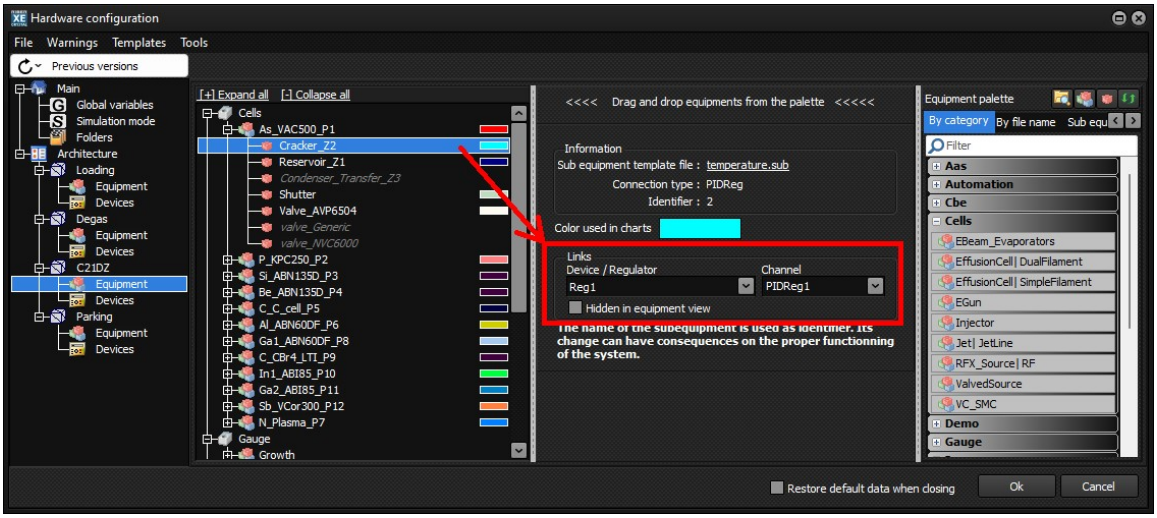


Figure 20: Example of a link between the sub-equipment temperature.sub, the device Reg1 and channel PIDReg1

When you develop a sub-equipment you must define a link type list. This list will define the devices with which the sub-equipment can be associated.
Several types of link can be defined for one sub-equipment.

In the device side, each tag owns a link type.

#	Name	R/W	Address	Data type	Quantity	Frequency	ChCount	ChFirst	ChOffset	FxA	FxB	Link Type	Comment	Opt
7	MV	Read	32770	float 32 bits	1	1s	8	1	1024	1	0	PIDReg	Measured value	-----

Figure 21: Example of link type for the device E3508_v330.rgr is PIDReg (in the modbus editor)

→ For example: the link type defined for the sub-equipment *temperature.sub* is **PIDReg** and the link type defined for the tags of a Eurotherm temperature regulator *E3508_v330* is also **PIDReg**, then this sub-equipment is compatible and can be linked to this device.

One device can be compatible with several sub-equipment, as all tags of the device can be given different link types.
For example, if the link type of the tag of a device is “bitIn” and the link type of another tag of the same device is “bitout” then this device can be associated with all sub-equipment which are compatible with “bitin” and “bitout”.

3.2.1 Links map

All the links between the device and the sub-equipment can be displayed in the menu of the window hardware configuration by clicking on **Tools/Link map** (or *connexion* depending of the version you are running).

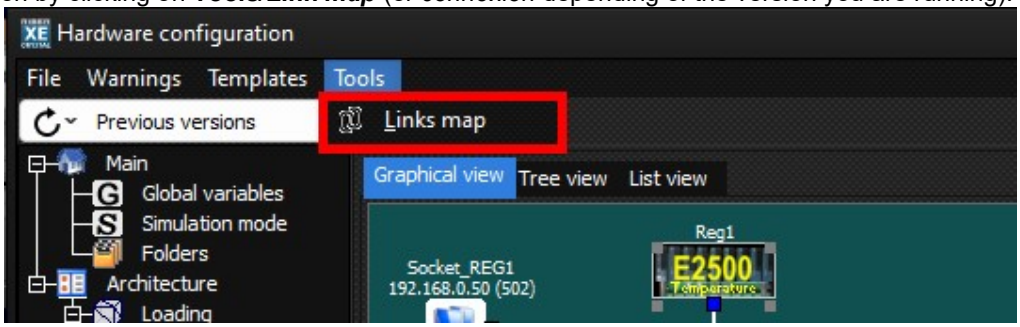


Figure 22: Selection of the links map

The following map window is then displayed:

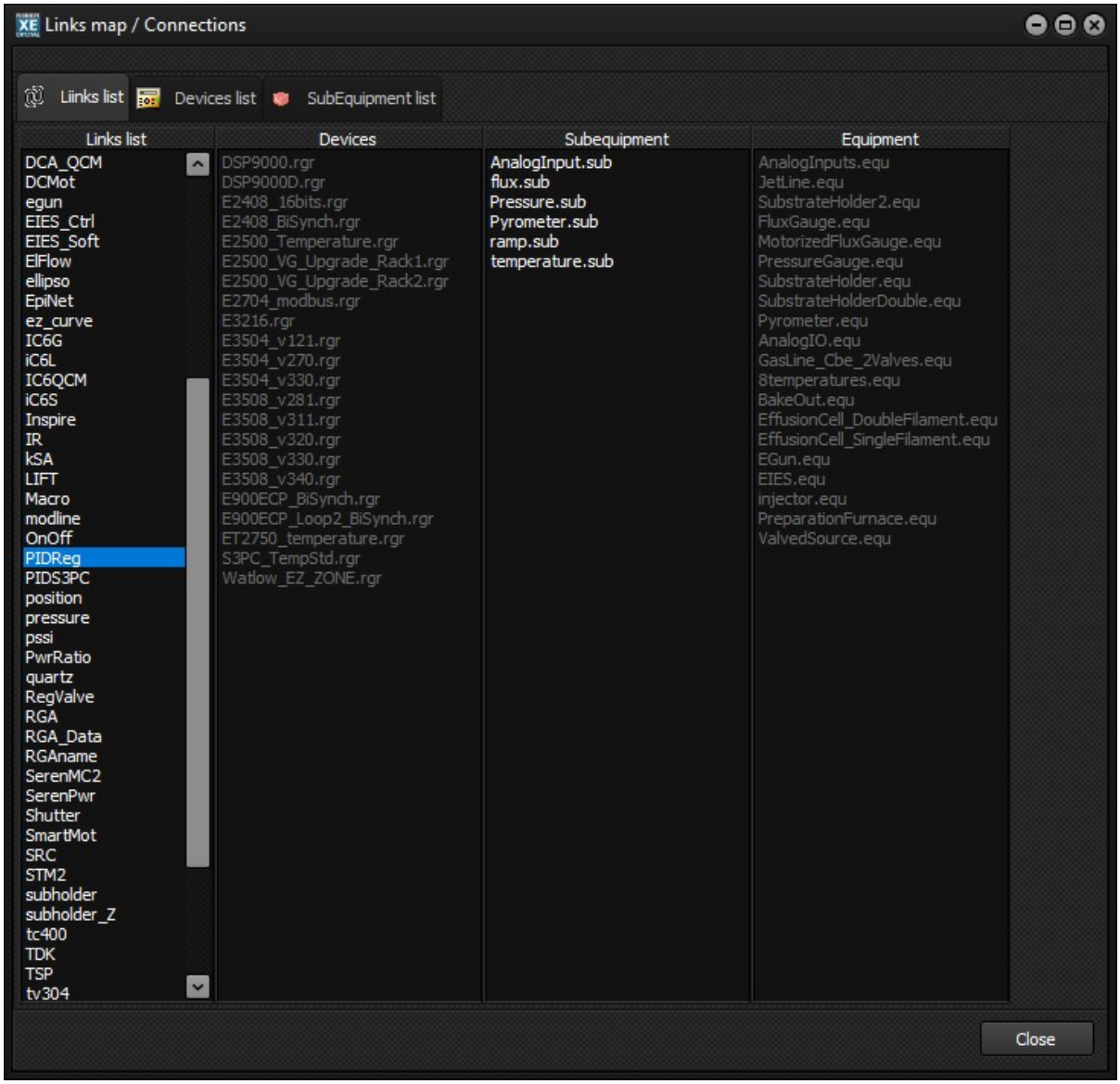


Figure 23: Links map window

On clicking the PIDReg option in the Links List:

- All the devices compatible with this link are displayed in the device column.
- All sub-equipment compatible with this link are displayed in the sub-equipment column.
- All equipment compatible with this link are displayed in the equipment column.

Similarly, if you click on a sub-equipment, all the equipment which use this sub-equipment are displayed in the Equipment column.

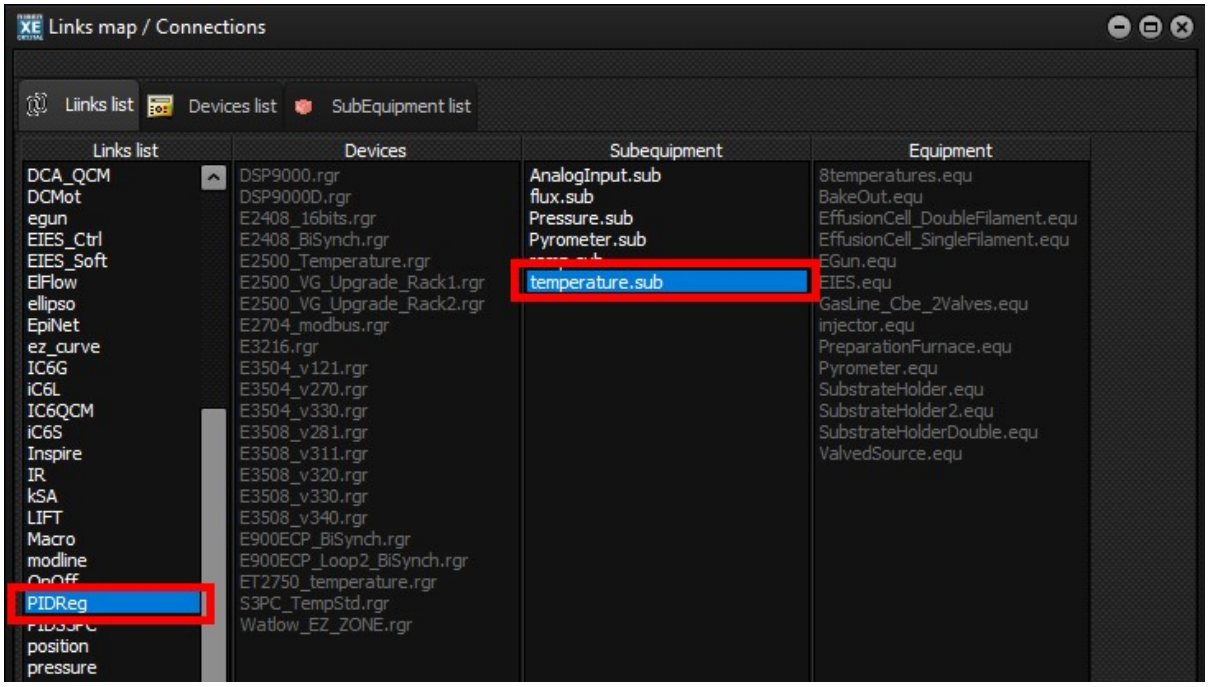


Figure 24: Click on the tab Devices list to display all devices found in the device template directory

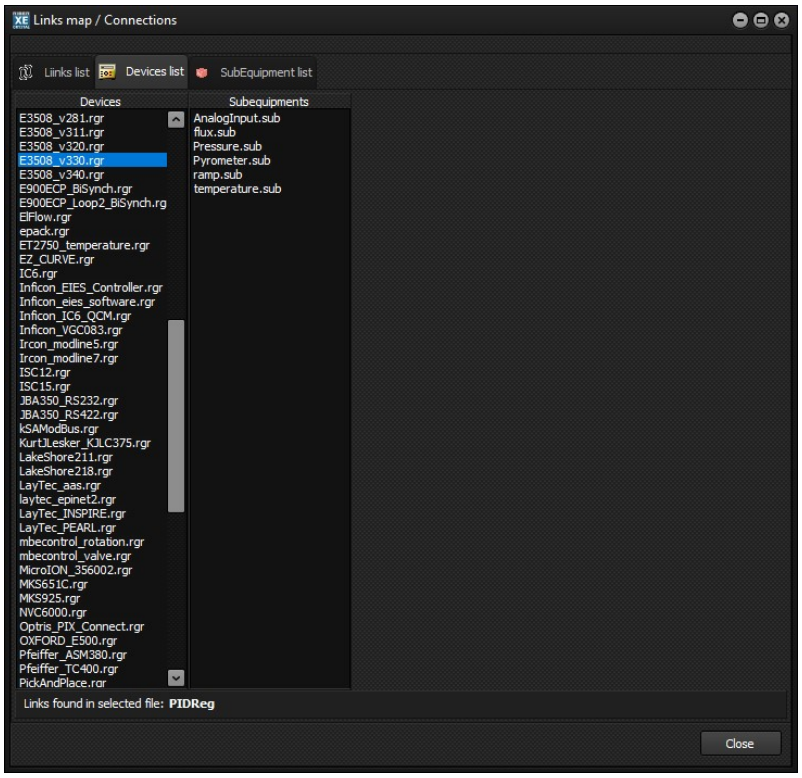


Figure 25: Display of sub-equipment compatible wiht a device

When you click on a device, all sub-equipment compatible with this device is displayed in the Sub-equipment column. The list of link type is also displayed in the bottom.

Click on the tab Sub-equipment list to display the list of Device and Equipment which are compatible with this sub-equipment.

3.3 Devices graphical view

3.3.1 Auto update modules

Generally, when you prepare a configuration, you can use predefined templates, following the process defined below:

Click on the *device* item of the tree view and select the tab *Graphical view* (see below).

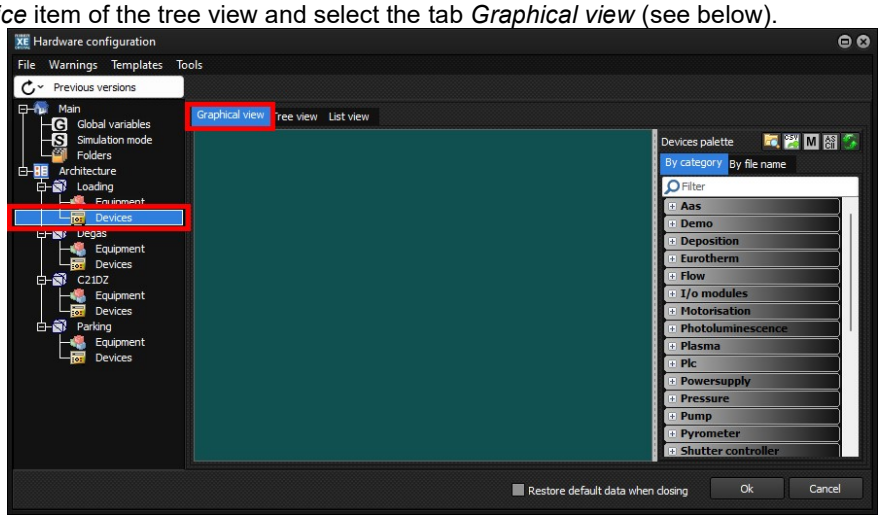


Figure 26: Graphical view selection

Drag and drop a socket and a device module on the graphical area.

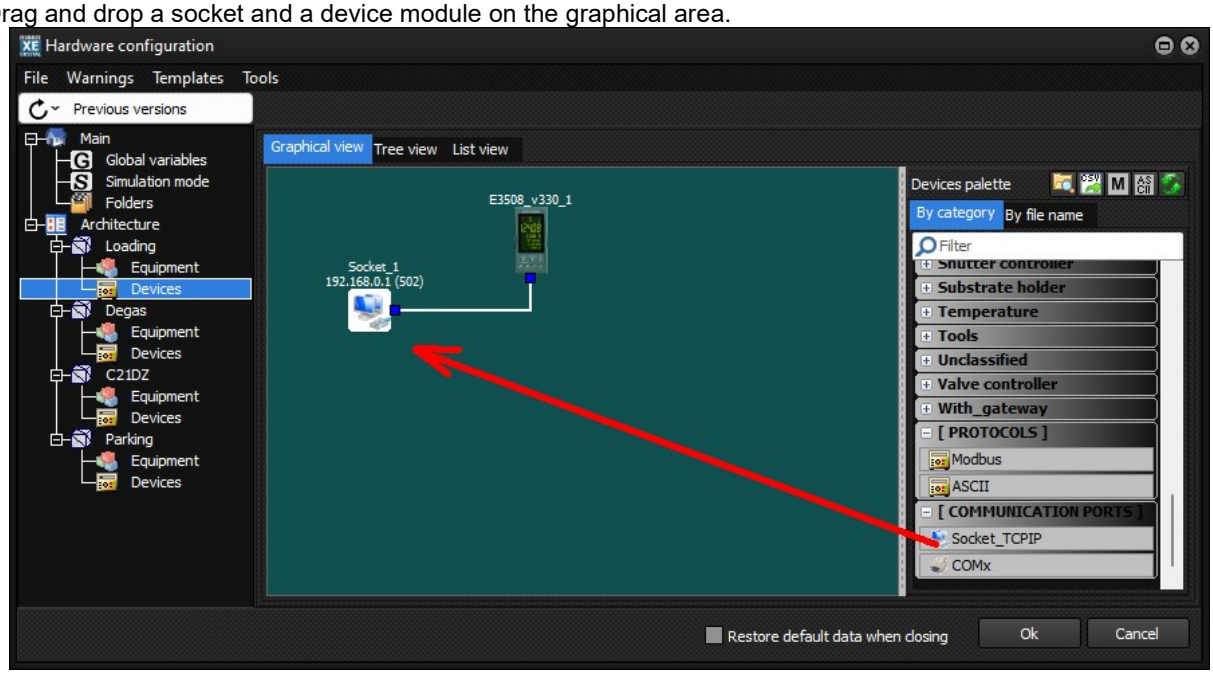


Figure 27: Drag and drop a socket and device module

Note that if you update the template file E3508_v330.rgr (e.g. when installing a new version of Crystal) then all modules present in the graphical view will be also updated.

3.3.2 Custom modules

It is also possible to customise a device module.
This is particularly useful if the modification is specific to the system and will never be reused on another system.

To customize a module, double click on it (or right click and select *Edit*)
The modbus or the ASCII editor will open.
Click on the advanced tab and then check the box Customized regulator

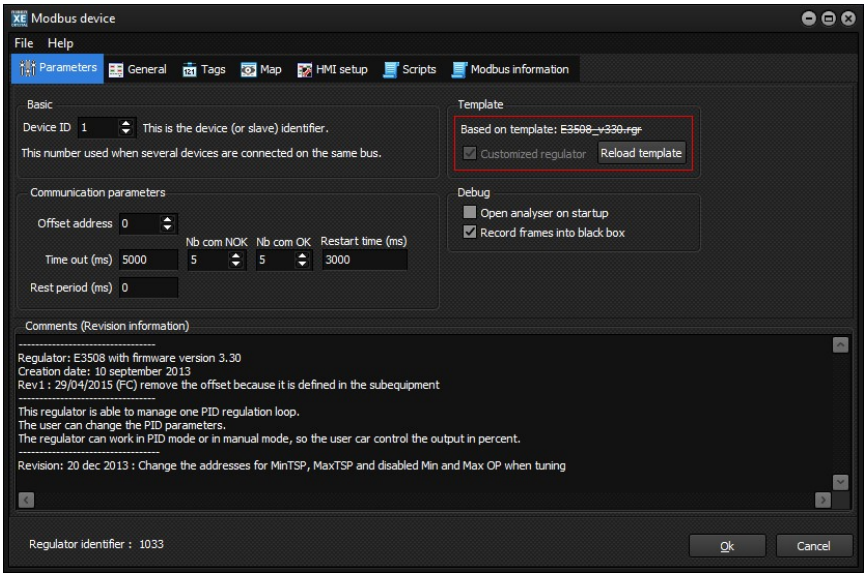


Figure 28: Customization of a module

Now the module appears on the screen with a red rectangle:

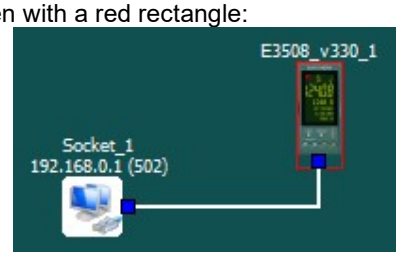


Figure 29: red rectangle indicating customized module

This means the module is not linked to the template. All modification made in the template will have no effect on this module (user beware!)

3.3.3 Create you own modules as a template

If you need to use your module several times in your configuration, or can see uses on other systems, you should create your own template to add it in the device palette.

Two files are needed to create a device template. The device itself is a file name of your choice with the extension .RGR and the bitmap is an image file that represents the device with the extension .BMP. **You must save the device file (.rgr) and image file (.bmp) into the template directory of your project NOT into the template of the program directory. If you do not do this, they will disappear from the current program directory when you update Crystal XE.**

Create a bitmap with paint or another image editor and save it into the project template/regulator directory.

Then, open the device editor (modbus or ASCII editor) and define a Folder and a Name (see below).

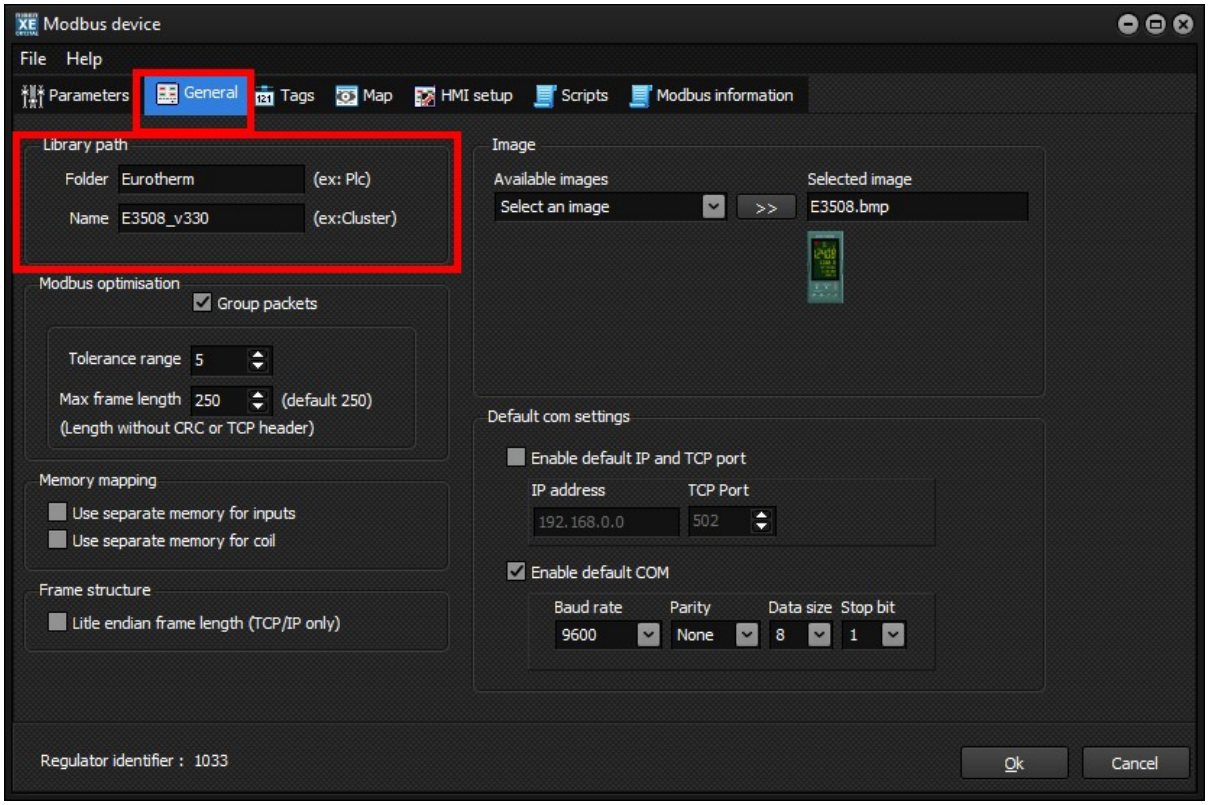


Figure 30: Preparing a custom device

The folder and the name will be used to add the device into to device palette (see next picture):

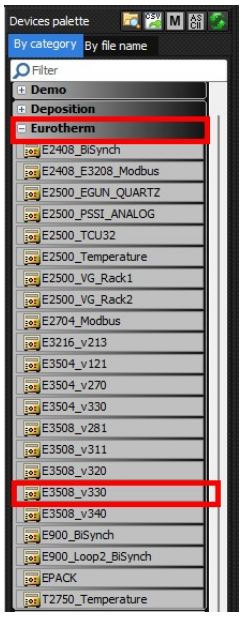


Figure 31: Updated device palette

Save your device template:
To save the file, go in the Modbus (or ASCII) editor and in the menu, select ***File / Save as…*** and save the file into the directory <project directory>/Template/Regulator

Then update the device palette by pressing the button 
Now you can drag and drop your template into the graphical view.

4 EDITORS

4.1 Devices editors

4.1.1 Modbus module

To create a new device module using modbus protocol, you may either begin from an empty page or copy an existing device and modify it.

To begin a new device module using modbus protocol from an empty page, click on this button

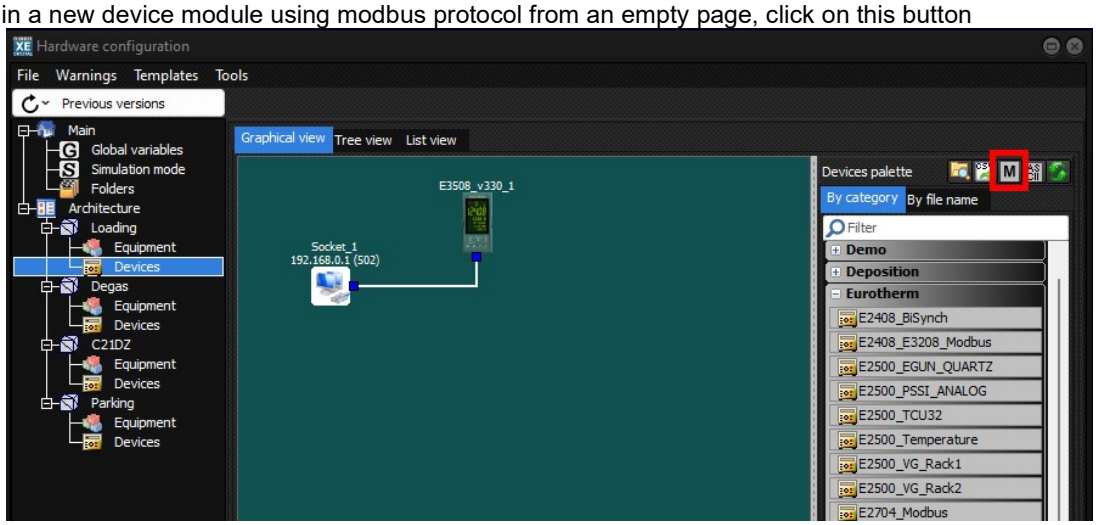


Figure 32: Starting a new modbus module

This will open the modbus editor (see below):

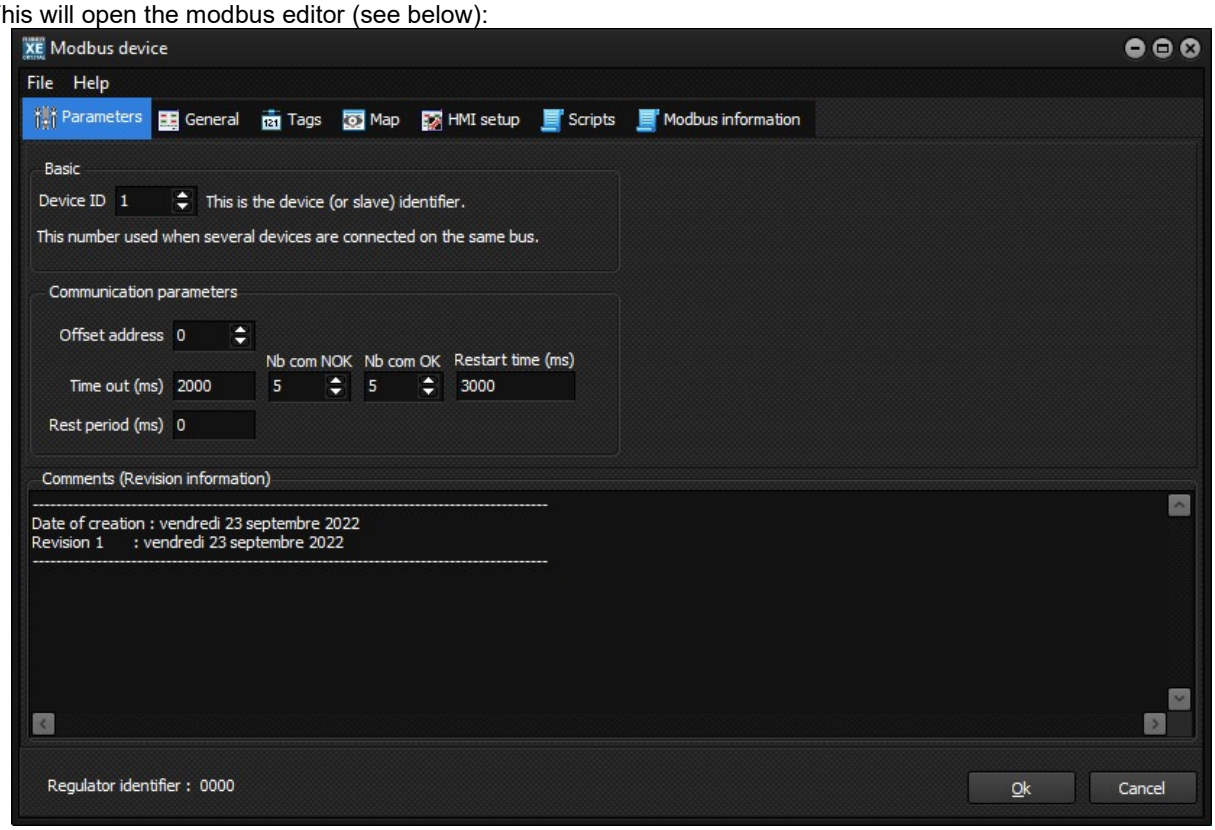


Figure 33: Modbus editor

Device ID

Figure 34: Device ID definition

This is the device identifier used by the modbus protocol. Depending of the device, this identifier can be programmed by a dipswitch on the device or by software. See the Modbus protocol document for further details as necessary http://modbus.org/docs/PI_MBUS_300.pdf.

Communication parameters

Figure 35: Communication parameters

Offset address is used to offset all the addresses define in the TAB Tags, should this be required.

Time out: If no answer is received after this time for a reading or a writing request then the request will be send again to the device. If the number of retries (defined in the Nb com NOK box) is reached then a timeout will occur. A critical event will raised and after a delay Crystal XE will try again. Then after a time out, if the device responds several times (as defined in the Nb com OK box) then the communication fault flag will be removed.

Rest period: this value defines the time to wait before each frame. This is in addition to the inter-message gap that is defined as part of the Modbus protocol. Frees up time for the device to process the last message. By default, this value is not necessary (leave it at zero).

Debug group

Figure 36: Debug box

Check box "Open analyzer on startup": if enabled, then the analyzer of this module will be opened when Crystal XE will open or when exiting the hardware configuration by pushing the OK button.

Check box "Record frames into black box": if enable and if the black box is enabled in the options window, then all frames of this module will be store in the black box files (see the user manual for more details).

Comments

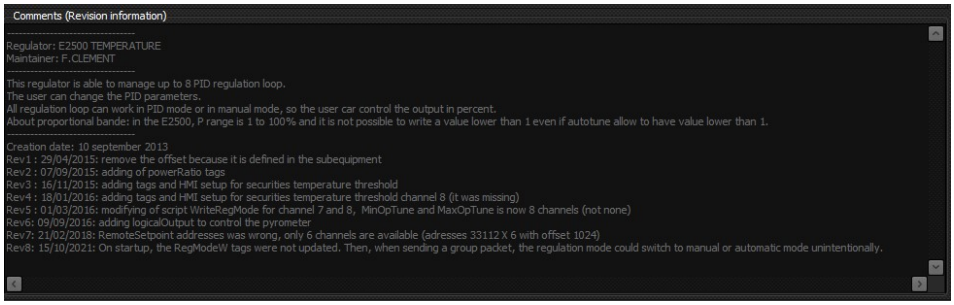


Figure 37: Comment section

This is a free text area where the user generally indicates the versions and evolutions of the template.

TAB *General*

Library path:

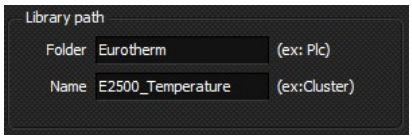


Figure 38: Library path for module

For information about this box, see the section "***Create your own module as a template***"

Modbus optimization / Group packets

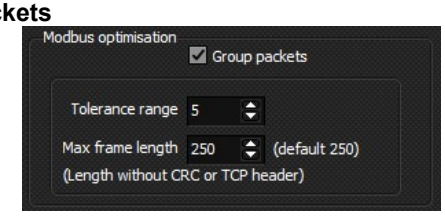


Figure 39: Modbus optimization box

When checked, this option allows optimizing the number of frames by regrouping the packets. You can define a tolerance range value for the addresses (by default 5) and a maximum frame length (by default 250)

For example, if the register at address 6 must be refreshed and the register at address 9 must be also refreshed then only one frame will be sent to the device to read multiple registers from address 6 to 9. But if this option is not enabled then one frame will be sent to the device to read the register at address 6 and another one to read the register at address 9.

So if the tolerance range is 5 and the registers 6,9,12,14,19,20,26,29,32 must be refreshed then only two frames will be sent:

- One to read multiple registers from 6 to 20 (=group of 15 registers)
- One to read the registers from 26 to 32 (=group of 7 registers)

(Frames are separate because the gap between 20 and 26 exceed the tolerance range or 5 registers)
In all case the frame size cannot exceed the Max frame length which has been defined.

Memory mapping

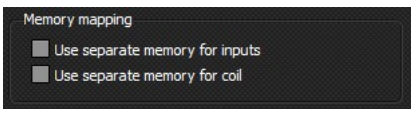


Figure 40: Memory mapping

If these boxes are not checked, the internal memory space used for holding registers will be the same as for inputs or coils. Check the box "Use separate memory for inputs" or "Uses separate memory for coil" to distinguish the memory spaces.
The script functions for writing and reading the modbus table only allow access to the holding registers. When these boxes are checked, it is not possible to access the memory space by the script functions.

Frame structure

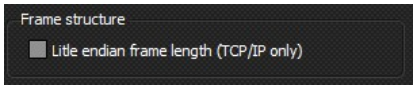


Figure 41: Frame structure

Unchecked by default.
The TCP header is :
TxData[0] = Transaction identifier MSB
TxData[1] = Transaction identifier LSB
TxData[2] = Protocol identifier MSB
TxData[3] = Protocol identifier LSB
TxData[4] = length MSB
TxData[5] = length LSB
The frame length occupy two bytes in the TCP frame (MSB+LSB).
Check this box only to swap the bytes if needed (LSB+MSB).

Image

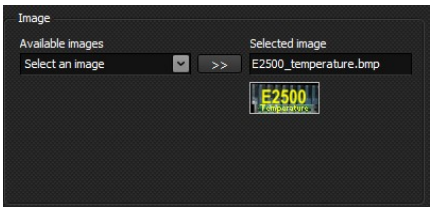


Figure 42: Edit the icon representation

Please select from the list, the image corresponding to the device. This image is used in the graphic view "Devices". If the image does not appear in the list, you can add a new bitmap file to the template\regulator sub folder.

Default COM settings

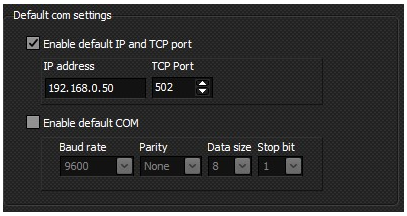
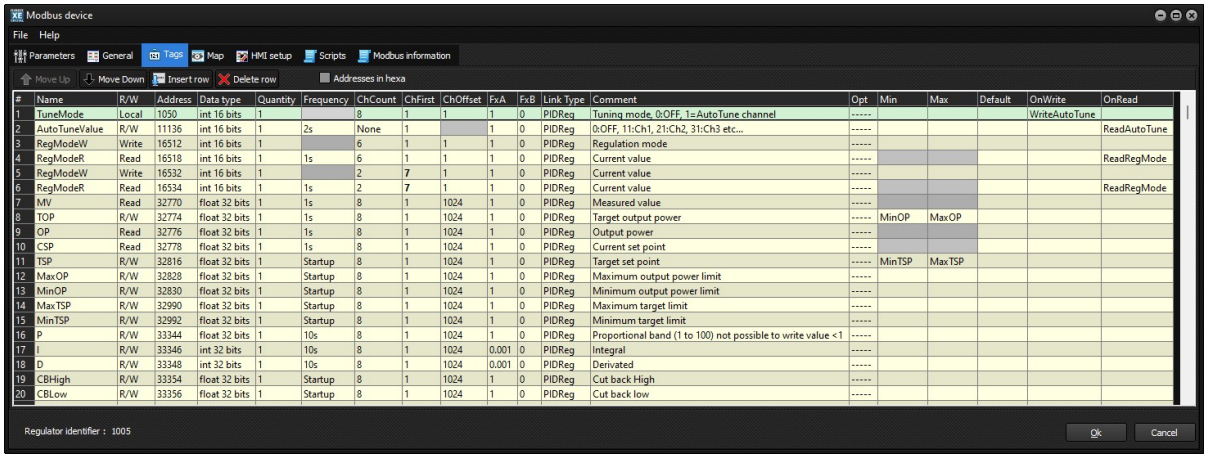


Figure 43: Edit default communication parameters

This depends on the communication mode of the device. Check or uncheck the boxes corresponding to the media supported by the device (TCP/IP Socket and Serial Port)
For each media, you can enter the default values that will be proposed to the user when creating the hardware configuration.

TAB *Tags*



#	Name	R/W	Address	Data type	Quantity	Frequency	ChCount	ChFirst	ChOffset	FxA	FxB	Link Type	Comment	Opt	Min	Max	Default	OnWrite	OnRead
1	TuneMode	Local	1050	int 16 bits	1		8	1	1	1	0	PIDReg	Tuning mode, 0:OFF, 1=AutoTune channel	-----				WriteAutoTune	
2	AutoTuneValue	R/W	11136	int 16 bits	1	2s	None	1		1	0	PIDReg	0:OFF, 11:Ch1, 21:Ch2, 31:Ch3 etc...	-----					ReadAutoTune
3	RegModeW	Write	16512	int 16 bits	1		6	1	1	1	0	PIDReg	Regulation mode	-----					
4	RegModeR	Read	16518	int 16 bits	1	1s	6	1	1	1	0	PIDReg	Current value	-----					ReadRegMode
5	RegModeW	Write	16532	int 16 bits	1		2	7	1	1	0	PIDReg	Current value	-----					
6	RegModeR	Read	16534	int 16 bits	1	1s	2	7	1	1	0	PIDReg	Current value	-----					ReadRegMode
7	MV	Read	32770	float 32 bits	1	1s	8	1	1024	1	0	PIDReg	Measured value	-----					
8	TOP	R/W	32774	float 32 bits	1	1s	8	1	1024	1	0	PIDReg	Target output power	-----	MinOP	MaxOP			
9	OP	Read	32776	float 32 bits	1	1s	8	1	1024	1	0	PIDReg	Output power	-----					
10	CSP	Read	32778	float 32 bits	1	1s	8	1	1024	1	0	PIDReg	Current set point	-----					
11	TSP	R/W	32816	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Target set point	-----	MinTSP	MaxTSP			
12	MaxOP	R/W	32828	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Maximum output power limit	-----					
13	MinOP	R/W	32830	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Minimum output power limit	-----					
14	MaxTSP	R/W	32990	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Maximum target limit	-----					
15	MinTSP	R/W	32992	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Minimum target limit	-----					
16	P	R/W	33344	float 32 bits	1	10s	8	1	1024	1	0	PIDReg	Proportional band (1 to 100) not possible to write value <1	-----					
17	I	R/W	33346	int 32 bits	1	10s	8	1	1024	0.001	0	PIDReg	Integral	-----					
18	D	R/W	33348	int 32 bits	1	10s	8	1	1024	0.001	0	PIDReg	Derivated	-----					
19	CBHigh	R/W	33354	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Cut back High	-----					
20	CBLow	R/W	33356	float 32 bits	1	Startup	8	1	1024	1	0	PIDReg	Cut back low	-----					

Figure 44: Tab tags in editor

Name

The name of the tag which may be used also by the sub-equipment to identify the tag. Allowed characters are letters, numbers and underline, no spaces, no special characters.
The name must not start with a number.

R/W

Tags may generate periodically Modbus frames which are sent automatically to the device (Tags of type Read, Write and R/W). If you don't want to send frames automatically, use a local Tag. In that case, you can define a script to custom the frame to send to the device.

None	Tag is disabled. Scripts onRead and onWrite will be never executed (for debug only)
Read	A Modbus command will be sent periodically to the device with the frequency defined in the column *Frequency*. Can only be read. It is not possible to change the value by using the script or HMI objects. A write frame will never be sent to the device.
Write	A Modbus command will be sent to the device when this tag will be written. Can be read and written by the script or HMI objects but only write frames are sent to the device.
R/W	A Modbus command will be sent periodically to the device with the frequency defined in the column *Frequency* and a Modbus command will be also sent to the device when this tag will be written.
Local	No read or write frame will be sent to the device. This tag is only used as a variable. Take care to use a free Address i.e. do not overlap with the address of one of the other types of tags

Table 1: R/W column in TAB tags

Address

This is the address of the register or the coil or the input depending of the **Data type**.
IMPORTANT: register, coil or input uses the same memory space. Then you cannot use the same address for a coil and a register (or input) If the device has input and holding register at the same address you need to use two modules connected to the same communication port.

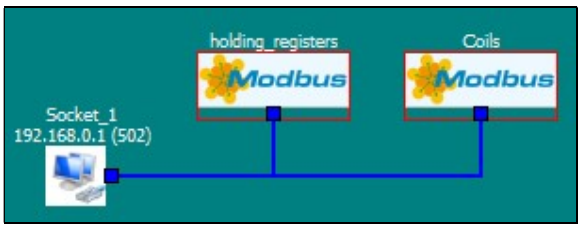


Figure 45: example of two modules on the same socket

You can display the address in decimal or hexadecimal by checking the option in the tool bar.

Data type

1 bit	For input or coil depending of the parameter of the R/W column. - If R/W column = **Read** then it is an **input** and the modbus **function 2** will be used. - If R/W column = **R/W** then it is **a coil** and the modbus **function 1** will be used for reading(read coil) and **function 5** (single) or **15** (multiple) for writing. - If R/W column = **Write** then it is **a coil** and the modbus **function 5** (single) or **15** (multiple) will be used for writing.
Int 16 bits	Holding register used as an integer of 16 bits (1 word of 16 bits is read/write each time)
Int 32 bits	Holding register used as an integer of 32 bits (2 word of 16 bits is read/write each time)
Int 64 bits	Holding register used as an integer of 64 bits (4 word of 16 bits is read/write each time)
Uint 16 bits	Holding register used as an unsigned integer of 16 bits (1 word of 16 bits is read/write each time)
Uint 32 bits	Holding register used as an unsigned integer of 32 bits (2 word of 16 bits is read/write each time)
Uint 64 bits	Holding register used as an unsigned integer of 64 bits (4 word of 16 bits is read/write each time)
Float 32 bits	Modbus itself does not define a floating point data type but it is widely accepted that it implements 32-bit floating point data using the IEEE-754 standard. This implementation settled on using two consecutive 16-bit registers to represent 32 bits of data or essentially 4 bytes of data. It is within these 4 bytes of data that single-precision floating point data can be encoded into a Modbus RTU message. For example the float number 1.123 will be store : - 1st register = 0x3F8F - 2nd register = 0xBE77 Other example, the float 1.23E-5 will be store: - 1st register = 0x374E - 2nd register = 0x5C19
Float 32 BE	BE for Big endian: the words are exchanged like this ABcd ==> cdAB

Table 2: Data Type column in TAB tags

Quantity

Default: 1.
Do confuse quantity and ChCount.
Quantity defines the number of memory locations occupied depending of datatype. For example, if the datatype is int 32 bits and quantity is 3 then 6 registers will be used for this tag and then 12 bytes will be read/written to the device. If you then define three channels of this tag (for example), then 18 registers will be defined.

Frequency

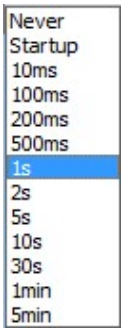


Table 3: List of available frequencies for update

This defines the minimum refreshing time of the tag.
This time is not guaranteed as it depends of the communication flow.

Select ***Never*** if you don't want Crystal send reading request for this tag. This does not prohibit sending a writing frame when writing to this tag, as defined in a script or request a read manually as part of a script.

Select ***Startup*** to refresh the tag only when Crystal is loading. Again, it can be refreshed as a script action.

ChCount, Chfirst and ChOffset

These columns define the channels.

ChCount is the number of channels, in other term, the number of times that this tag is repeated. For example, it you communicate with a Temperature regulator which has 2 PIDs with exactly the same tags, then select 2 for the ChCount, 1 for the ChFirst (if the first channel is to be numbered one) and ChOffset should be set to contain the difference of addresses between the specified tag on the two channels.

ChFirst is used when the offset is not constant between each channel. This is used, for example, by the Eurotherm E2500_temperature. The tag RegModW begins at address 16512 with an offset of 1 for the first 6 channels (up to 16517). But the channels #7 and #8 are at address 16532 and 16533.

Then, to do this, we must define two tags with the same name. ChFirst of the second tag is equal to 7.

#	Name	R/W	Address	Data type	Quantity	Frequency	ChCount	ChFirst	ChOffset
3	RegModeW	Write	16512	int 16 bits	1		6	1	1
4	RegModeW	Write	16532	int 16 bits	1		2	7	1

Table 4: Use of two tags for non-uniform channel spacing

FxA, FxB

It is possible to execute a linearization function based on a gain (FxA) and an offset (FxB).

When reading a value from the device, the function ***y=ax + b*** is applied with a=FxA, and b=FxB.
Then the value of the tag will be equal to <Tag Value> = fxA * <value read> + FxB

When writing the tag the reverse function is applied. The value sent to the device will be x = (y-b)/a
This means <value sent> = (<Tag Value> - FxB) / FxA

If linearization is not required, assign FxA = 1 and FxB = 0

Link Type

This defines a link type for this tag only. This is to associate the device to a sub-equipment.
Several identifiers can be used for the same tag but they must be separated by ";" as in the example : *bitIn;Shutter*

→ **See Architecture Section for more details.**

Comment

This is a free comment area.

Opt(ion)

- **Save value**: when checked, the value of the tag will be saved in the common data file of the project directory when closing CrystalXE or by the menu File / Save / Data as... The data will be also be reloaded at startup.
- **Write to device at startup**:
  When this option is checked then there are two cases:
  - If the option **Save value** is checked then the data loaded from the data file will be sent to the device at startup.
  - If the option **Save value** is **not** checked then the default value (see default column) will be sent to the device at startup.
- **Always present:** When checked, this means the tab won't be deleted at startup if it is not linked to a sub-equipment else it will be deleted. Because only tags which are linked to a sub-equipment are activated, all other are deleted.
- **No read request after writing:** This option prevents a read request from the tag concerned being made after a write frame has been sent to the device. This option is not activated by default, so a read request is systematically sent after a write frame.
- **Input register:** Use the modbus function 4 to request data (read n input registers)

Min

Low limit of the tag when writing to the tag. This avoids the possibility of sending an out of range value to the device. Such an out of range value will not be written into the modbus table image.

This limit is not used when reading a tag from the device. For example if Min=5 and the value read from the device is 3 then the value of the tag will become equal to 3.
It is possible to enter either a constant value or a tag name.

Max

Same as Min but for the high limit.

Default

Default value to assign to the tag at startup (if defined)
See also the Column **Opt** for more details.

OnWrite

To add a new script to the list, see the tab Script and push on the button Add.

If a script is assigned to this column then it will be executed when writing to the tag. If the tag has several channels then there will be an instance of this script for each channel.

The parameters oldVal and NewVal can be used in the script to reference the previous value before the write occurred, and the new written value.

The OnWrite script is executed:

- each time the tag is written (by a script or a recipe or by an object in an HMI form..)
- when another script of the regulator runs the function WriteTagValue or WriteTagValueEx if the parameter SendToDevice is true (not when it is false).
- when a tag of a sub-equipment is linked to this tag and is written. In that case, the OnWrite of the Sub-equipment and the OnWrite of the regulator are executed at the same time.

In other term, the OnWrite instruction is executed when the tag is sent to the device.

Depending of the script options, if the script is already running when a new write event occurred then there are two options:

- Either it will restart from the beginning
- or nothing will append and the current script will continue running.

The script options can be changed in the script editor, in the menu **Options / Script options** and the checkbox **"Wait the end of execution…"** See the debug section to have more details about Script options.

OnRead

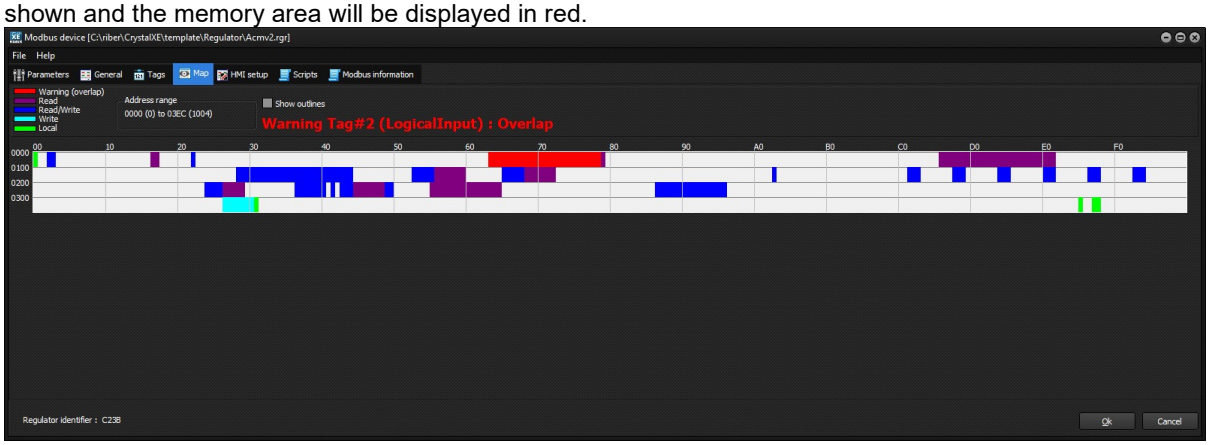
To add a new script to the list, see the tab Script and push on the button Add.

If a script is assigned to this column then it will be executed after a new value is read from the device. The parameters oldVal and NewVal can be used in the script to reference the previous value before the read occurred and the new read value.

The OnRead script is only executed when a value is read from the device (when receiving the frame), never when another script or other module in Crystal XE read the tag.

TAB Map

This map shows the occupation of the modbus table. If a tag overlaps another tag then an error will be shown and the memory area will be displayed in red.



You can define your own HMI to setup the regulator if needed.

When it is defined, this HMI can be opened by a popup menu in the devices view.
Right click on the device and select Setup, this will open the setup window that you have defined.

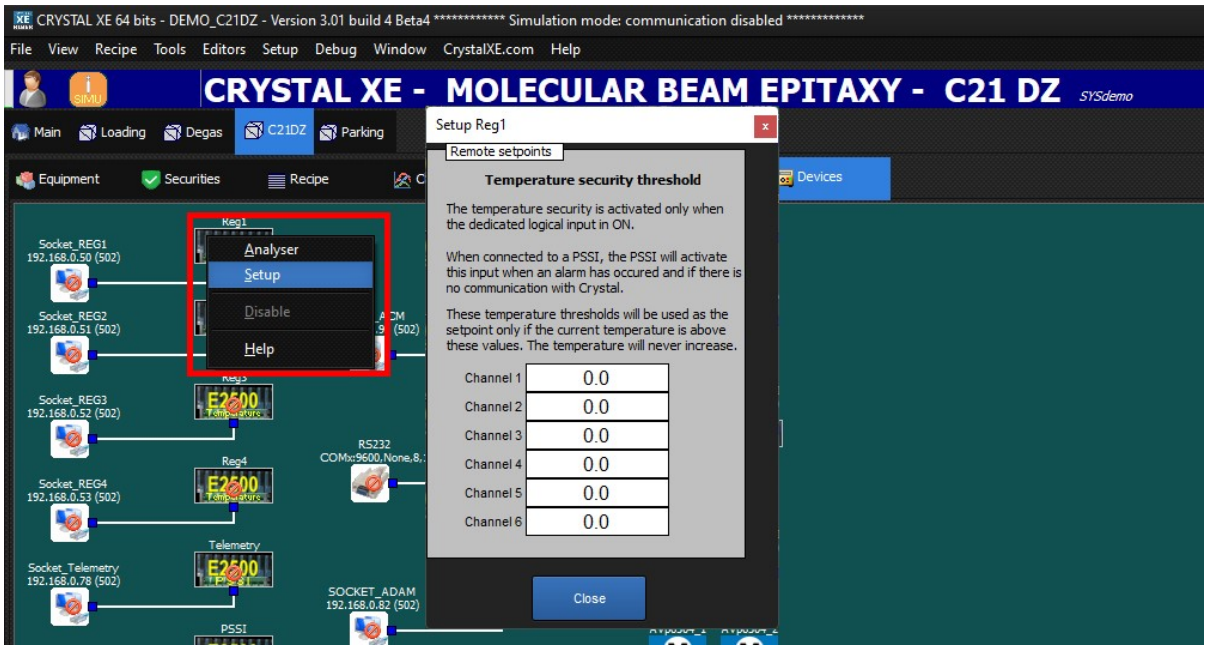


Figure 46:pop-up window to set up regulator

Note: The user must add a 'close' button when editing the HMI.
A script must be associated with this button to close the window (by calling the function close)

→ For more information about HMI, see the part about creating of HMI.

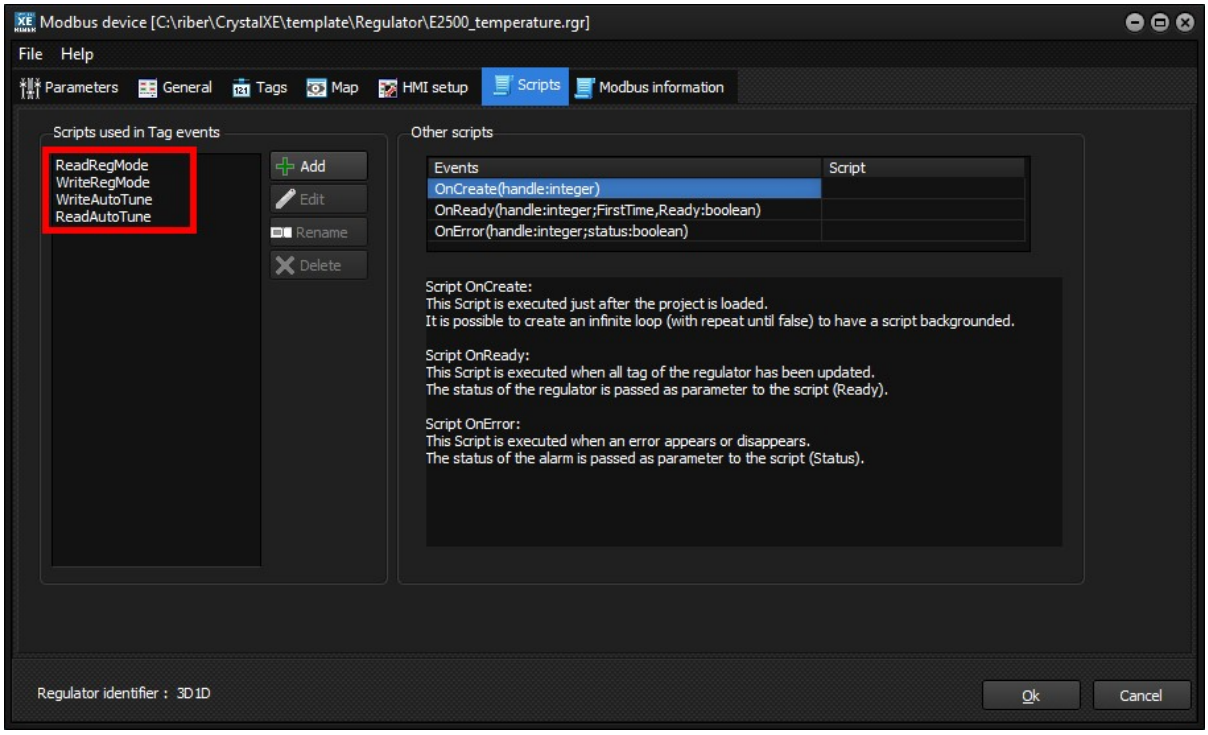
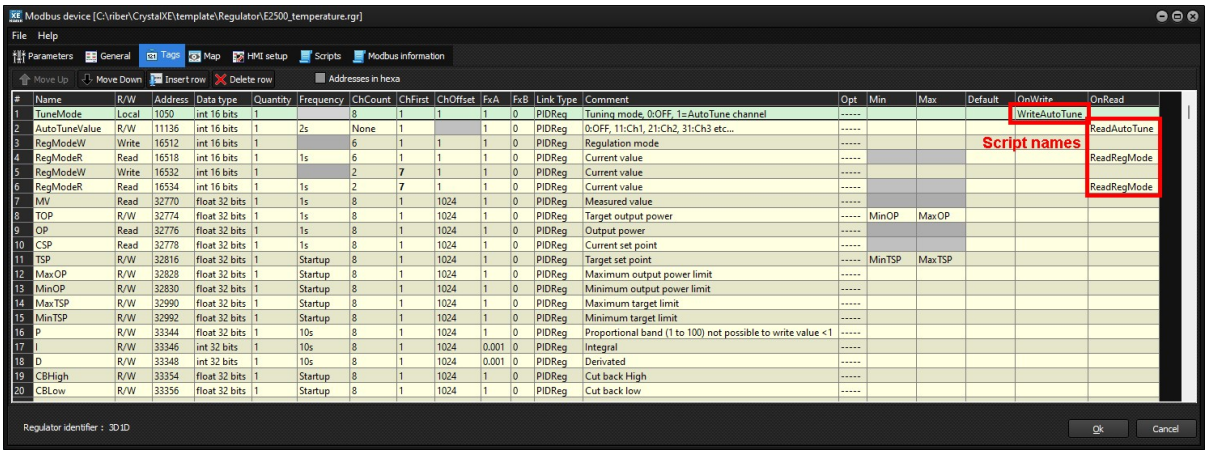


Figure 47: TAB scripts window

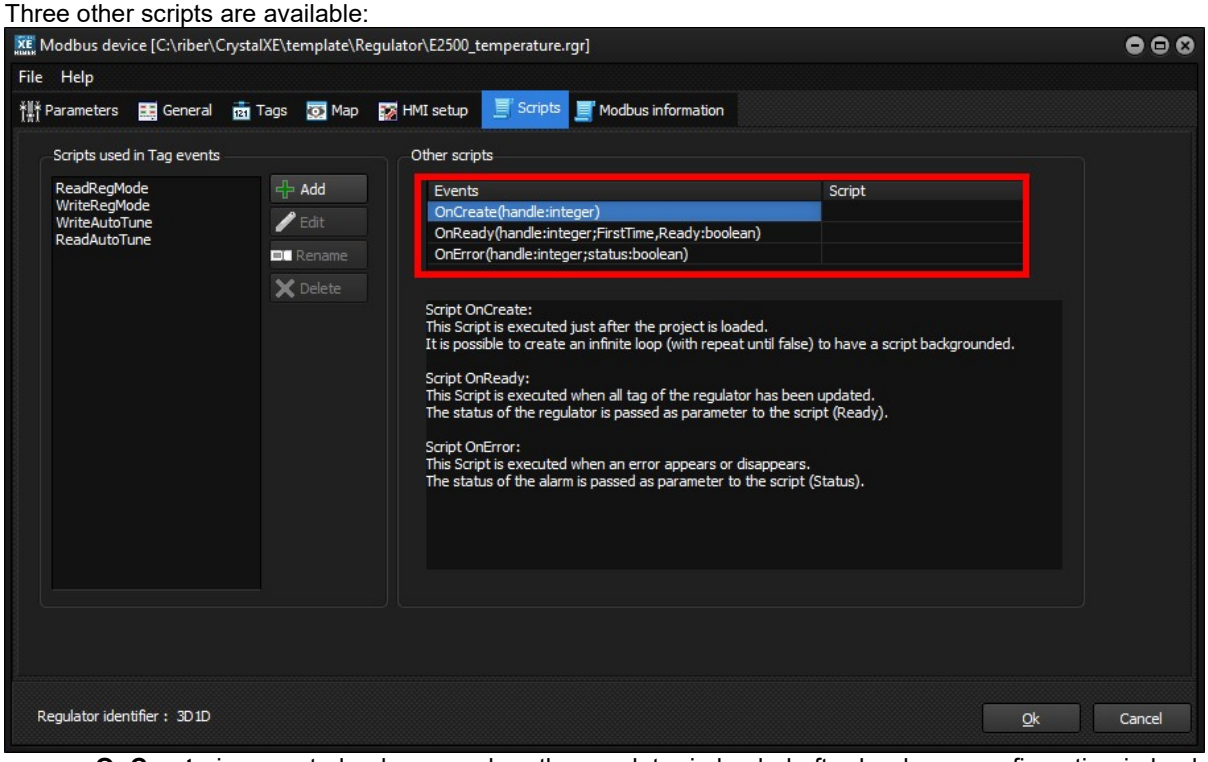


The scripts that are needed by the events OnRead and OnWrite of the tags can be added in this editor.
Once a new script has been added to the list, it will be visible in the combo box of the column OnRead and OnWrite of each tag.
To create a new script, first click the **Add** button.
To edit a script, select the script and click on the **Edit** button or **double click** on the script name.
To rename a script and all its references in the tag view, click on the **Rename** button.
To remove a script, click on the **Delete** button.

Parameters used by these scripts:

- **Handle (integer):** Contain an integer in which each byte represent the current regulator number, the linked sub-equipment number and the equipment number. Refer to the script function GetHandleStr for more details.
- **IdTag (word):** this is the TagId of the current tag.
- **Channel (word):** this is the current channel number of the current tag.
- **Onload (boolean):** This is true when the tag is loaded at startup from the data file or the default value.
- **OldVal (real):** Value of the tag before a read or a write (depends of the Event which call the script)
- **NewVal (real):** value read of written of the tag (depends of the the event which call the script)

Three other scripts are available:



- **OnCreate** is executed only once when the regulator is loaded after hardware configuration is loaded. If you need a script running all the time in background task then you can use an infinite loop in the OnCreate script like “repeat …. Until false”. In that case, we recommend to add a sleep(xxx) to avoid overloading the CPU. See above to have details about the parameter *handle*.

- **OnReady** is executed after all tags have been updated. This may occurred several time in a Crystal session because after a communication fault if the connection is reestablished then the OnReady will occur again. It may be useful to know if it is the first OnReady event by testing the parameter FirstTime. The parameter *Ready* indicates if it is ready or not. See above to have details about the parameter *handle*.

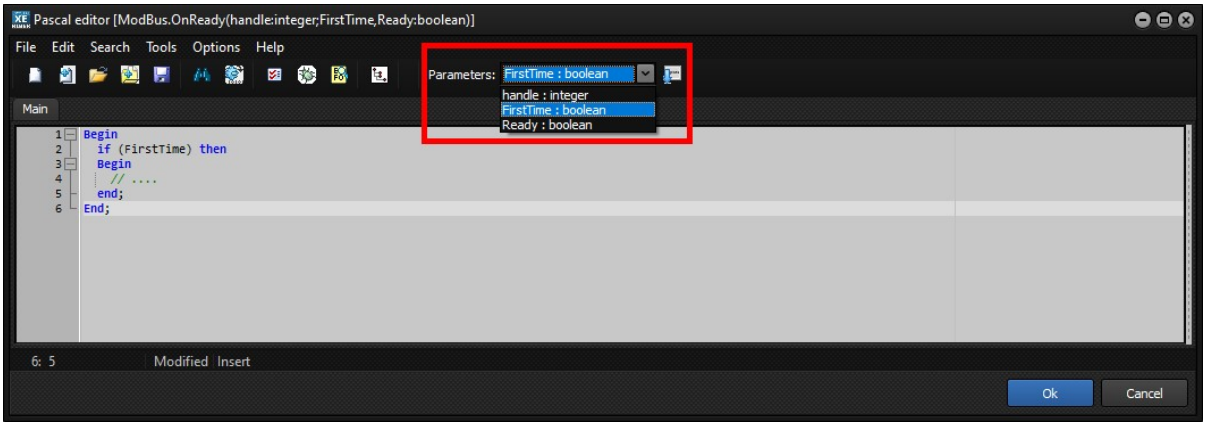


Figure 48: Example of using the parameter “FirstTime” in the script OnReady

- **OnError** is executed when the error status changes (when error occurs or disappears). Check the parameter *status* to know if the module is in error (true) or if the error has disappeared (false)
  See above to have details about the parameter *handle*.

4.1.2 ASCII module

To create a new device module using ASCII protocol, you may either begin from an empty page or copy an existing device and modify it.

To begin a new device module using ASCII protocol from an empty page, click on this button

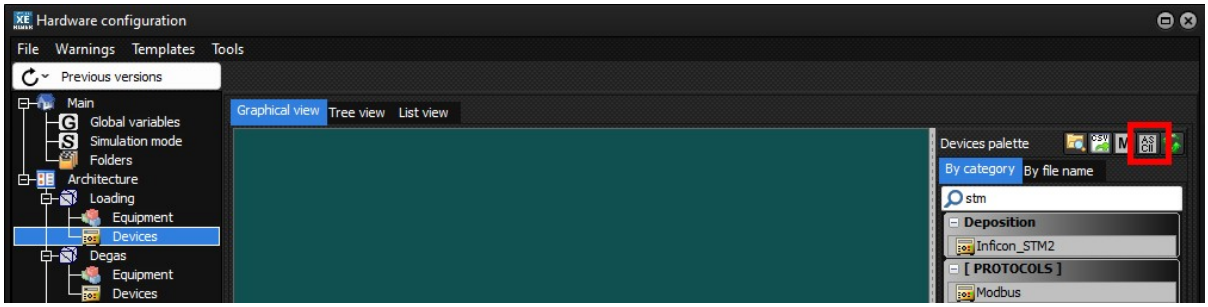


Figure 49: Starting a new template based on the ASCII protocol

This will open the ASCII editor (see below):

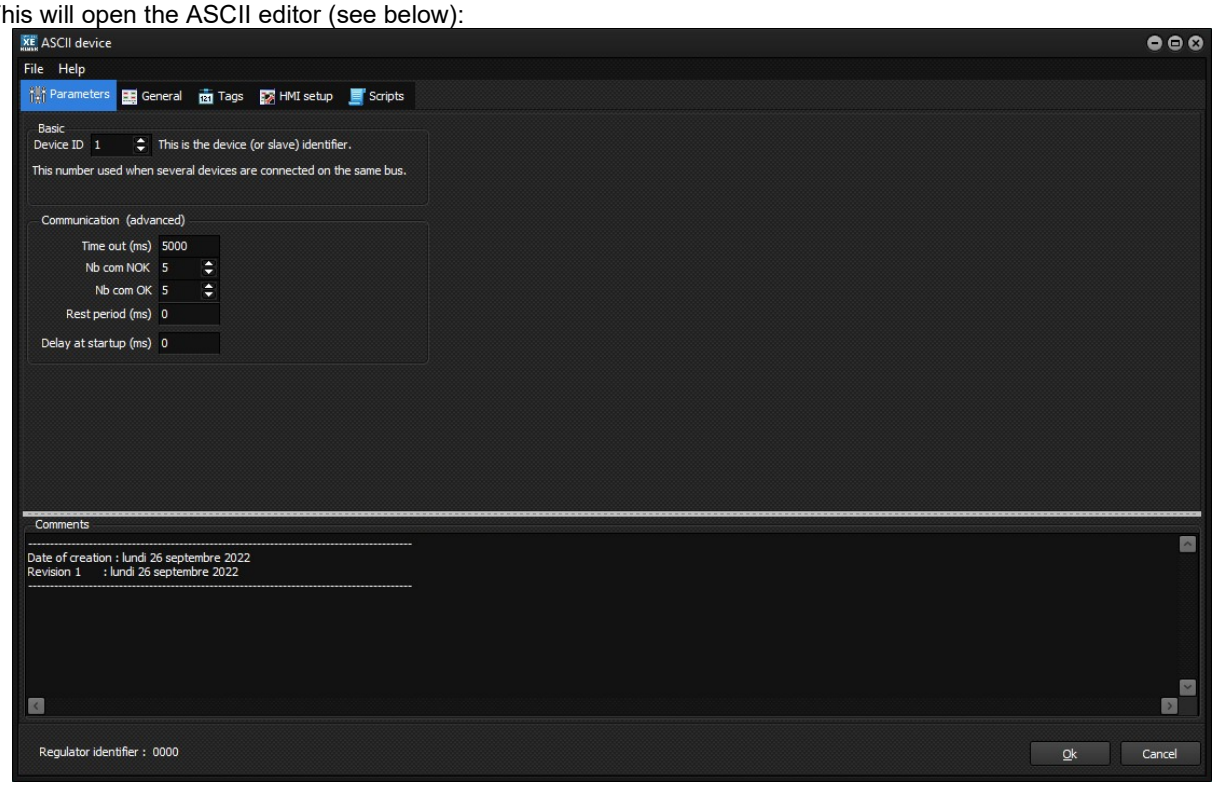


Figure 50: ASCII editor

TAB *Parameters*

Device ID

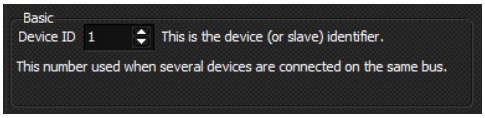


Figure 51: Device ID definition

This is the device identifier used by the ASCII protocol. Depending of the device, this identifier can be programmed by a dipswitch on the device or by software. This is used for the mnemonic <GID> and <UID>

Communication parameters

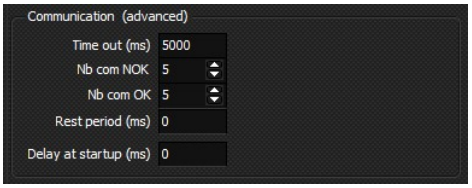


Figure 52: Communication parameters

Time out: If no answer is received after this time for a reading or a writing request then the request will be send again to the device. If the number of retries (defined in the Nb com NOK box) is reached then a timeout will occur. A critical event will raised and after a delay Crystal XE will try again. Then after a time out, if the device responds several times (as defined in the Nb com OK box) then the communication fault flag will be removed.

Rest period: this value defines the time to wait before each frame. This is in addition to the inter-message gap that is defined as part of the Modbus protocol.

Delay at startup (ms): The communication will start at the end of this time, after Crystal XE is loaded. This way, it is possible to make sure that all modules have been loaded before starting to refresh the tags.

Comments

Figure 53: Comment section

This is a free text area where the user generally indicates the versions and evolutions of the template.

Customize the template

This box appears only when you double click on the module. It does not appear when editing the sub-equipment.

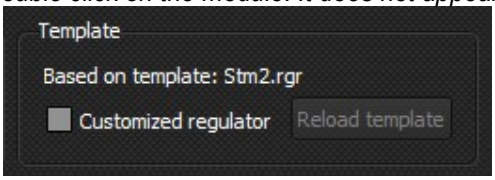


Figure 54: Debug box

Debug group

This box appears only when you double click on the module. It does not appear when editing the sub-equipment.

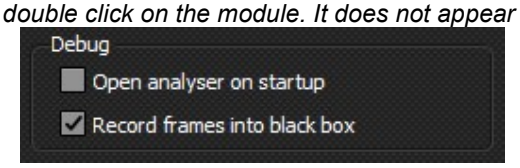
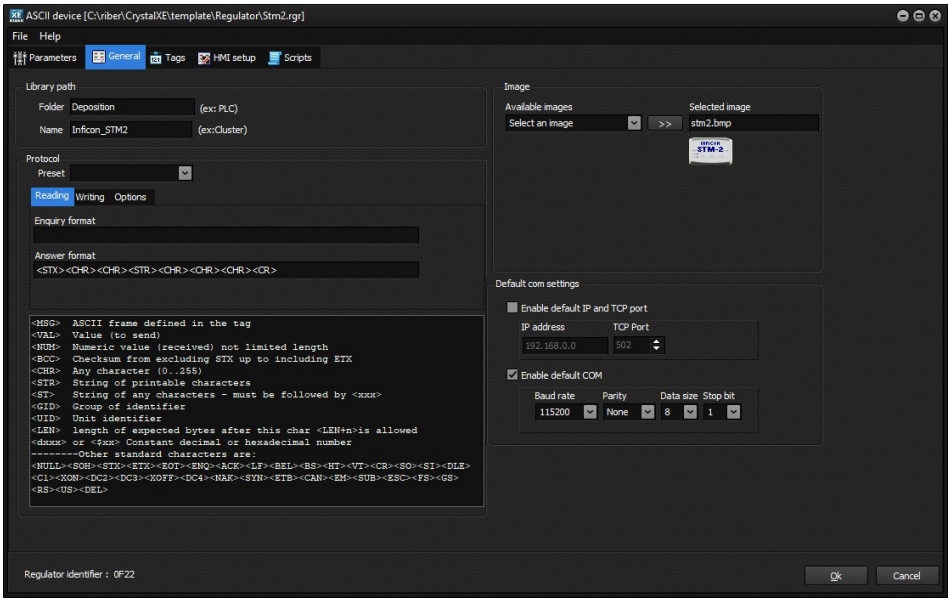


Figure 55: Debug box

Check box "Open analyzer on startup": if enabled, then the analyzer of this module will be opened when Crystal XE will open or when exiting the hardware configuration by pushing the OK button.

Check box "Record frames into black box": if enable and if the black box is enabled in the options window, then all frames of this module will be store in the black box files (see the user manual for more details)..

TAB *General*

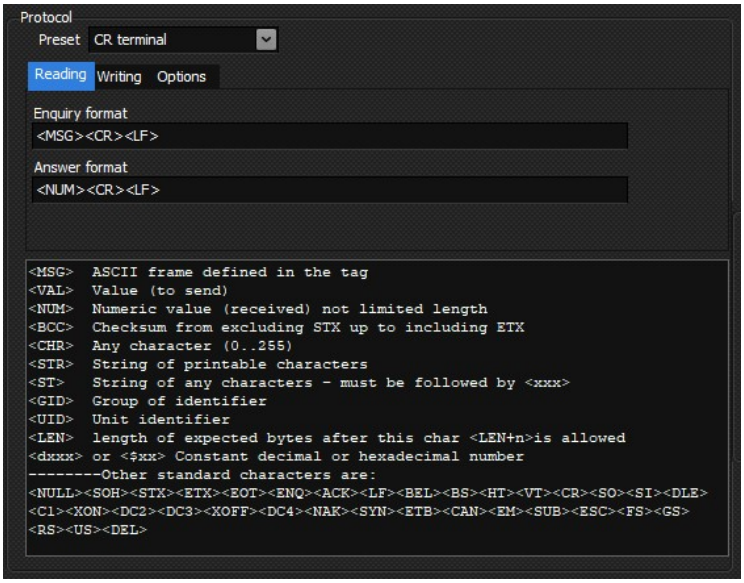


Library path:

Figure 56: The library path defines the location of the library in the palette

For information about this box, see the section "***Create your own module as a template***"

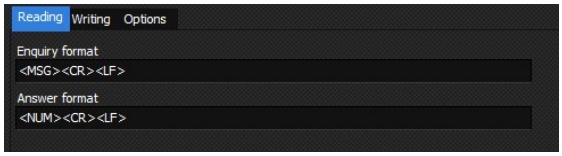
Protocol



This is where you can define how the exchanges between Crystal XE and the device take place. The Reading tab defines the exchange protocol during a tag reading. The Writing tab, when writing/modifying a tag.
Crystal XE sends the enquiry frame and receives a frame following the Answer format.

Use the Preset combo box to select a standard ASCII protocol like the EibiSynch protocol.

Protocol / Tab Reading



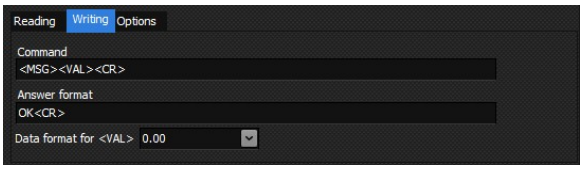
If you fill the Enquiry format field then all tags which are in Read or R/W will send a request frame with the period defined in the frequency column. To define the Enquiry format, click on the link to display the help.

Note: Leave the "Enquiry format" field empty if you want to manage the sending by a specific script in column “OnRequest” in the tags tab.

Example of reading frame for the EibiSynch format:

- Enquiry format: <EOT><GID><GID><UID><UID><MSG><ENQ>
- Answer format: <STX><MSG><NUM><ETX><BCC>

Protocol / Tab Writing



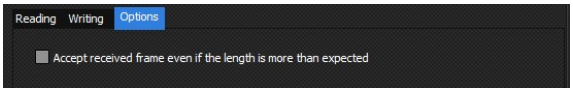
Same as for the tab reading, fill the command field and answer format. Then all tags which are in Write or R/W will send a command with this format when a write event will occur.
You can also define the format or the value <VAL> which will be sent in the frame.

Note: Leave the "Command" field empty if you want to manage the sending by a specific script in column “OnWrite” in the tags tab.

Example of writing frame for the EibiSynch format:

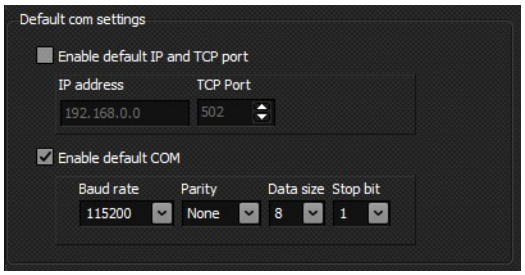
- Command: <EOT><GID><GID><UID><UID><STX><MSG><VAL><ETX><BCC>
- Answer format: <ACK>

Protocol / Options



Check the box to accept received frame even if the length is more than expected.

Default com settings:



Enable or not default TCP/IP settings or serial COM settings. This will be used when linking the device with a socket or serial com port.

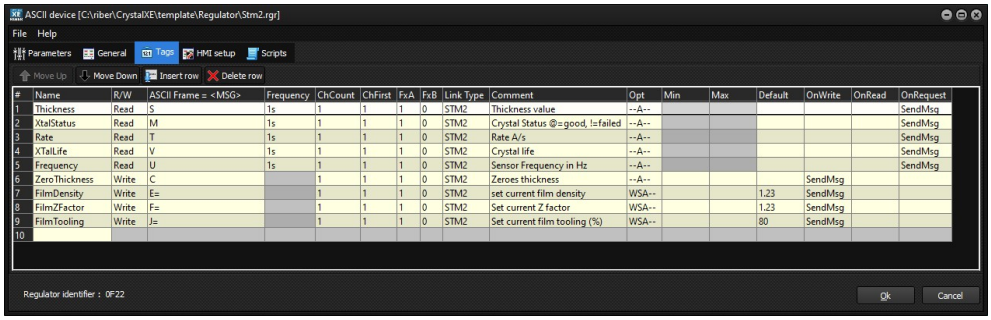


Figure 57: Tags tab in editor

Name

The name of the tag which may be used also by the sub-equipment to identify the tag. Allowed characters are letters, numbers and underline, no spaces, no special characters.
The name must not start with a number.

R/W

None	Tag is disabled. Scripts onRead and onWrite will be never executed (for debug only)
Read	Can only be read. It is not possible to change the value by using the script or HMI objects. A write frame will never be sent to the device.
Write	Can be read and write by the script or HMI objects but only write frames are sent to the device.
R/W	Both read and write can be done on this tag.
Local	No read or write frame will be sent to the device. This tag is only used as a variable. Take care to use a free Address.

ASCII frame (MSG)

Define the message to assign to the <MSG> parameter.
In other terms, the parameter <MSG> which may defined in the protocol definition for Reading frame and Writing frame (see the tab General) will be replaced by this message.

Frequency

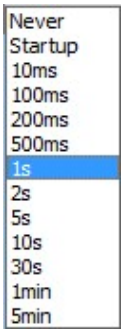


Table 5: List of available frequencies for update

This defines the minimum refreshing time of the tag.
This time is not guaranteed as it depends of the communication flow.

Select ***Never*** if you don't want Crystal send reading request for this tag. This does not prohibit sending a writing frame when writing to this tag, as defined in a script.

Select ***Startup*** to refresh the tag only when Crystal is loading. Again, it can be refreshed as a script action.

ChCount and Chfirst

These columns define the channels.

ChCount is the number of channels, in other term, the number of times that this tag is repeated. For example, it you communicate with a Temperature regulator which has 2 PIDs with exactly the same tags, then select 2 for the ChCount, 1 for the ChFirst (if the first channel is to be numbered one) and ChOffset should be set to contain the difference of addresses between the two channels.

FxA, FxB

It is possible to execute a linearization function based on a gain (FxA) and an offset (FxB).

When reading a value from the device, the function y=ax + b is applied with a=FxA, and b=FxB.
Then the value of the tag will be equal to <Tag Value> = fxA * <value read> + FxB

When writing the tag the reverse function is applied. The value sent to the device will be x = (y-b)/a
This means <value sent> = (<Tag Value> - FxB) / FxA

If linearization is not required, assign FxA = 1 and FxB = 0

Link Type

This defines a link type for this tag only. This is to associate the device to a sub-equipment.
Several identifiers can be used for the same tag but they must be separated by ";" as in the example : *bitIn;Shutter*
→ **See section about links in the Architecture Section for more details.**

Comment

This is a free comment area that is usually used to comment on version changes.

Opt(ion)

Available options are:

- **Save value**: when checked, the value of the tag will be saved in the common data file of the project directory when closing CrystalXE or by the menu File / Save / Data as... The data will also be reloaded at startup.
- **Write to device at startup**:
  When this option is checked then there are two cases:
  - If the option **Save value** is checked then the data loaded from the data file will be sent to the device at startup.
  - If the option **Save value** is **not** checked then the default value (see default column) will be sent to the device at startup.
- **Always present:** When checked, this means the tab won't be deleted at startup if it is not linked to a sub-equipment else it will be deleted. Because only tags which are linked to a sub-equipment are activated, all other are deleted.
- **Do not insert value in frame:** If checked the value will not be inserted in the writing frame for this tag only.
- **String of characters:** When this box is checked the tag contains a string of characters and not a value.

Min

If defined, this value is the low limit of the tag when writing to the tag. This avoids the possibility of sending an out of range value to the device.
This limit is not used when reading a tag from the device. For example if Min=5 and the value read from the device is 3 then the value of the tag will become equal to 3.
It is possible to enter either a constant value or a tag name.

Max

Same as Min but for the high limit.

Default

Default value to assign to the tag at startup (if defined)
See also the Column **Opt** for more details.

OnWrite

To add a new script to the list, see the tab Script and push on the button Add.

If a script is assigned to this column then it will be executed when writing to the tag. If the tag has several channels then there will be an instance of this script for each channel.

The parameters oldVal and NewVal can be used in the script to reference the previous value before the write occurred and the new written value.

The OnWrite script is executed:

- each time the tag is written (by a script or a recipe or by an object in an HMI form..)
- when another script of the regulator runs the function WriteTagValue or WriteTagValueEx if the parameter SendToDevice is true (not when it is false).
- when a tag of a sub-equipment is linked to this tag and is written. In that case, the OnWrite of the Sub-equipment and the OnWrite of the regulator are executed at the same time.

Depending of the script options, if the script is already running when a new write event occurred then it will restart from the beginning or nothing will append. The script options can be changed in the script editor, in the menu **Options / Script options** and the checkbox **"Wait the end of execution…"**

OnRead

To add a new script to the list, see the tab Script and push on the button Add.

If a script is assigned to this cell then it will be executed **after** a new value is received from the device. The parameters oldVal and NewVal can be used in the script to reference the previous value before the read occurred and the new read value.

The OnRead script is only executed when a value is received from the device (when receiving the frame).

OnRequest

To add a new script to the list, see the tab Script and push on the button Add.

If a script is assigned to this cell then it will be executed when the time which is defined in the column frequency will be expired.

TAB HMI Setup

You can define your own HMI to setup the device if needed.

When it is defined, this HMI can be opened by a popup menu in the devices view.
Right click on the device and select Setup, this will open the setup window that you have defined.

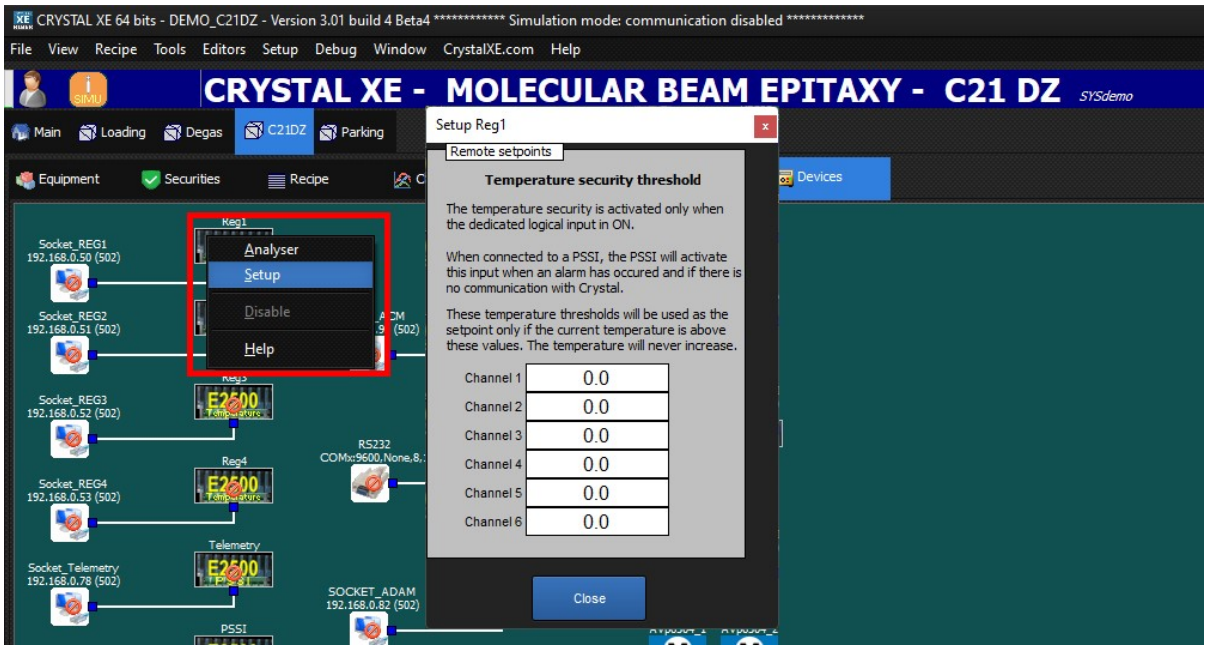


Figure 58:pop-up window to set up regulator

Note: The user must add a 'close' button when editing the HMI.
A script must be associated with this button to close the window (by calling the function close)

→ For more information about HMI, see the part about creating of HMI.

TAB Scripts

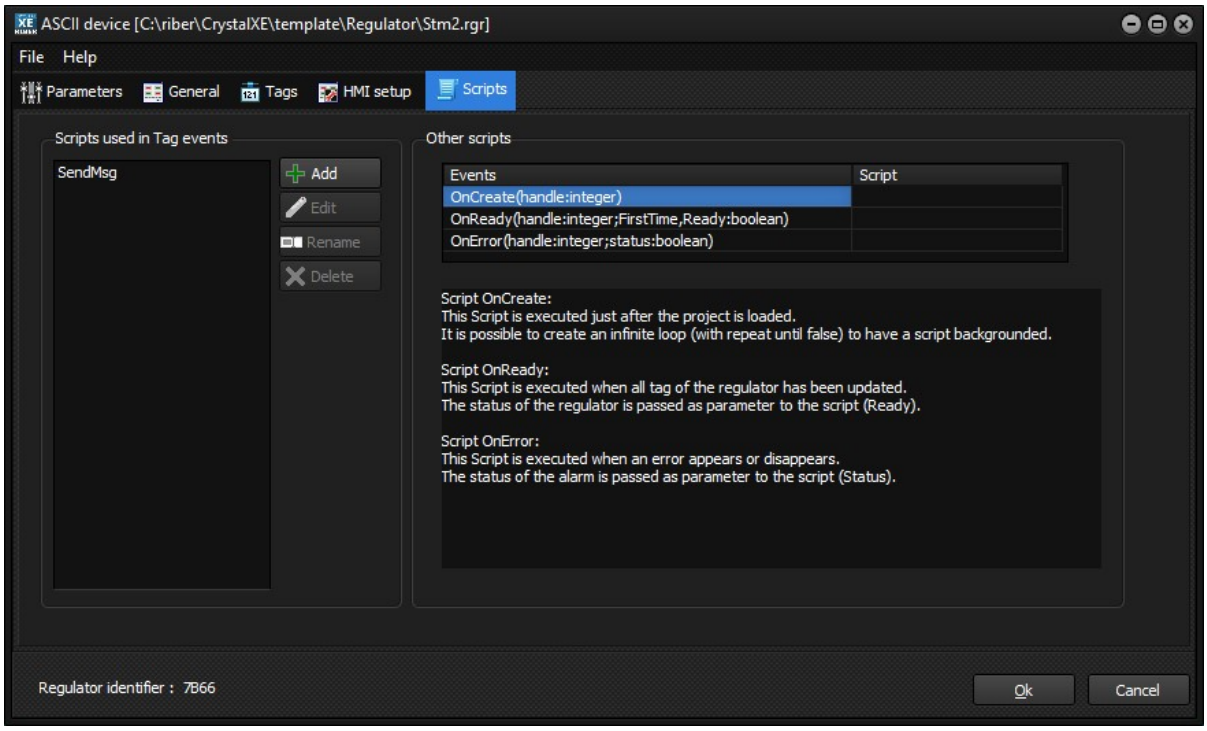
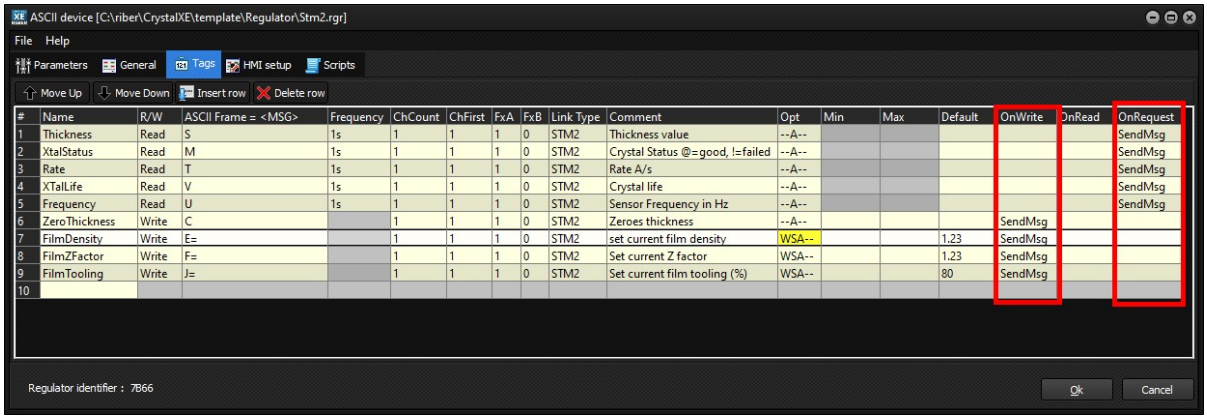


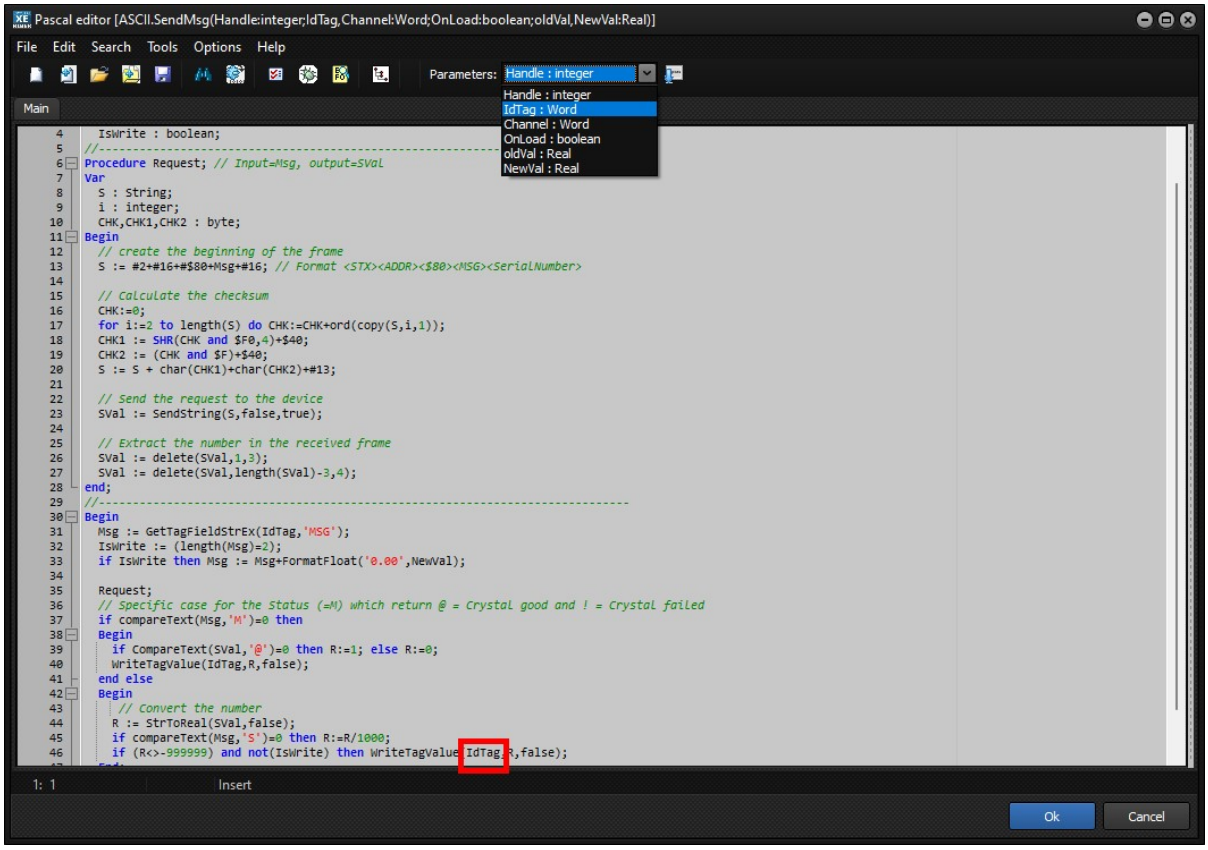
Figure 59: TAB scripts window



The scripts that are needed by the events OnWrite, OnRead and OnRequest of the tags can be added in this editor.
Once a new script has been added to the list, it will be visible in the combo box of the column OnWrite, OnRead and OnRequest of each tag.
To create a new script, first click the **Add** button.
To edit a script, select the script and click on the **Edit** button or **double click** on the script name.
To rename a script and all its references in the tag view, click on the **Rename** button.
To remove a script, click on the **Delete** button.

Parameters are passed in the script. They are visible through the combo box "Parameters" and can be inserted in the editor through the button on the right of the list.

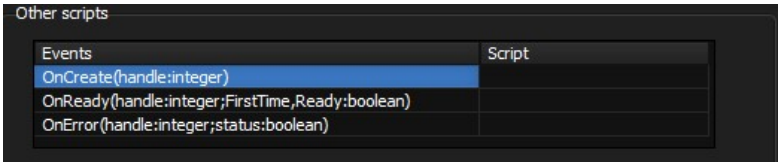
Example with the parameter IdTag:



Parameters used by these scripts:

- **Handle (integer):** Contain an integer in which each byte represent the current regulator number, the linked sub-equipment number and the equipment number. Refer to the script function GetHandleStr for more details.
- **IdTag (word):** this is the TagId of the current tag.
- **Channel (word):** this is the current channel number of the current tag.
- **Onload (boolean):** This is true when the tag is loaded at startup from the data file or the default value.
- **OldVal (real):** Value of the tag before a read or a write (depends of the Event which call the script)
- **NewVal (real):** value read of written of the tag (depends of the the event which call the script)

Three other scripts are available:



- **OnCreate** is executed only once when the regulator is loaded after hardware configuration is loaded. If you need a script running all the time in background task then you can use an infinite loop in the OnCreate script like “repeat …. Until false”. In that case, we recommend to add a sleep(xxx) to avoid overloading the CPU. See above to have details about the parameter *handle*.

- **OnReady** is executed after all tags have been updated. This may occurred several time in a Crystal session because after a communication fault if the connection is reestablished then the OnReady will occur again. It may be useful to know if it is the first OnReady event by testing the parameter FirstTime. The parameter *Ready* indicates if it is ready or not. See above to have details about the parameter *handle*.

- **OnError** is executed when the error status changes (when error occurs or disappears). Check the parameter *status* to know if the module is in error (true) or if the error has disappeared (false)
  See above to have details about the parameter *handle*.

Object: Create a device to control the Turbo Pump Agilent Twis Torr 84 FS.

The device protocol to read data consists to send this frame:
<STX><ADR><MSG><ETX><CRC1><CRC2>

With:
<STX> : 0x02
<ADR>: 0x80
<MSG>: ASCII message specific to the tag to read
<ETX> : 0x03
<CRC1><CRC2> is the ASCI representation of the checksum calculated in a byte from <STX> to <ETX>

The answer is:
<STX><ADR><VAL><ETX><CRC1><CRC2>
<VAL> is the ASCI representation of the value (ex : the string '2240.5E-07')

1) Clear the fields "Enquiry format" in the tab Reading and Writing.
2) For the Answer format, enter: <STX><CHR><NUM><ETX><CHR><CHR>
3) Go in the tab script, add a script that you name RequestMsg (for example) and copy this code:

```
//-----------------
Const
  STX = #$2;
  ETX = #$3;
Var
  S, Frame,Msg  : String;
  Bcc: byte;
  i : integer;
  Value : real;
Begin
   // we get the message of concerned tag
  Msg := GetTagFieldStrEx(IdTag,'MSG');

  // We build the frame to send
  Frame := #$80+Msg+ETX;

  // We calculate the BCC
  Bcc := 0;
  for i:=1 to length(Frame) do Bcc := Bcc XOR ord(copy(Frame,i,1));

  // We add STX and the BCC in ASCII at the end of the frame
  Frame := STX + Frame + IntToHex(Bcc,2);

  // We send the frame now
  S := SendString(Frame,false,true); // false says that it is a reading frame, then the field Answer will be
used to check the frame and to return the result into the script

    // In this example we don't check the checksum
    // We extract the value
  delete(S,1,2); // we delete <STX> and #$80
  S := copy(S,1,pos(ETX,S)-1);

 // We convert the value in a variable of type real
 Value := StrToReal(S,false);

   // We update the value of the tag
   WriteTagValue(IdTag,Value,False);
End;
//-----------------
```

4) Add a tag, then in the column MSG of this tag enter 2240 (the message of this tag for this device) and in the column OnRequest, select the script that you have just written (RequestMSG)
5) Do the same for each tags

To write a value, you must defined another script like this one but the function SendString must have the parameter to TRUE to indicate using the writing format that you can define in the tab General / Writing / Answer format.

4.2 Equipment editor

To create new equipment, you may either begin from an empty page or by editing an existing equipment and modifying it.

IMPORTANT: To be registered in the configuration, when you update an equipment template file, you need to remove all examples of this equipment from the tree view of the hardware configuration and to drag and drop the instances of the equipment again.
This is only applies to equipment, not to sub-equipment and devices (regulators).

To edit an existing template, right click on an item in the palette in the tab “by category” or “by file name”:

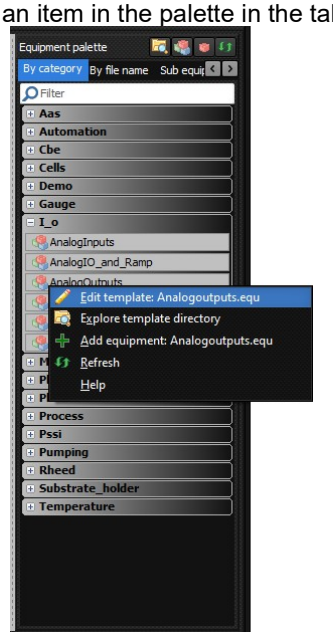


Figure 60: Selection of an existing equipment template

Alternatively, click on the link of an equipment:

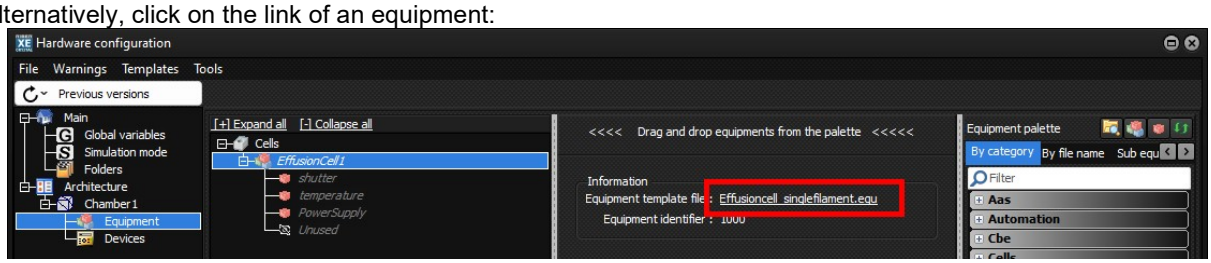


Figure 61:Alternative selection of equipment template

To begin new equipment from an empty page, click on this button

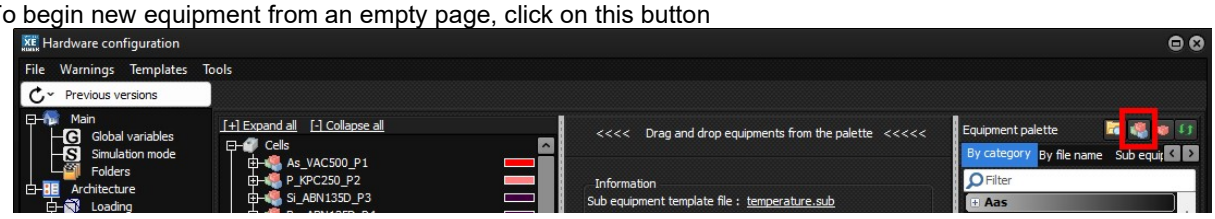


Figure 62: New equipment definition on an empty page

The equipment editor consists of several tabs

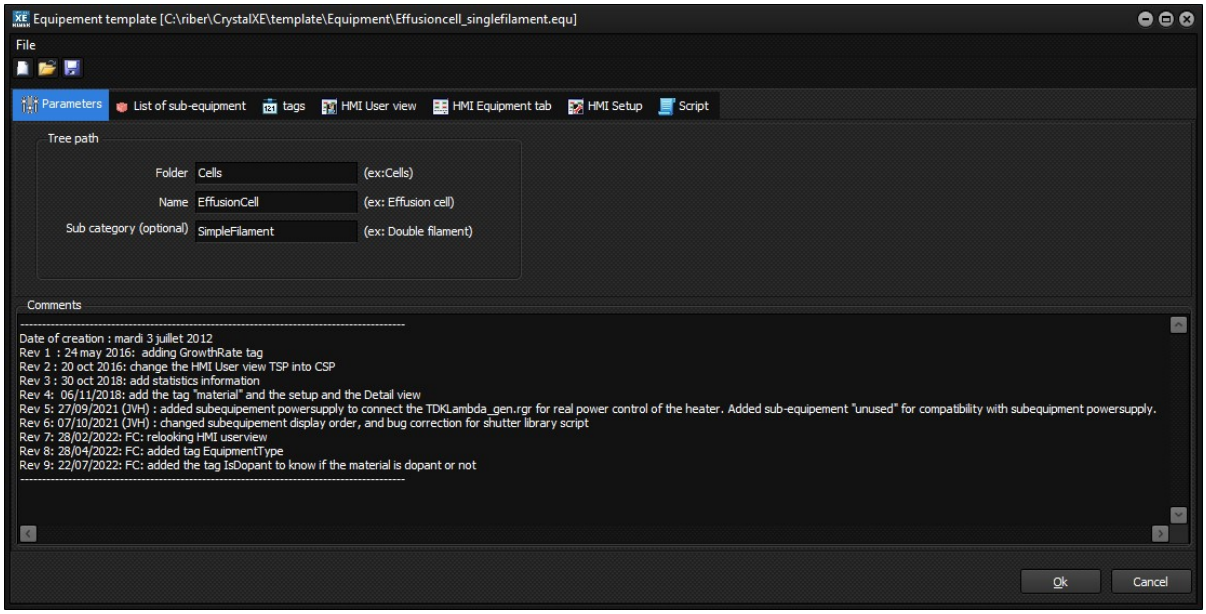


Figure 63: Equipment editor

TAB *Parameters*

Tree path:

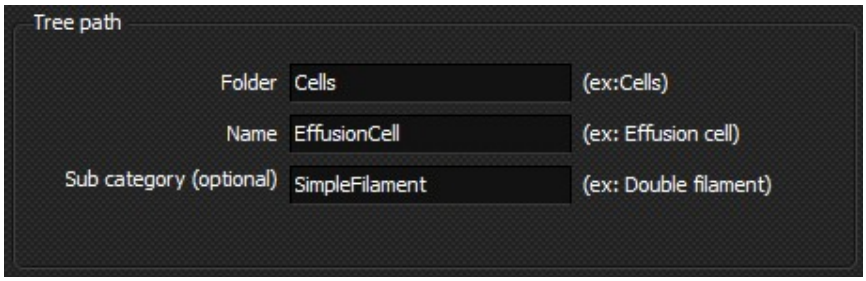


Figure 64: Tree path in the tab parameters

For information about this box, see the section "***Create your own module as a template***"

Comments

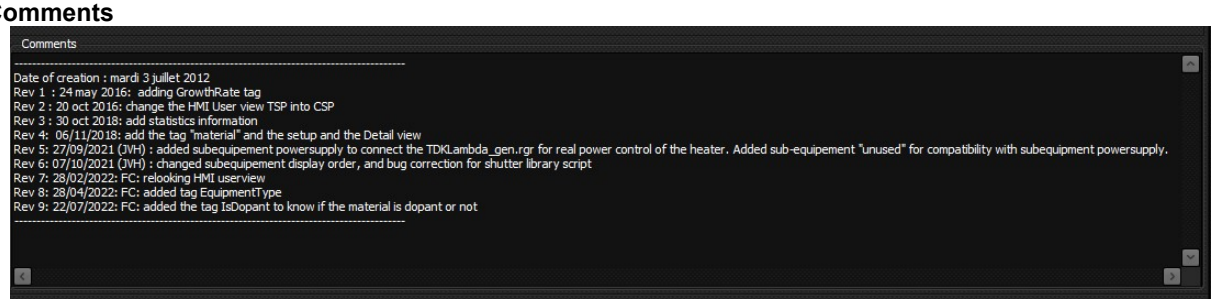


Figure 65: Comment section

This is a free text area where we generally indicate the versions and evolutions of the template.

TAB *List of sub-equipment*

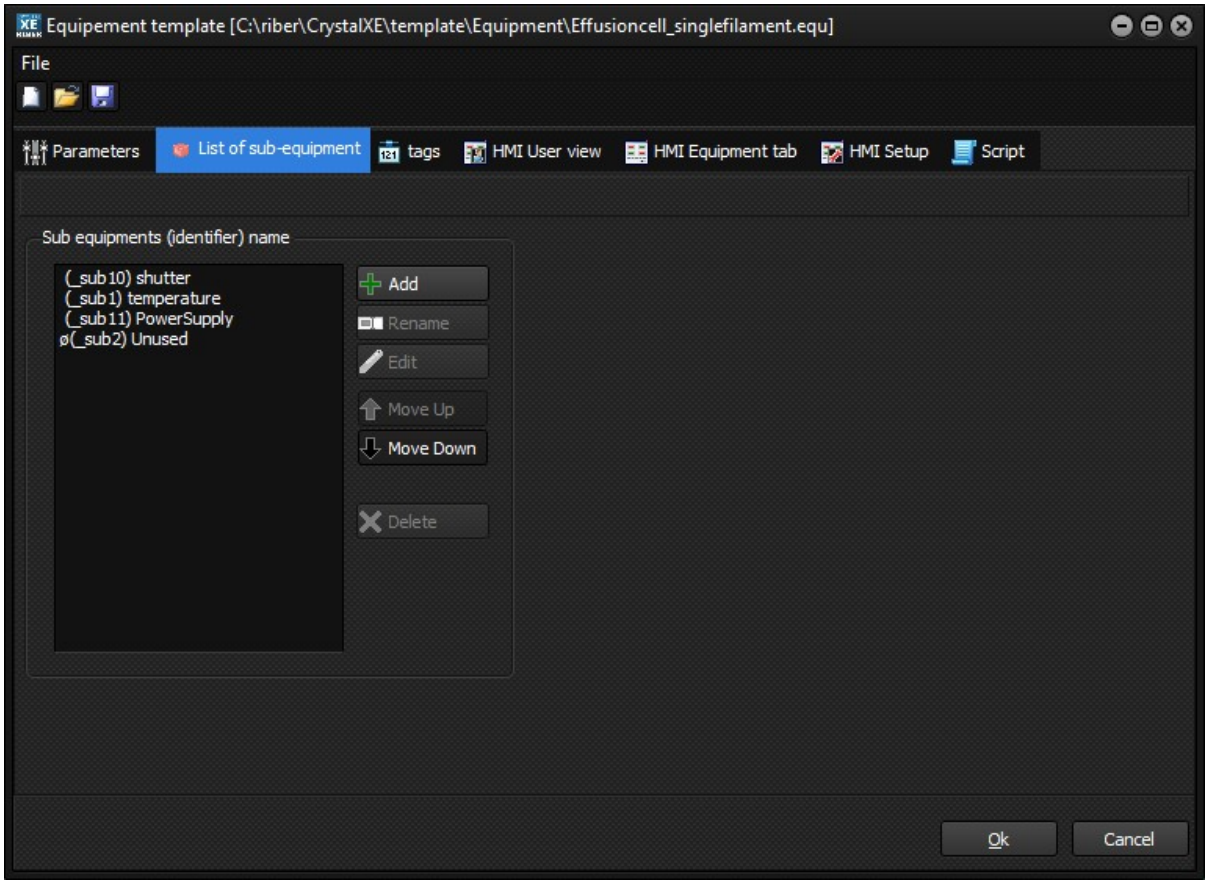


Figure 66: List of sub-equipment

Click on “add” to add a new sub-equipment and the appropriate other buttons to rename, edit, move or delete the selected item.

Tags of the sub-equipment may be accessible by the scripts of the equipment or objects in the HMI by prefixing _subx to the tag name, where x is the identifier of the sub-equipment. This identifier may be modified here.

Example : To access to the tag MV of the sub-equipment identified by _sub2, use the property _sub2.MV
This property can be used in the HMI system, HMI Setup and HMI user and scripts.

In a script, if you want to know if a sub-equipment is connected then check the Boolean ***_subx.RegConnected***

Use the function PopupDetailView if you want to open a popup window which displays the detail view. For example, in the user view you can add a button to open the detail view on clicking on this button in the user view.

See example below:

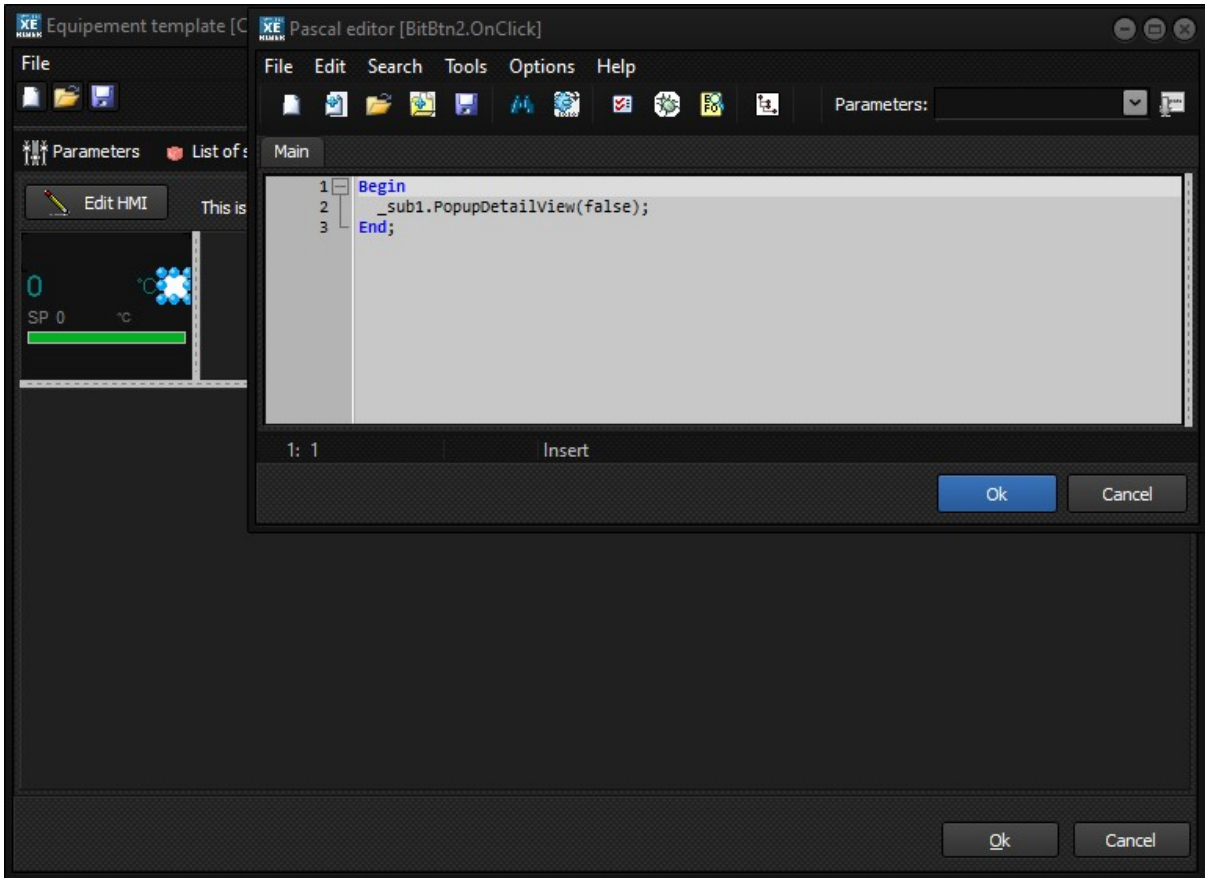
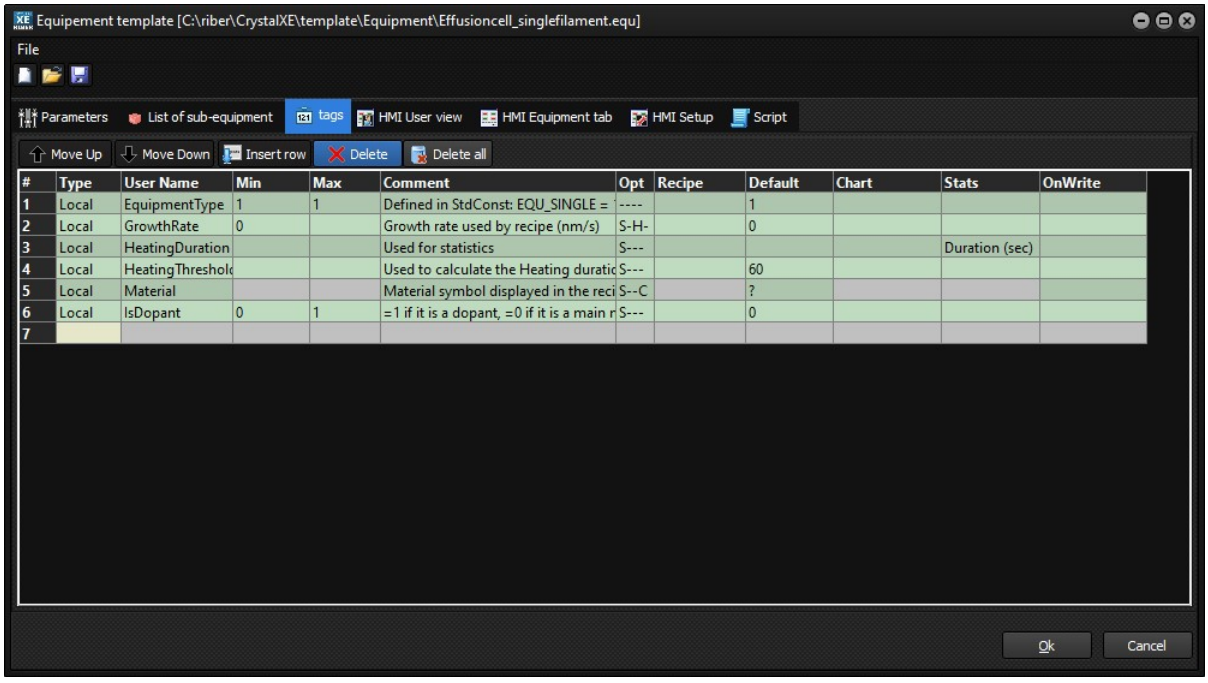


Figure 67: Example of use of PopupDetailView

TAB *Tags*



#	Type	User Name	Min	Max	Comment	Opt	Recipe	Default	Chart	Stats	OnWrite
1	Local	EquipmentType	1	1	Defined in StdConst: EQU_SINGLE =	----		1			
2	Local	GrowthRate	0		Growth rate used by recipe (nm/s)	S-H-		0			
3	Local	HeatingDuration			Used for statistics	S---				Duration (sec)	
4	Local	HeatingThreshold			Used to calculate the Heating duratic	S---		60			
5	Local	Material			Material symbol displayed in the reci	S--C		?			
6	Local	IsDopant	0	1	=1 if it is a dopant, =0 if it is a main r	S---		0			
7											

Figure 68: List of tags of equipment

Type

Type may be **Local** or **Extern**.

Local: The tag is like a local variable.
Extern: The tag is a short cut to a property which will be defined by the user when configuring the project. For example, if a valve controller needs to move the valve in case of the temperature increases, then we need to implement an external tag for the temperature reading. This will be a link to another tag. During the configuration, the user will specify which temperature measurement to use for the valve.

Another example, the bakeout need to know the pressure to stop heating when the pressure increases. The next pictures illustrate how to configure the external tag used to read the pressure.

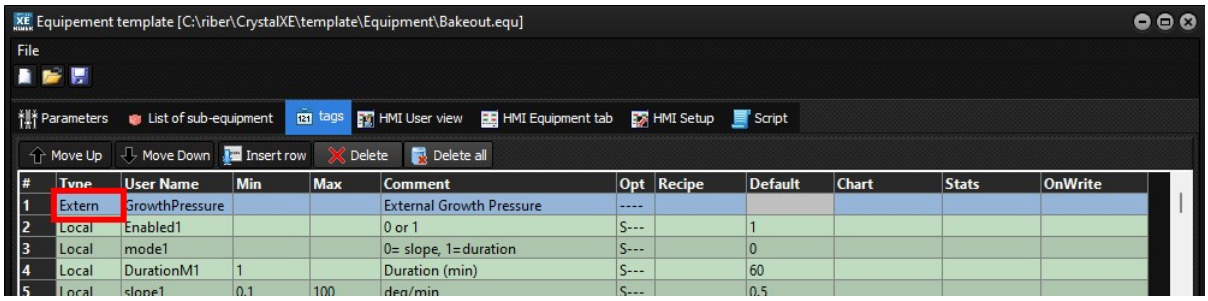


Figure 69: Example of external tag

Then when adding the equipment in the configuration, user will have the possibility to link this tag (see picture below):

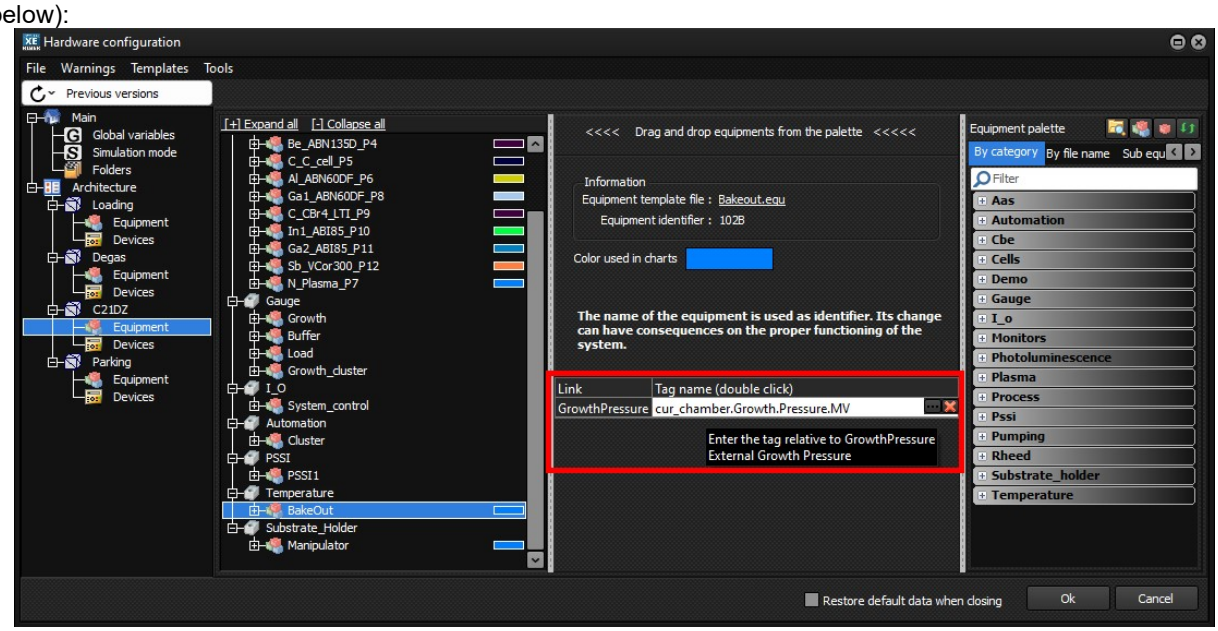


Figure 70: Example of using external tag

User name

The name of the tag which may be used also by the sub-equipment identify the tag. Allowed characters are letters, numbers and underline, no spaces, no special characters.
The user name must not start with a number.

Min

Low limit of the tag when writing to the tag.
It is possible to enter either a constant value or a tag name.

Max

Same as Min but for high limit.

Comment

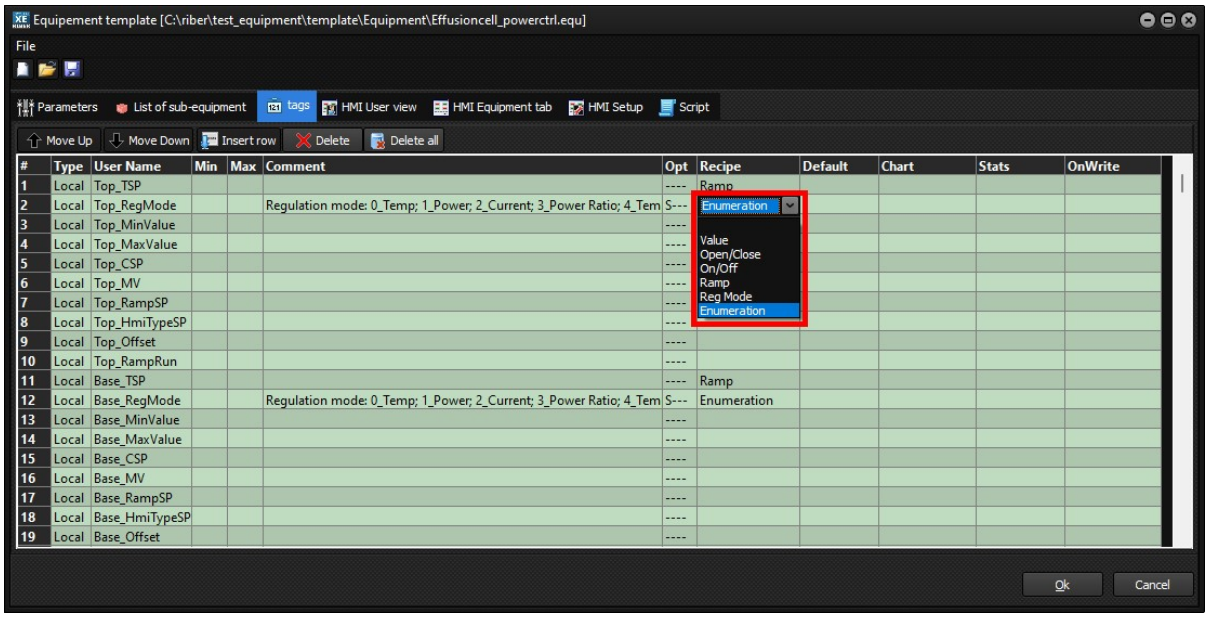
This is a free comment.

Opt(ion)

Available options are:

- **S-Save and load value in data file**: when checked, the value of the tag will be saved in the common data file of the project directory when closing CrystalXE or by the menu File / Save / Data as... And also, the data will be reloaded at startup.
- **R-Add to default Recorder template**: when checked the tag will be added to the default template of the recorder.
- **H-Hidden in the data explorer**: When checked this tag will not be visible in the data explorer.

Recipe



#	Type	User Name	Min	Max	Comment	Opt	Recipe	Default	Chart	Stats	OnWrite
1	Local	Top_TSP				----	Ramp				
2	Local	Top_RegMode			Regulation mode: 0_Temp; 1_Power; 2_Current; 3_Power Ratio; 4_Tem	S---	Enumeration				
3	Local	Top_MinValue				----					
4	Local	Top_MaxValue				----					
5	Local	Top_CSP				----					
6	Local	Top_MV				----					
7	Local	Top_RampSP				----					
8	Local	Top_HmiTypeSP				----					
9	Local	Top_Offset				----					
10	Local	Top_RampRun				----					
11	Local	Base_TSP				----	Ramp				
12	Local	Base_RegMode			Regulation mode: 0_Temp; 1_Power; 2_Current; 3_Power Ratio; 4_Tem	S---	Enumeration				
13	Local	Base_MinValue				----					
14	Local	Base_MaxValue				----					
15	Local	Base_CSP				----					
16	Local	Base_MV				----					
17	Local	Base_RampSP				----					
18	Local	Base_HmiTypeSP				----					
19	Local	Base_Offset				----					

Figure 71: Editor types in recipe

If you want this tag to be modified by a recipe, select which editor to show to the user when editing the recipe.

Value:

If the selection is **Value** then the user can enter a value for this tag.
For example, this is used to enter the rotation speed of a manipulator.

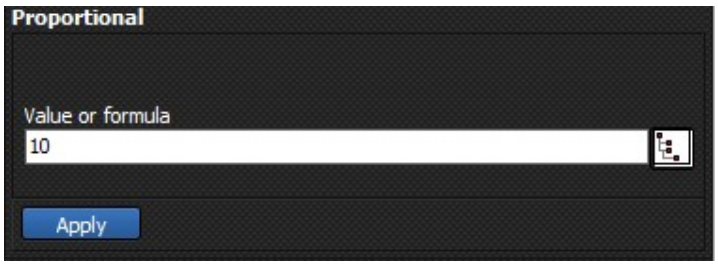


Figure 72: Example of value editor in recipe

Open/Close:

For example, this is used to open or close a shutter:

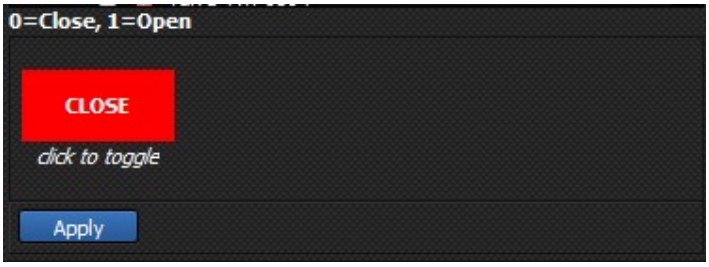


Figure 73: Example of Open/Close editor in recipe

On/Off:
For example, this is used to turn on or off the bake Out:

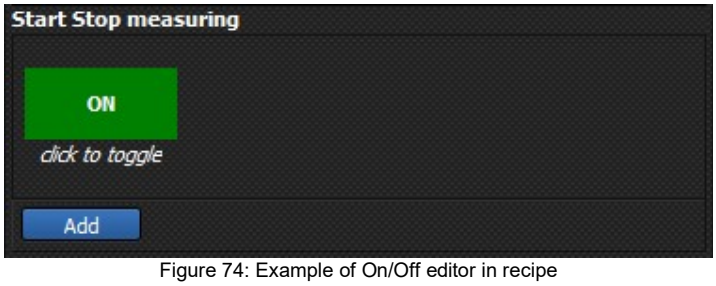


Figure 74: Example of On/Off editor in recipe

Ramp:
Used to enter a ramp set point:

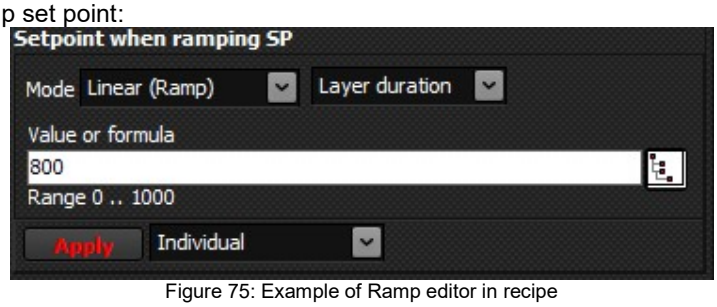


Figure 75: Example of Ramp editor in recipe

Reg Mode:
To change the regulation mode auto or manu:

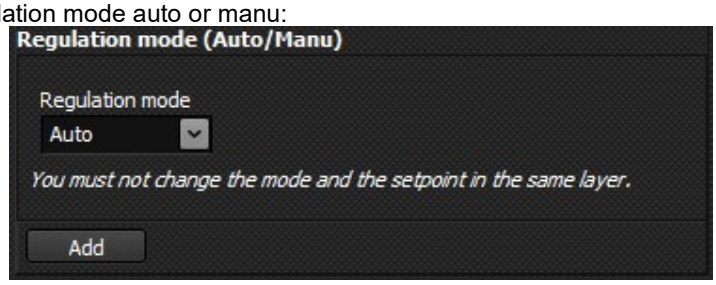


Figure 76: Example of Reg Mode editor in recipe

Enumeration

The enumeration type allows the user to make a choice from a proposed list.
The enumeration type is particular because it also uses the comment of the tag to detail its content.
In this case, the comment must respect a certain format which is the following:
<Label of the tag> **:** <n1>**_**<Label of item1>**;** <n2>**_**<label of item2>**;** etc….

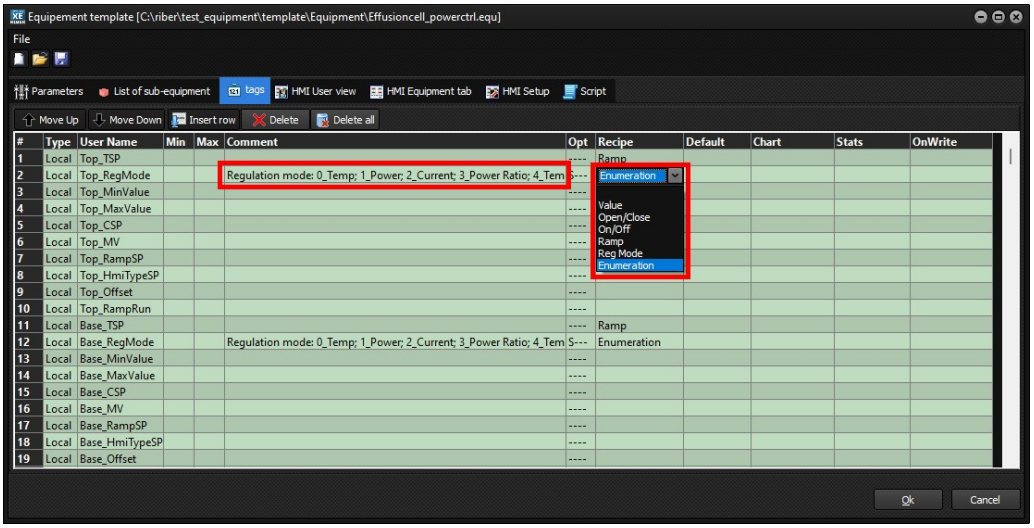


Figure 77: Example of Enumeration in the template editor

In this example, the comment is :
Regulation mode: 0_Temp; 1_Power; 2_Current; 3_Power Ratio; 4_Temp. Offset; 5_Resistance

On the user's side, when entering the recipe, the proposed editor will be this one:

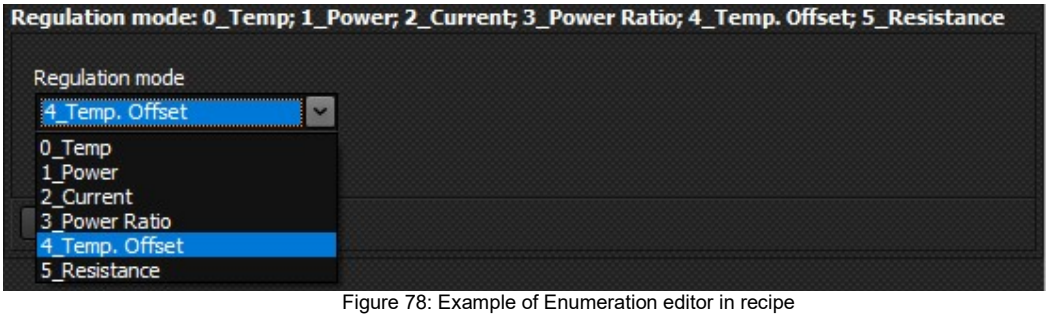


Figure 78: Example of Enumeration editor in recipe

Default

Default value to assign to the tag at startup (if defined)

Chart

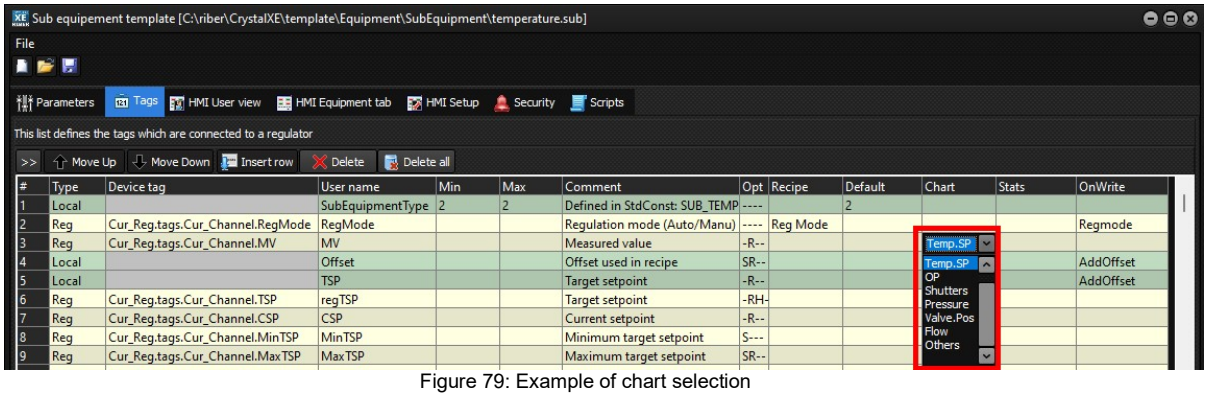
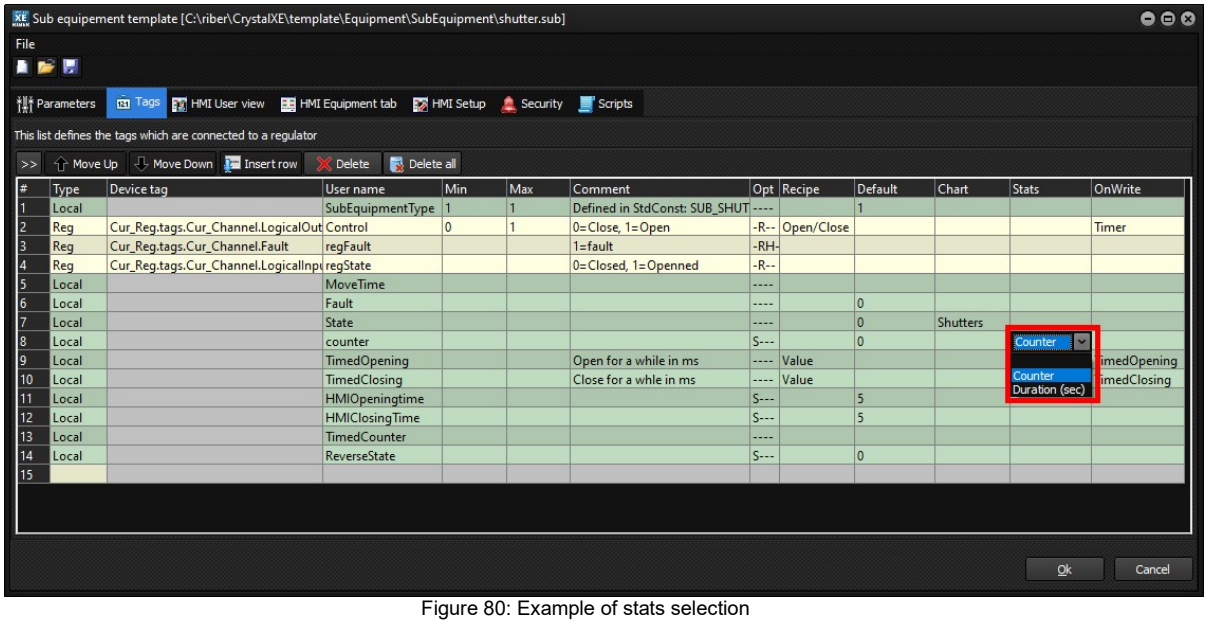


Figure 79: Example of chart selection

If you want this tag to be added to a chart, select with which chart.

Stats



#	Type	Device tag	User name	Min	Max	Comment	Opt	Recipe	Default	Chart	Stats	OnWrite
1	Local		SubEquipmentType	1	1	Defined in StdConst: SUB_SHUT	----		1			
2	Reg	Cur_Reg.tags.Cur_Channel.LogicalOut	Control	0	1	0=Close, 1=Open	-R--	Open/Close				Timer
3	Reg	Cur_Reg.tags.Cur_Channel.Fault	regFault			1=fault	-RH-					
4	Reg	Cur_Reg.tags.Cur_Channel.LogicalInpu	regState			0=Closed, 1=Openned	-R--					
5	Local		MoveTime				----					
6	Local		Fault				----		0			
7	Local		State				----		0	Shutters		
8	Local		counter				S---		0			
9	Local		TimedOpening			Open for a while in ms	----	Value				imedOpening
10	Local		TimedClosing			Close for a whle in ms	----	Value				imedClosing
11	Local		HMIOpeningtime				S---		5			
12	Local		HMIClosingTime				S---		5			
13	Local		TimedCounter				----					
14	Local		ReverseState				S---		0			
15												

Figure 80: Example of stats selection

You can choose to add a tag to the statistics table (Accessible in the Statistics tab of the chamber concerned).
You have two choices:
- Counter: This option will count the number of times the tag will change state.
- Duration: This option will display a duration of activation or use and the user can set a threshold on this duration. When the time is exceeded, an alert will be generated. The activation duration must be calculated in a script of the template.

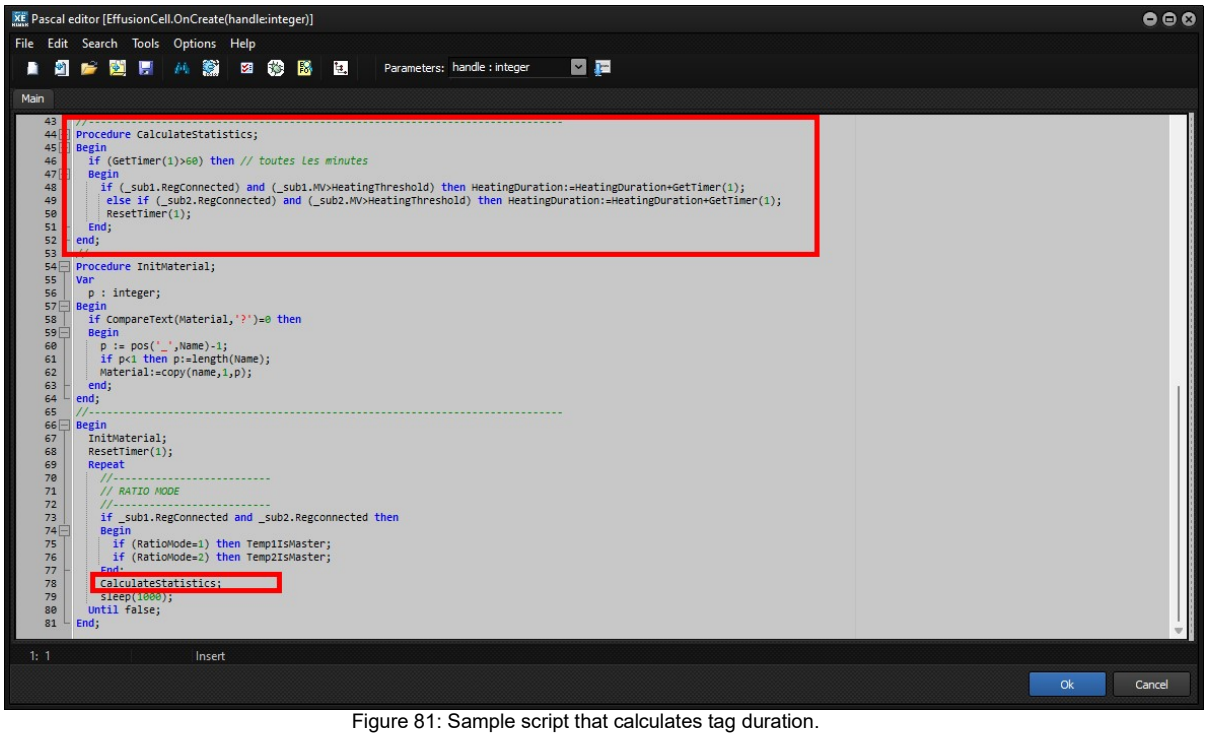
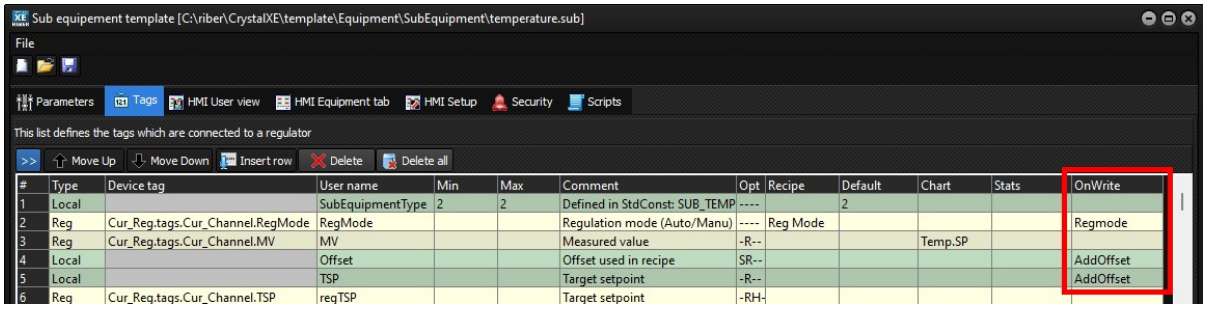


Figure 81: Sample script that calculates tag duration.

OnWrite



To add a new script to the list, see the tab Script and push on the button Add.
If a script is assigned to this cell then it will be executed when writing to the tag. If the tag has several channels then there will be as many instance of the script as channels count.

The parameters oldVal and NewVal can be used in the script to know the previous value before the write occurred and the new written value.

The OnWrite script is executed each time the tag is written (by a script or a recipe or by an object in a HMI setup form)

Depending of the script options, if the script is already running when a new write event occurred then it will restart from the beginning or nothing will append. The script options can be changed in the script editor, in the menu **Options / Script options** and the checkbox **"Wait the end of execution…"**

TAB HMI User

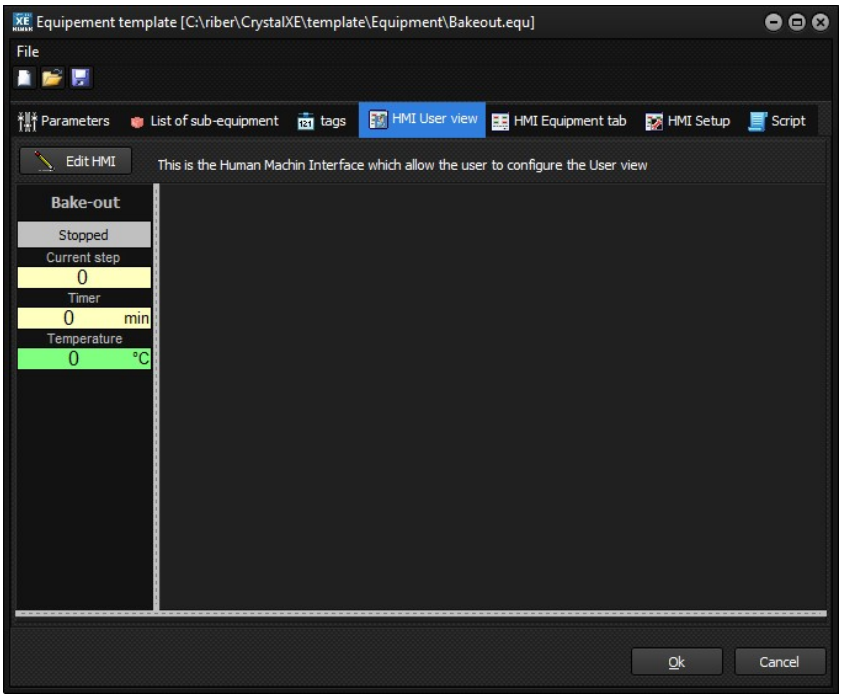


Figure 82: HMI user editor

First, if necessary, resize the active window using the splitters before editing the form.
Then, click on the "Edit HMI" button to modify it.
Visual components can display or modify sub-equipment tags using the "_subx." prefix.
All equipment scripts can access sub-equipment tags using this prefix, but you must ensure that the sub-equipment is connected to a device by reading its RegConnected property.

This HMI is used as a group of object when creating the user view. It can be drag and dropped to the user view.

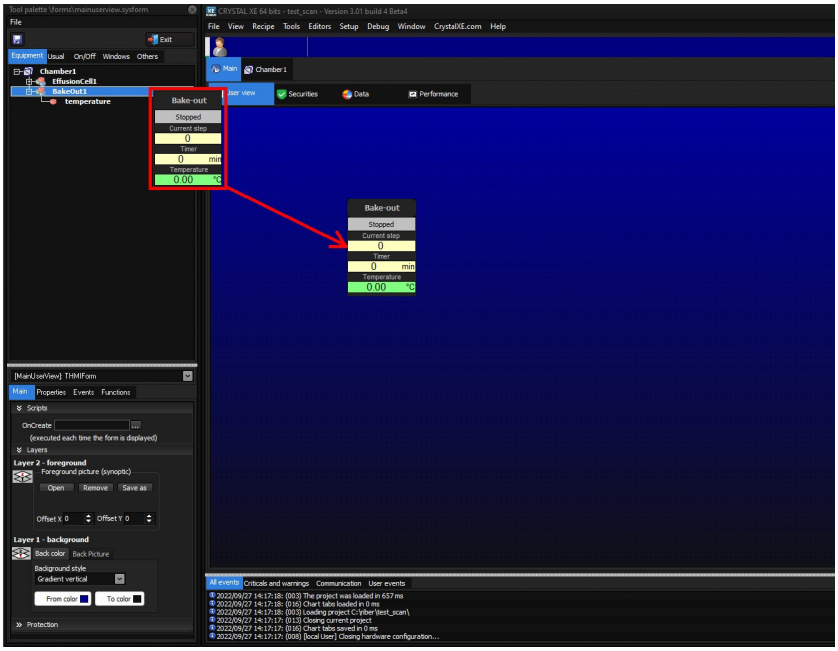
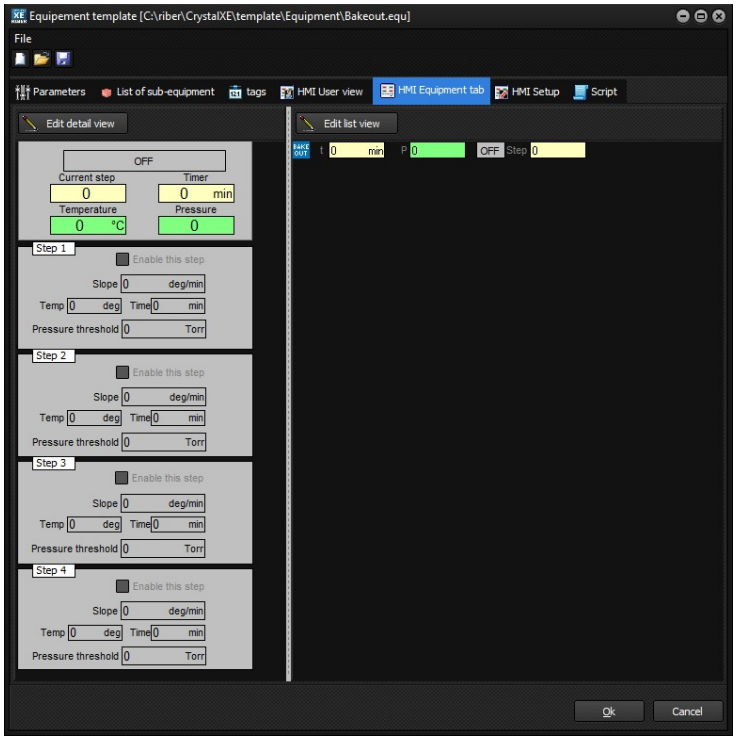


Figure 83: Example of using the HMI User of the backout equipment

→ For more information about HMI, see the part about creating of HMI.

TAB HMI Equipment



The left part of this window is used to define the detail view of the equipment and the center part for the list view.
To edit these HMIs, press the relevant button.

→ For more information about HMI, see the part about programming of HMI.

TAB HMI Setup

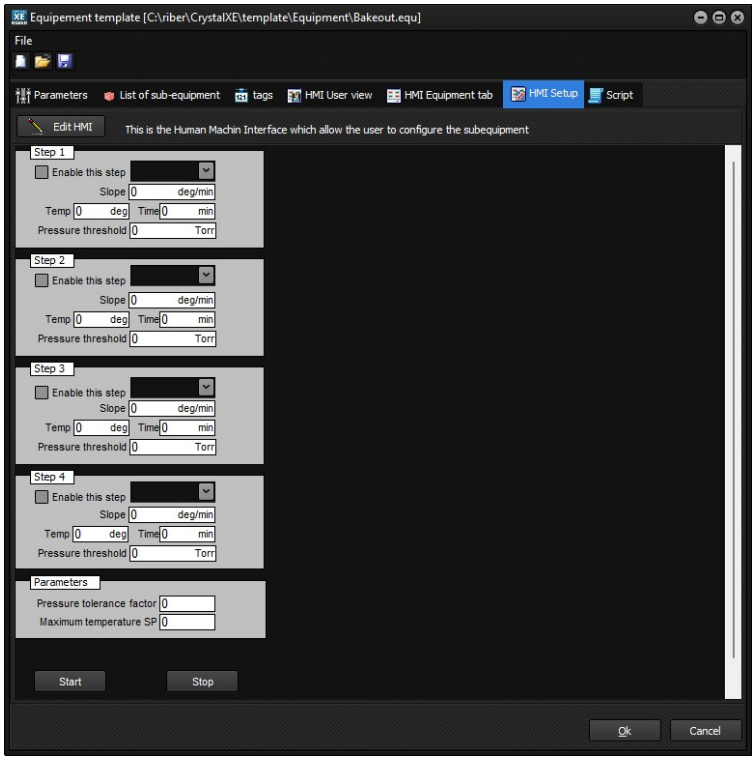
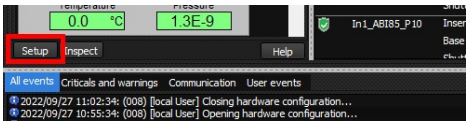


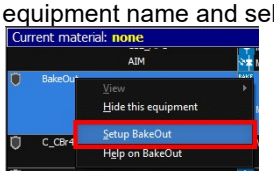
Figure 84: Editing HMI Setup

You can define your own HMI to setup the equipment if needed (optional).

When it is defined, this HMI can be opened by pushing the setup button in the bottom of the detail view.



Or in the list view by right clicking on the equipment name and selecting "Setup <name>"



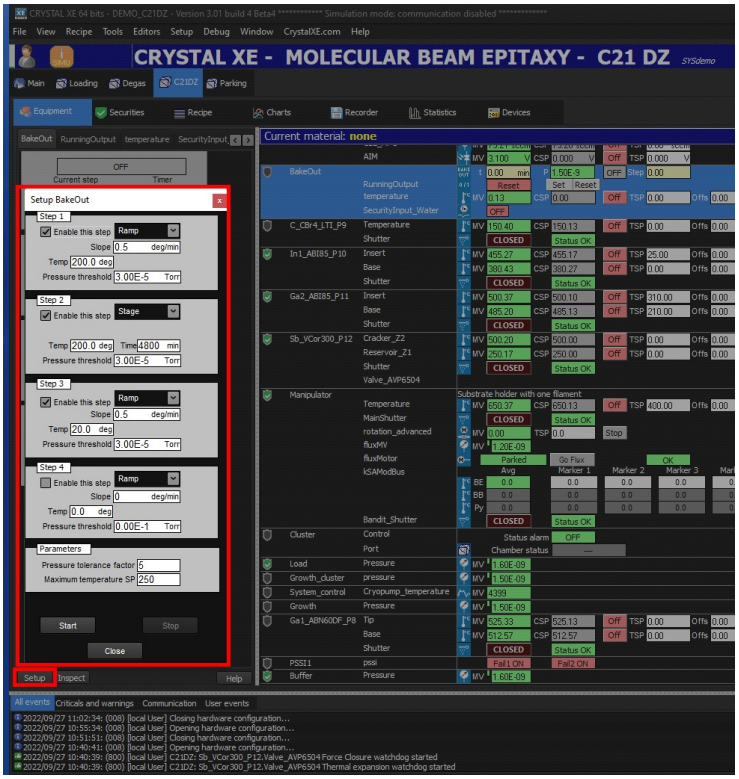


Figure 85: Using of HMI Setup

Note: The “close” button must be added by you when editing the HMI.

→ For more information about HMI, see the part about creating of HMI.

TAB Scripts

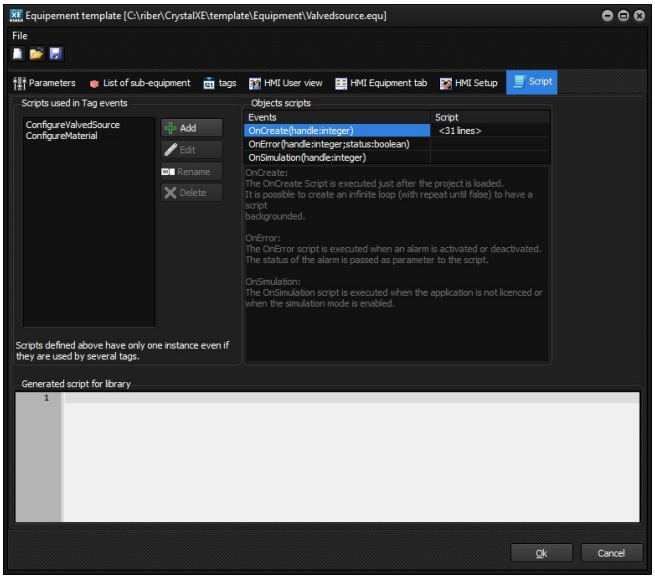


Figure 86: Editing scripts

In the list (Scripts used in Tag events), add all the script needed by the events OnWrite of the tags.
Once a new script has been added to the list, it will be visible in the combo box of the column OnWrite of each tag.
To create a new script, push the **Add** button.
To edit a script, select the script and push on the **Edit** button or **double click** on the script name.
To rename a script and all its references in the tag view, push on the **Rename** button.

To remove a script, push on the **Delete** button.

Parameters used by these scripts:

- **Handle (integer):** Contain an integer in which a byte represents the current equipment number. Refer to the script function GetHandleStr for more details.
- **IdTag (word):** this is the TagId of the current tag.
- **Onload (boolean):** This is true when the tag is loaded at startup from the data file or the default value.
- **OldVal (real):** Value of the tag before a read or a write (depends of the Event which call the script)
- **NewVal (real):** value read of written of the tag (depends of the the event which call the script)

Three other scripts are available:

- **OnCreate** is executed only once when the equipment is loaded after hardware configuration file is opened. If you need a script running all the time in background task then you can use an infinite loop in the OnCreate script like “repeat …. Until false”. In that case, we recommend to add a sleep(xxx) to avoid overloading the CPU. See above to have details about the parameter *handle*.

- **OnError** is executed when the error status changes (when error occurs or disappears). Check the parameter *status* to know if the module is in error (true) or if the error has disappeared (false)
  See above to have details about the parameter *handle*.

- **OnSimulation** is executed when Crystal XE runs in simulation mode.
  When simulation mode is enabled, it is possible to write the tags which are in read only mode. You can write a script to define the behavior of the equipment in simulation mode.

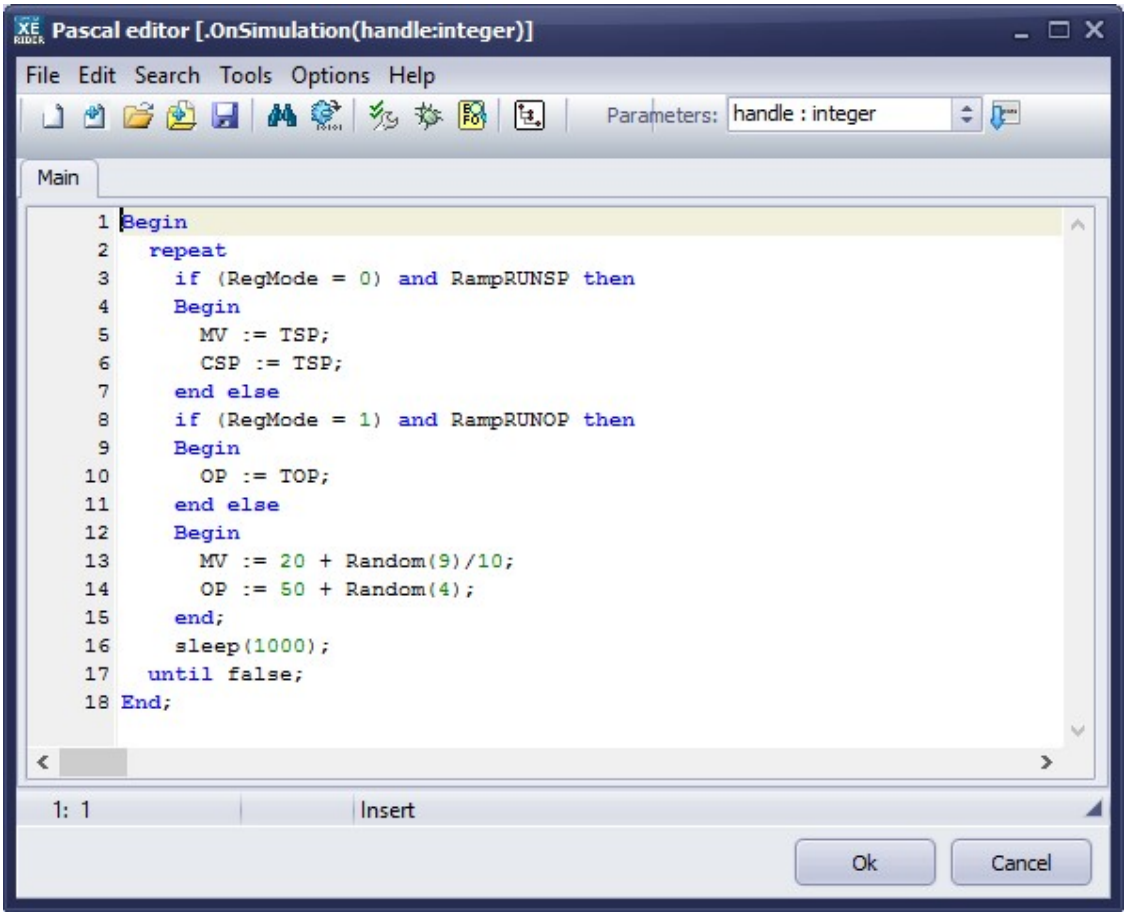


Figure 87: Example of script OnSimulation for the sub-equipment temperature.sub

4.3 Sub-equipment editor

To create new sub-equipment, you may either begin from an empty page or edit an existing sub-equipment and modify it.

To edit an existing template, right click on an item in the palette in the tab “sub equipment” and select Edit template.

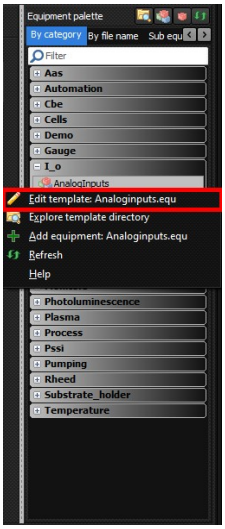


Figure 88: Edit an existing sub-equipment template

To begin a new sub-equipment from an empty page, click on this button

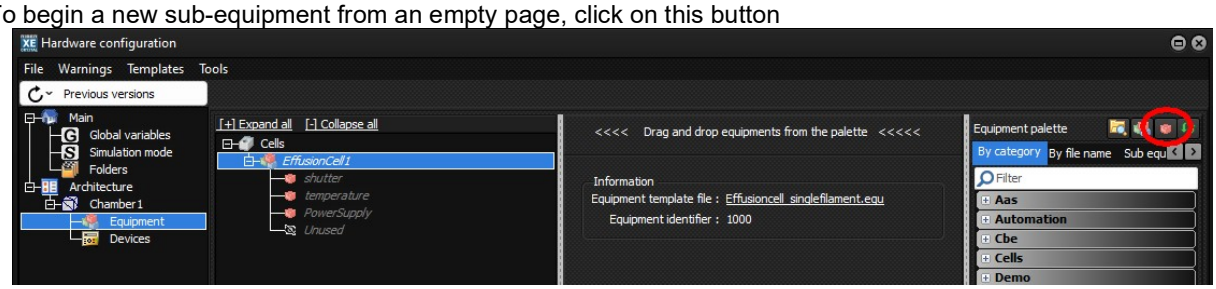


Figure 89: Button to begin a new sub-equipment

The other way is to click on the link of a sub-equipment:

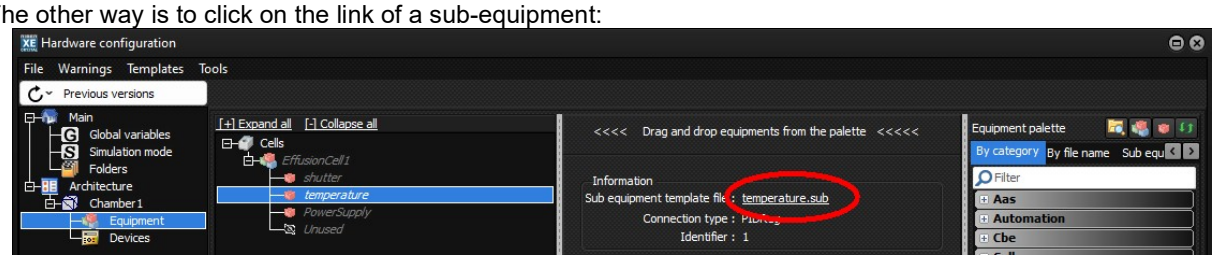


Figure 90: Edit an existing sub-equipment template

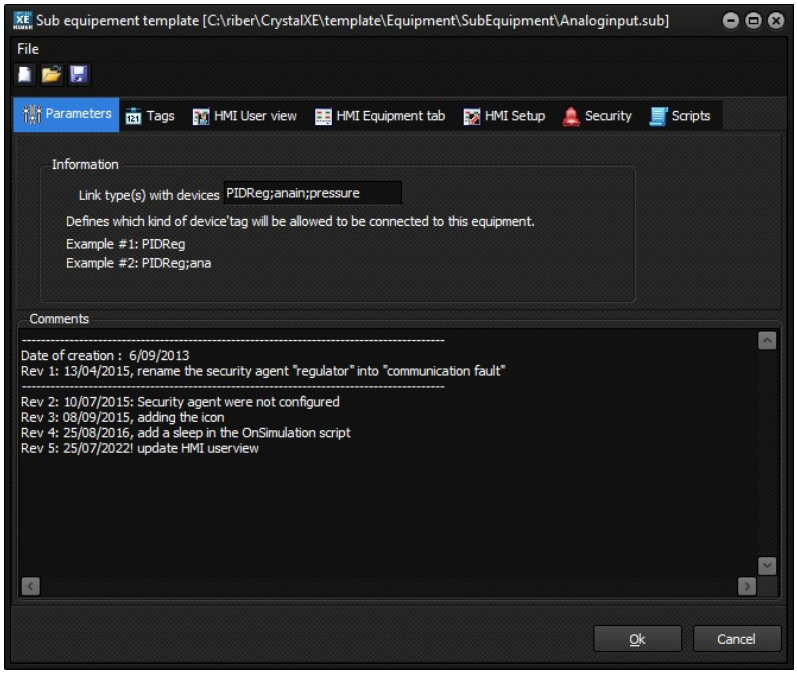


Figure 91: Sub-equipment template editor

Define the link type in the appropriate field.
For more information about the link type, refer to the section Architecture of this document.
Several link type may be entered separated by ';'
For example, the link types for the sub-equipment analoginput.sub is **PIDReg;anain;pressure**

Comments

This is a free text area where we generally indicate the versions and evolutions of the template.

TAB *Tags*

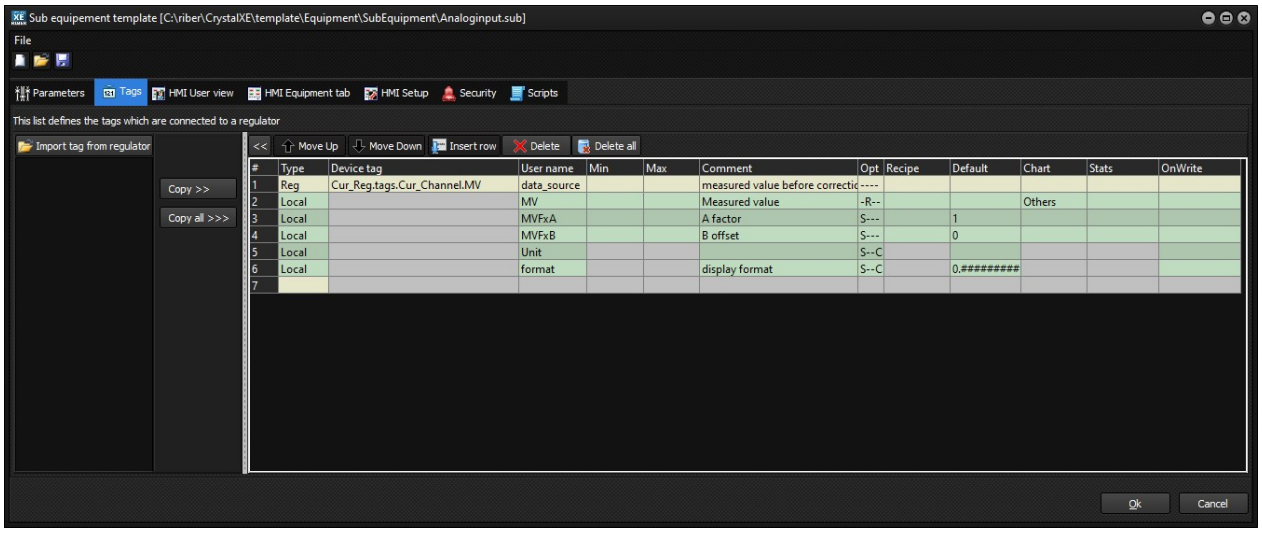


Figure 92: Tags tab in editor

A shortcut can be used to access the tags of the associated regulator (if type=Reg)
To import the tags of a regulator (device), click on the button "Import tag from regulator" and select the file. Then you can copy only selected items or all items. This will generate a link (=shortcut) to the tag of the regulator.

Type

Reg: The tag is a short cut on a device's tag.
Local: The tag is like a local variable.
Extern: The tag is a short cut on a property which will be defined by the user when configuring the project.
. For example, if a valve controller needs to move the valve in case of the temperature increases, then we need to implement an external tag for the temperature reading. This will be a link to another tag. During the configuration, the user will specify which temperature measurement to use for the valve.

Another example, the bakeout need to know the pressure to stop heating when the pressure increases. The next pictures illustrate how to configure the external tag used to read the pressure.

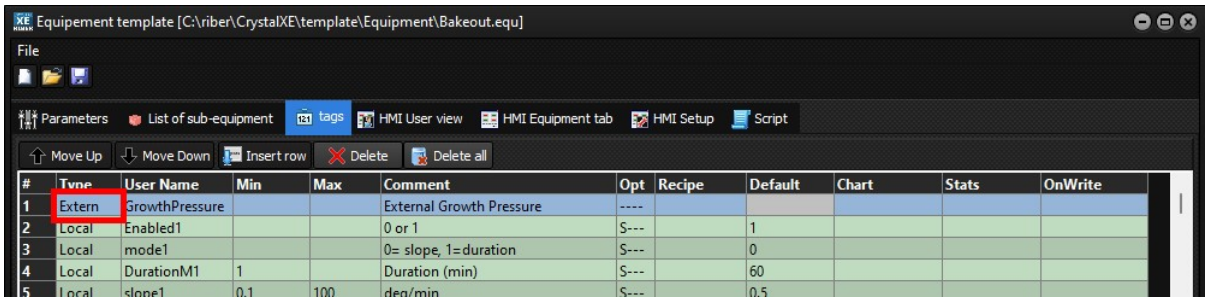


Figure 93: Example of external tag

Then when adding the equipment in the configuration, user will have the possibility to link this tag (see picture below):

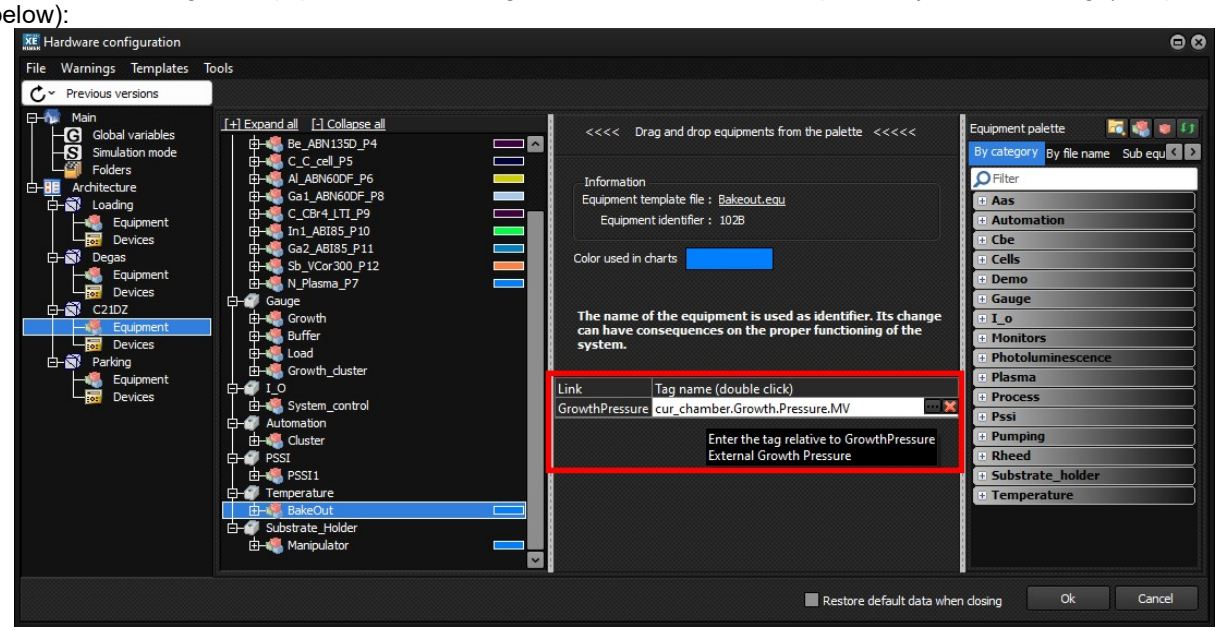


Figure 94: Example of using external tag

For others columns, refer to the column descriptions in the section "Equipment editor".

TAB HMI User

Same as the equipment editor.

TAB Scripts

Same as the equipment editor.

5 PROGRAMMING HMI

Depending of the editor used (equipment, sub-equipment, modbus or ASCII), there are three kind of HMI:

- HMI User view
- HMI Equipment tab, which contains HMI detail view and system view.
- HMI Setup

To display data in HMI you can use objects from the library like the Indicator to display a value, or you can also use scripts.

To display a tag of the current editor, add an indicator and select the tag name as the property.

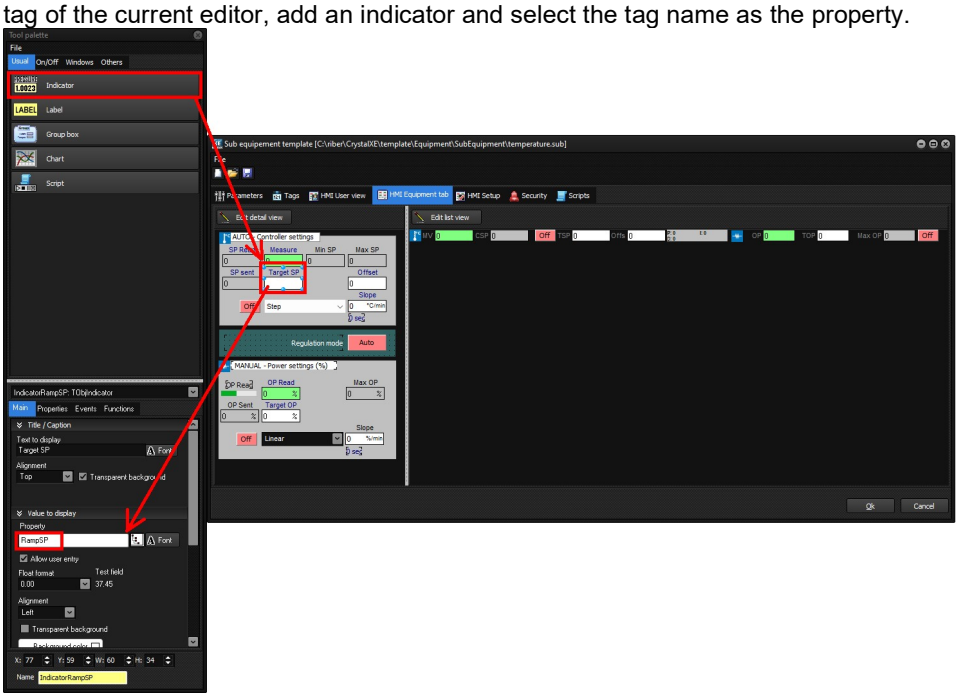


Figure 95: Example to display the tag RampSP using the object indicator

If you edit an equipment and would like to display the tag RampSP of the sub-equipment identified by _sub2 then you have to enter ***_sub2.RampSP***

SCRIPT

To make an animation or to configure the HMI when loading the form then you can write a script in the event OnCreate of the form.

To create or edit a script which runs in background then either you click into the background of the form outside of an object and click on the button […] on the left of the OnCreate field in the tool palette:

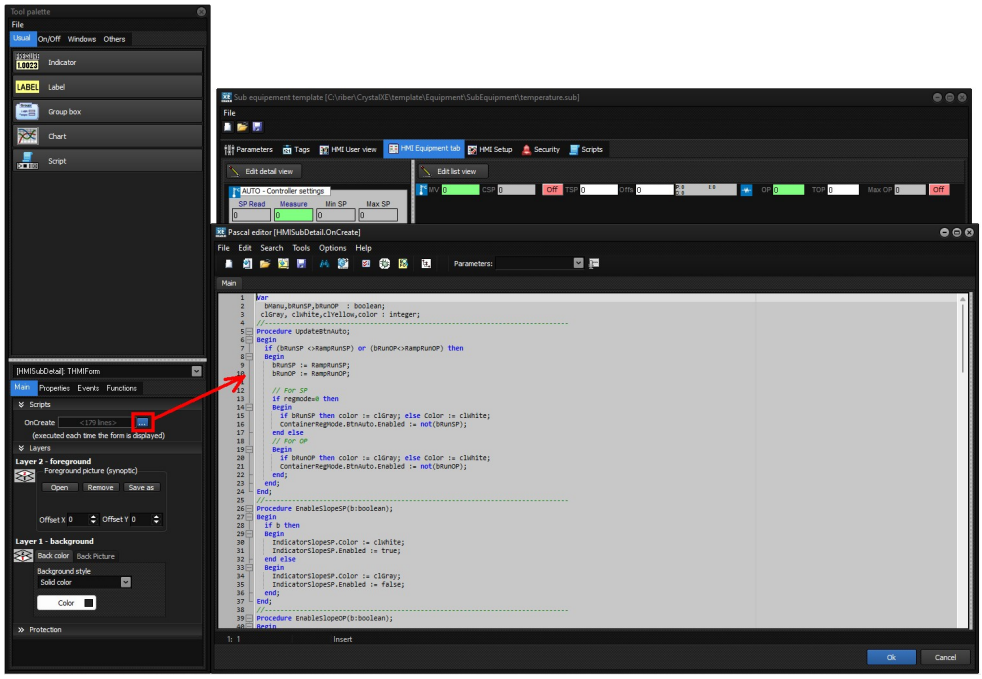


Figure 96: Example of creating a script running in background

In the script, you can use the **name of the object** to access to their properties.
Fro example, to change the color of an indicator: ***Indicator1.color := $FF0000;*** will change the background color of the indicator1 to blue.

In the script, you can use the tag name which is defined in the current editor.

You can also click on the button 
For example, for the tag RampSP you can write something like: **if RampsSP>200 then …**

To have a script running always in the background, you can write an infinite loop ***Repeat … Until false;*** as shown in the picture above. In that case, we strongly recommend adding the sleep function in the loop to avoid any flickering of objects, and to avoid overloading the CPU.

WINDOW SIZE
In all user view, if you want to increase the size of the viewing area then move the splitter **before** you edit the form.

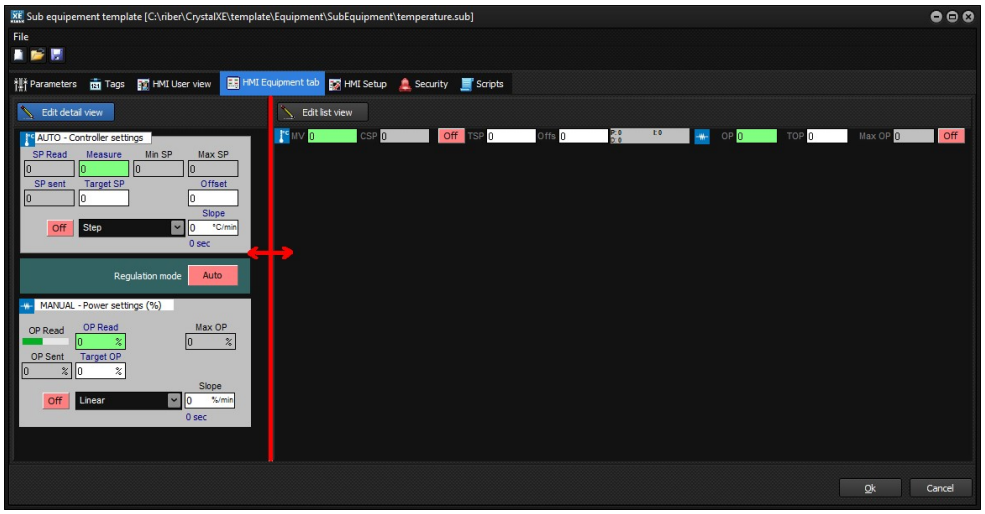


Figure 97: Move the splitter before editing the HMI

6 DEBUG

6.1 CONSOLE

To debug a script, you can use the procedure WriteConsole.
This procedure allows you to output text messages to a console.
The console will be opened automatically when calling this procedure but you can also open it from the main menu: View / Console.

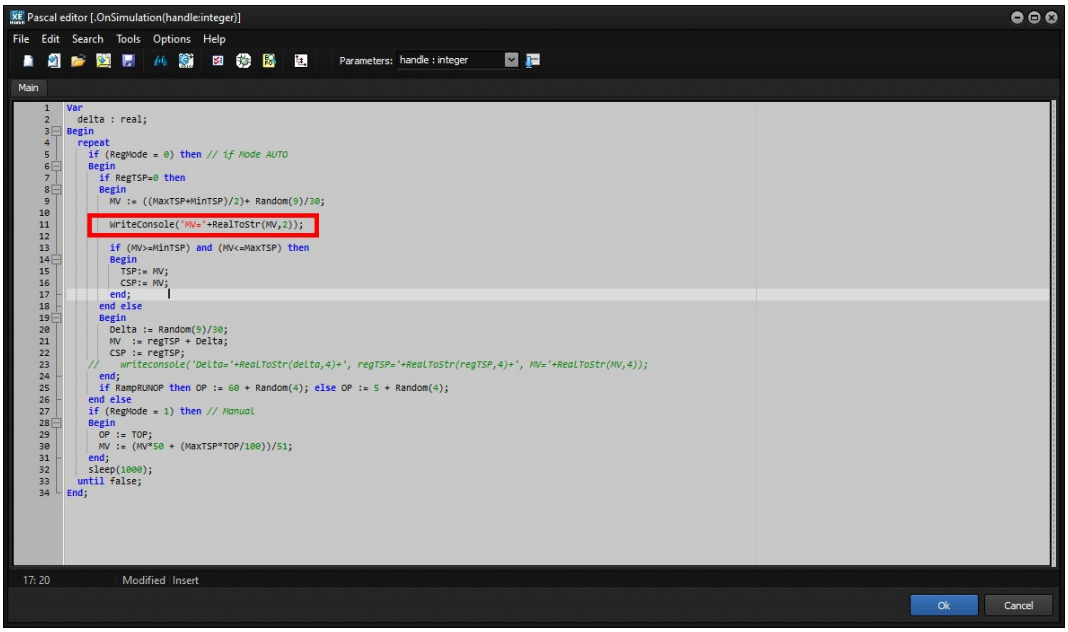


Figure 98: Example of a script using the function WriteConsole

This will result in the console window opening and displaying the messages, as shown below:

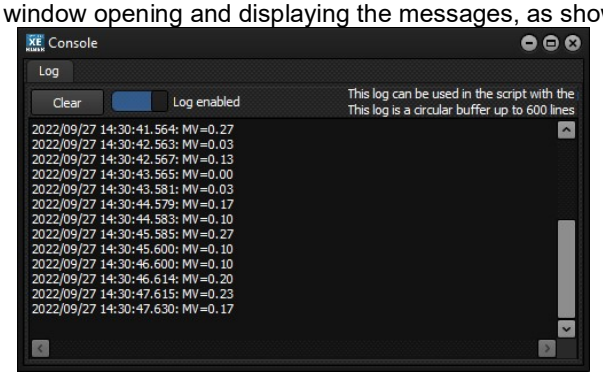


Figure 99: Crystal XE console

6.2 SCRIPT DEBUG

In the main menu of CrystalXE, select Debug/Script explorer

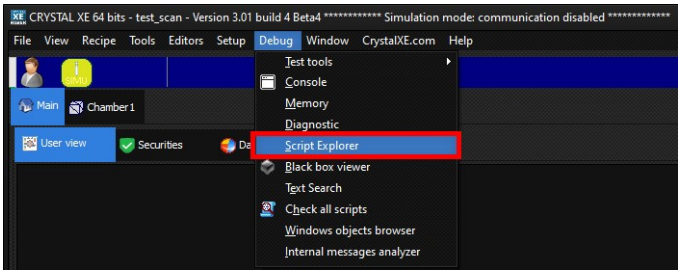


Figure 100: Crystal XE menu to show the script explorer

In the Script explorer window, select the appropriate object and script name and push on the button Show source to follow the execution of the script and display local variables.

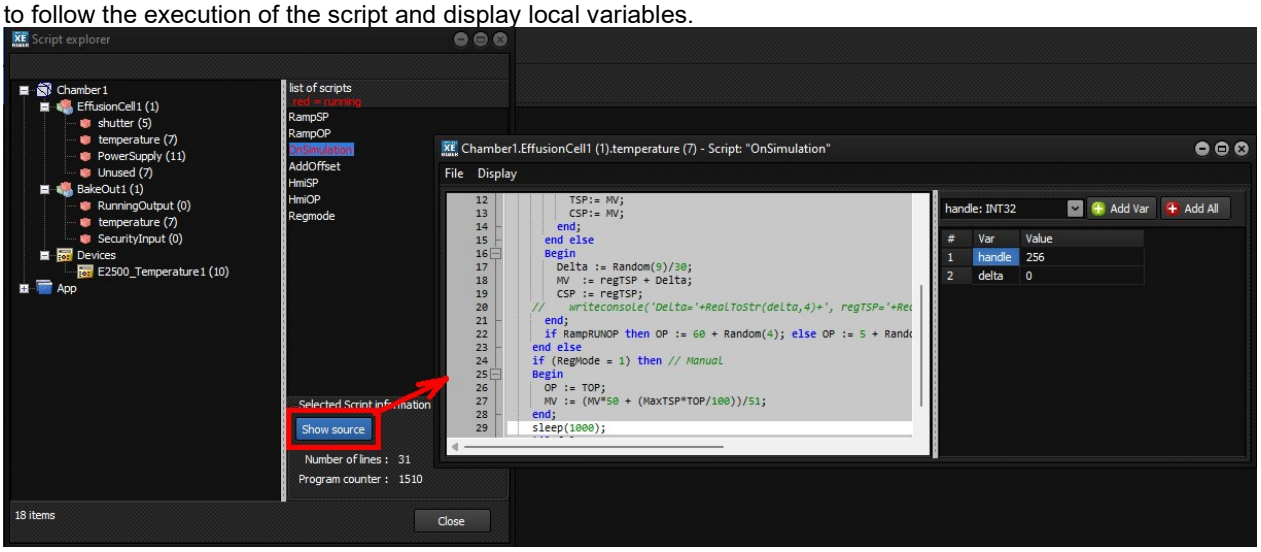
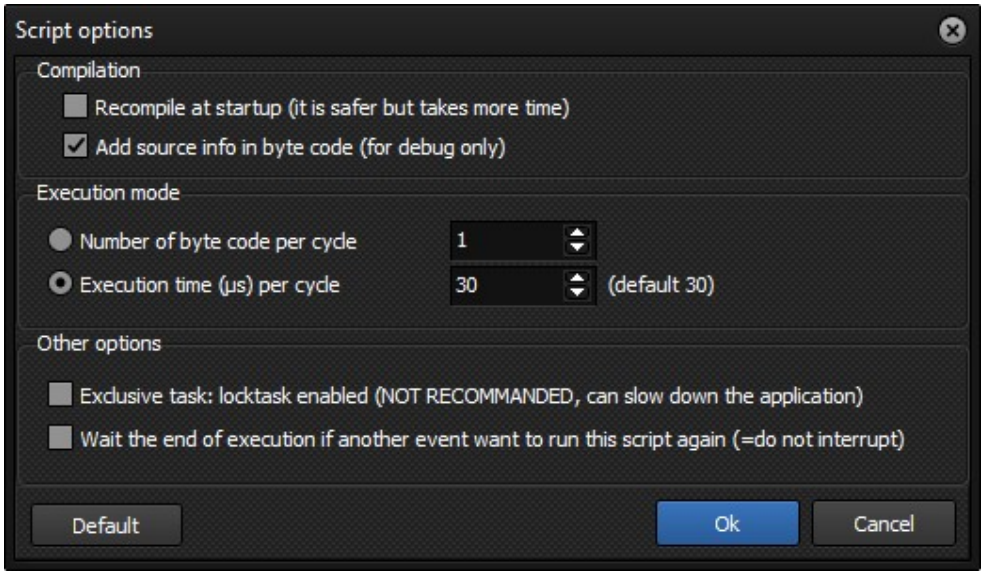


Figure 101: Script explorer window

6.3 TASK PRIORITY - SCRIPT OPTIONS

In the main menu of the script editor, select **Options / Script options**.



The Pascal script is compiled into a binary code (byte code) which is interpreted line by line.
As all scripts are multi-task, the processor executes each script one after the other one.
Only a limited number of lines are executed to share the execution between all the scripts.
By default the execution time is 30 microseconds per script.

If you need to improve the execution of your script, you can change this time but you can also choose to execute a number of bytes code per cycle.

There is an internal time out to avoid freezing the system but it is not recommended to trig this time out.
We don't recommend changing these parameters as they can slow down all the system.

In Crystal XE there are two threads. One if for all script in the forms (User view, tool bar, user floating forms, All HMI in System view like detail view, list view, setup..) and the other one is for all scripts in the devices and equipment and sub-equipment (OnRead, OnWrite, OnCreate etc...) and recipes, this one is faster and priority. So recipe is directly synchronized with the equipment, sub equipement and devices to improve the response time and to run as real-time.

6.4 COMMUNICATION ANALYSER

When a Crystal Application is running, go to the tag Device of the appropriate chamber and double click on a module, or right click and select ***Analyzer***.

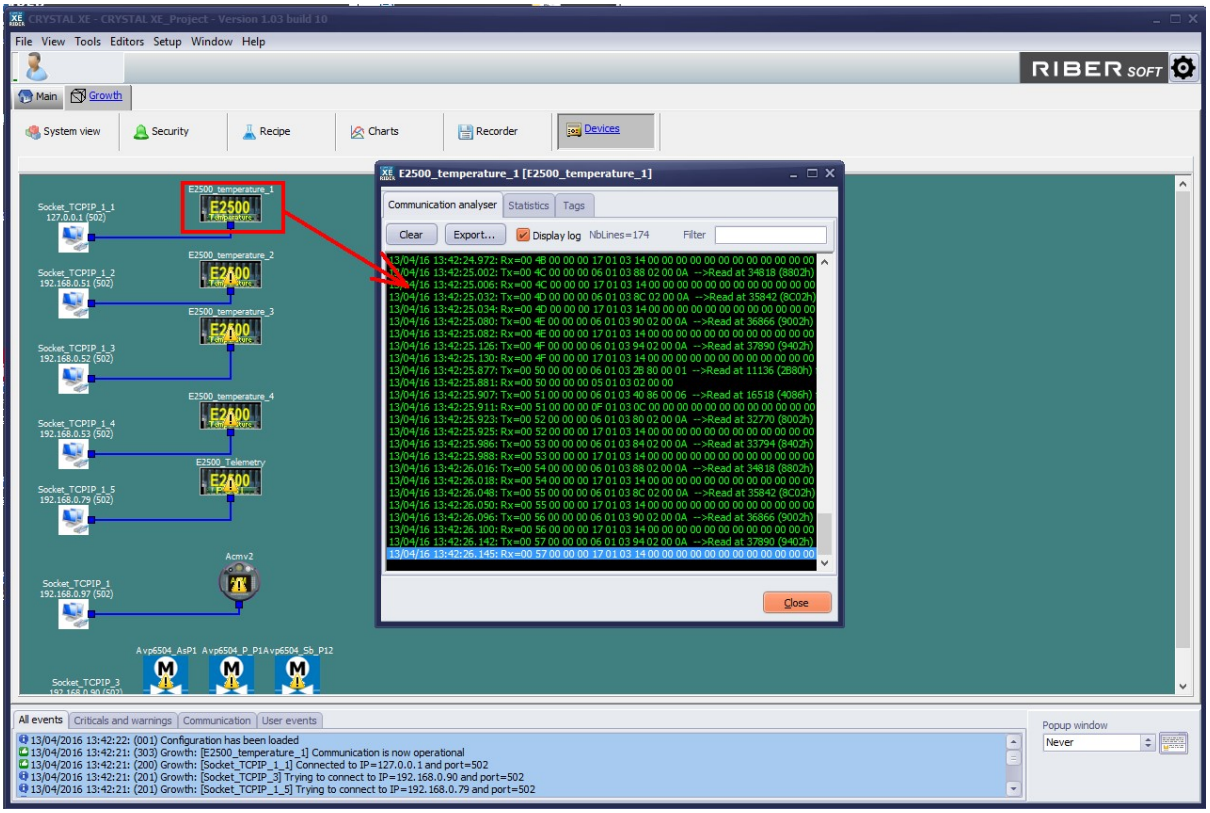


Figure 102: Communication analyzer

As this window is non modal, you can open several analyzers at the same time.

To open the analyzer automatically when CrystalXE is opening, go in the hardware configuration and check this box:

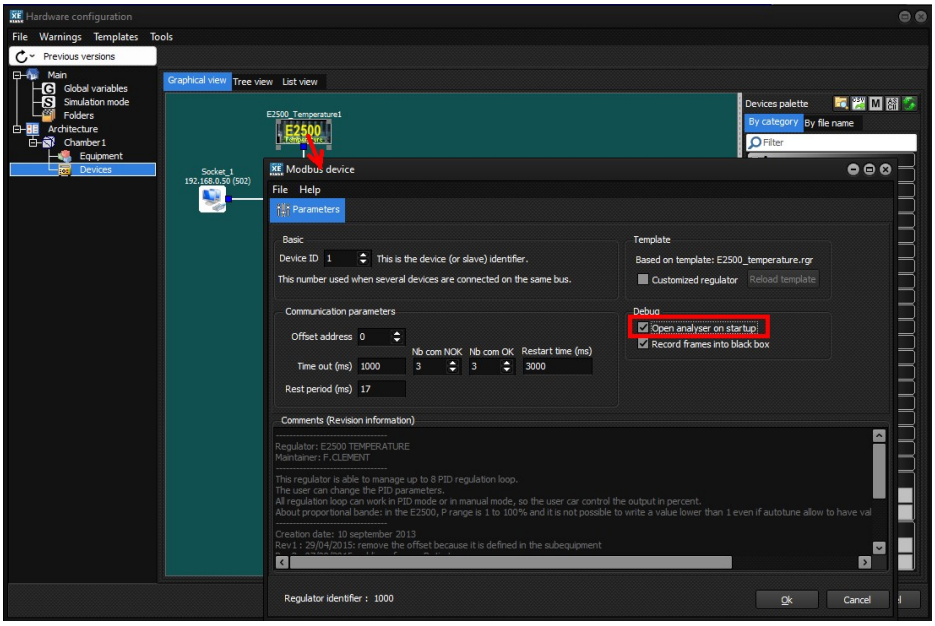


Figure 103: open analyzer on startup

You can add a filter to the analyzer.
In the Filter field, enter a string to be used as a filter.

For example, if we want to display only frames with the string Read:

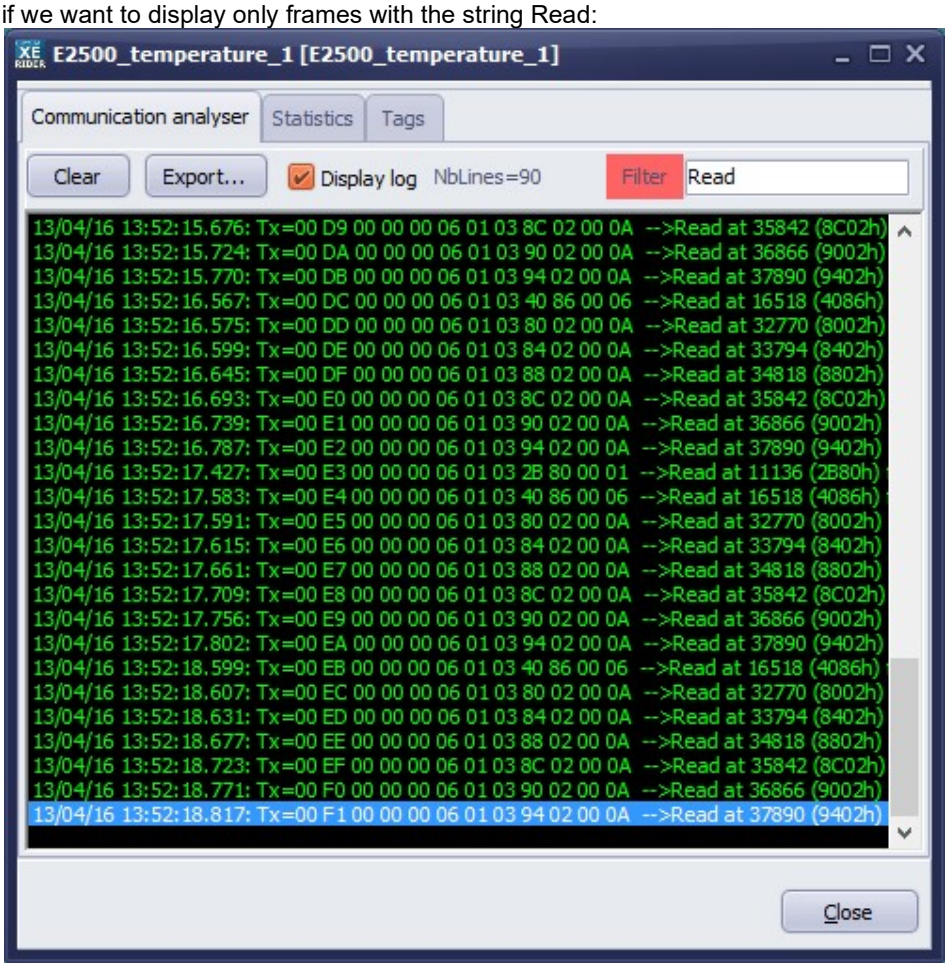


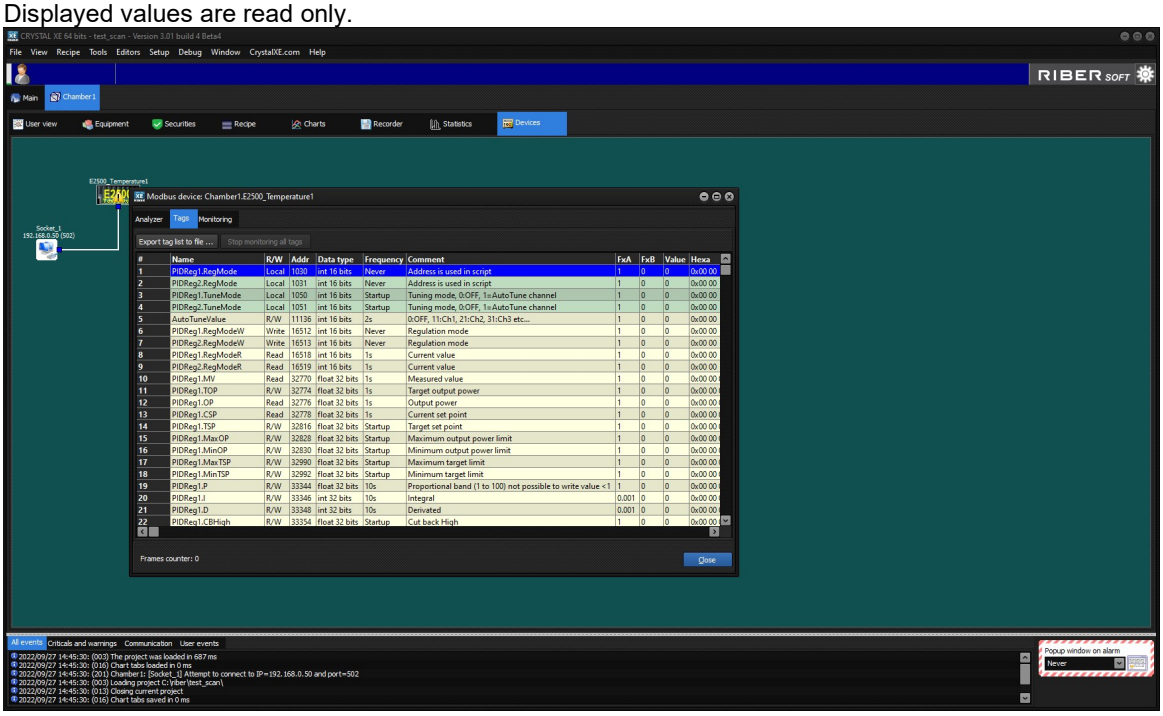
Figure 104: Using filter in the analyzer

6.5 WATCH TAGS

6.5.1 Watch tags of devices

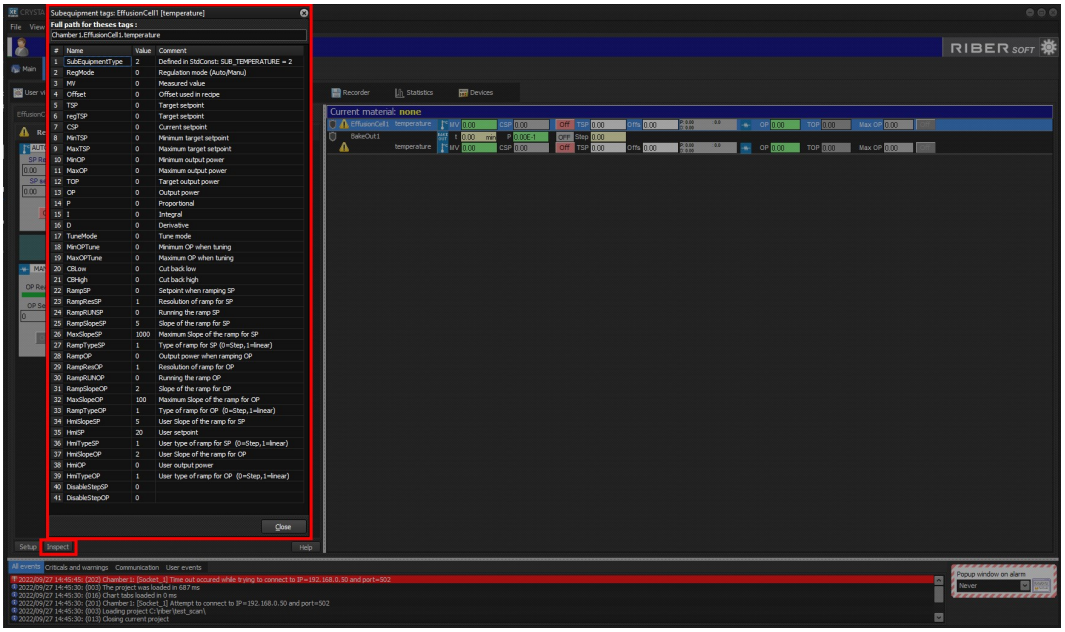
To display values of all tags of a device, double click on the device or right click and select “Analyzer”.
In the analyzer window, click on the tab “Tags”.

Displayed values are read only.



6.5.2 Watch tags of sub-equipment and equipment

To display values of all tags of a sub-equipment or equipment, select the appropriate sub-equipment or equipment and push on the button Inspect. Displayed values are read only.



6.6 BLACK BOX

The black box will record all communication frames of the selected device (all devices by default) to a circular buffer in several files.

→ Use of the Black Box during the normal running of the software is not recommended because it uses hard disk space, memory space and CPU usage.

Enable the black box

To enable the black box, open the *Option* window (Click on the top and right button and click on the *Options* button, or in the menu select *Setup / Options*)

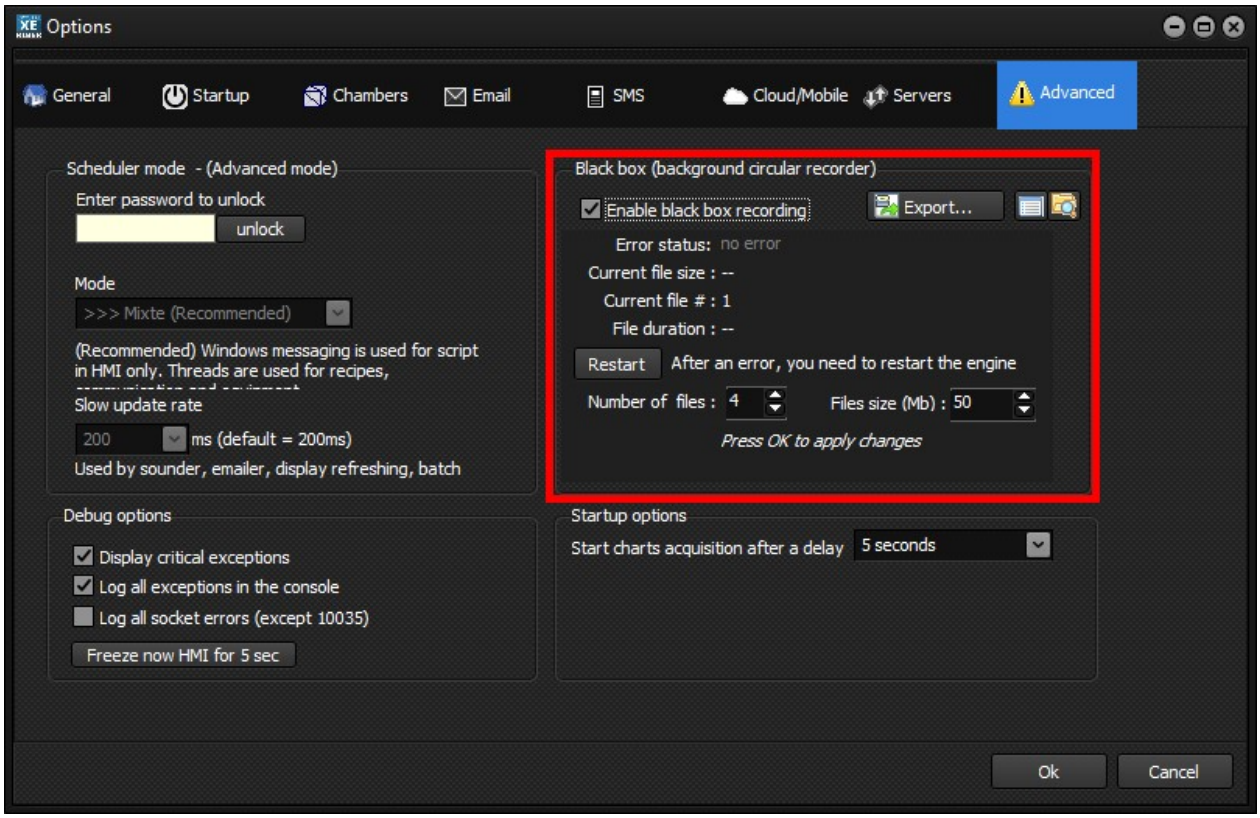


Figure 105: Configuration of the black box

Check the box to enable black box recording.
The black box will save all frames in several files.
The number of files and also the size of each file can be defined here.

View the black box

The data format is binary.
You will need the black box viewer to display the frames.
To open the black box viewer, from the main menu select Tools/Black box viewer or you can also click on the following button in the Options window.

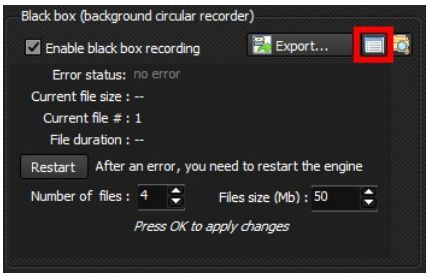


Figure 106: Black box parameters

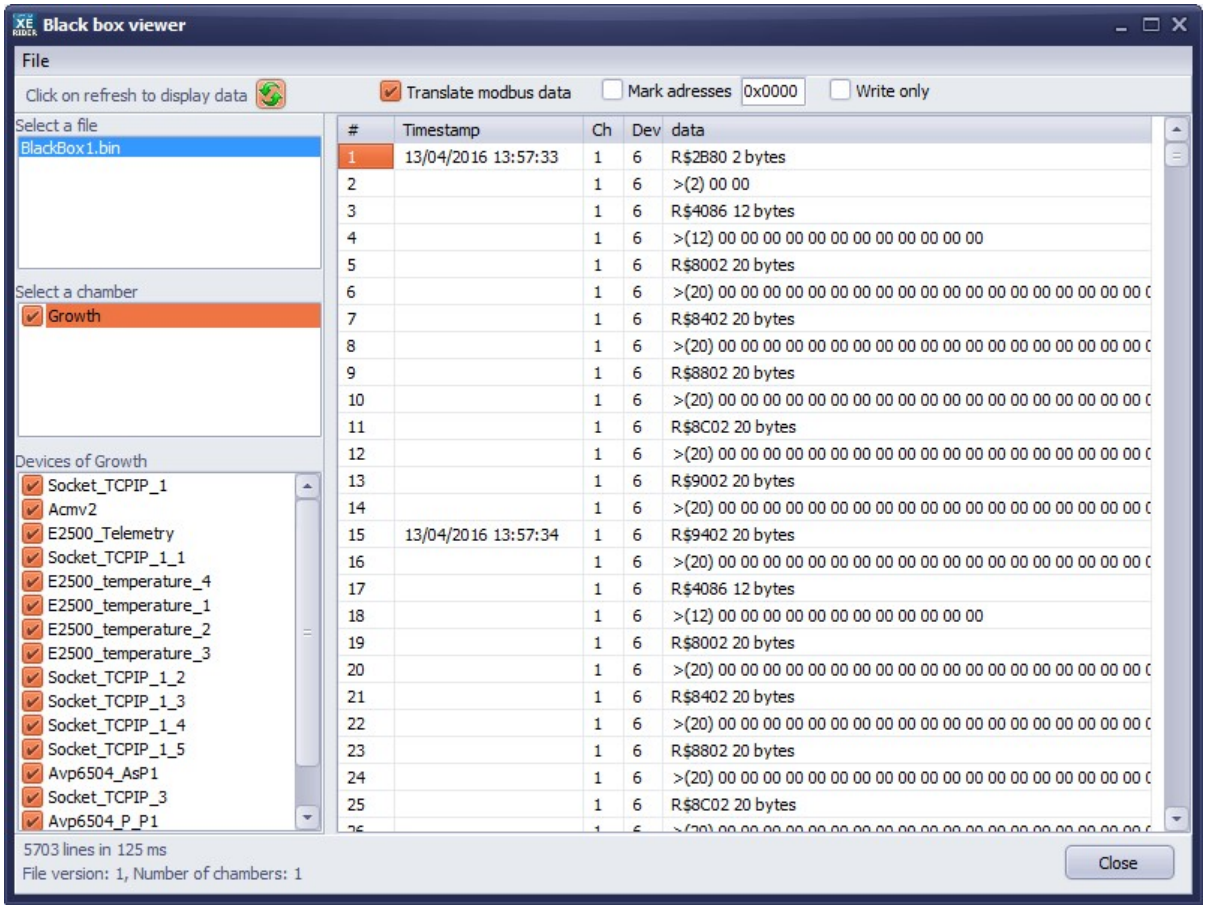


Figure 107: Black box data viewer

Click on the refresh button to refresh the window.

6.7 DIAGNOSTIC

In the main menu, select ***Debug / Diagnostic***

This window informs you about the system and different status of Crystal XE.

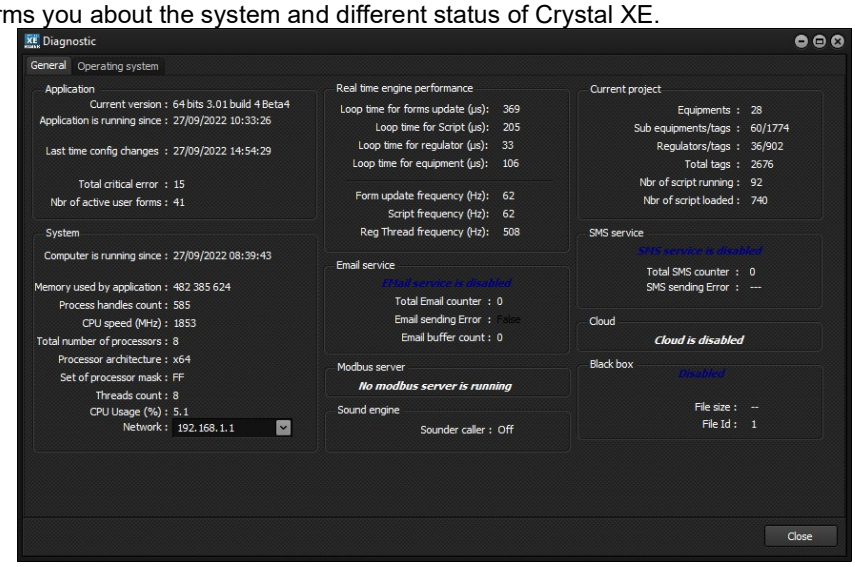


Figure 108: Diagnostic window

6.8 TOOLS

In the main menu, select Tools / Test tools

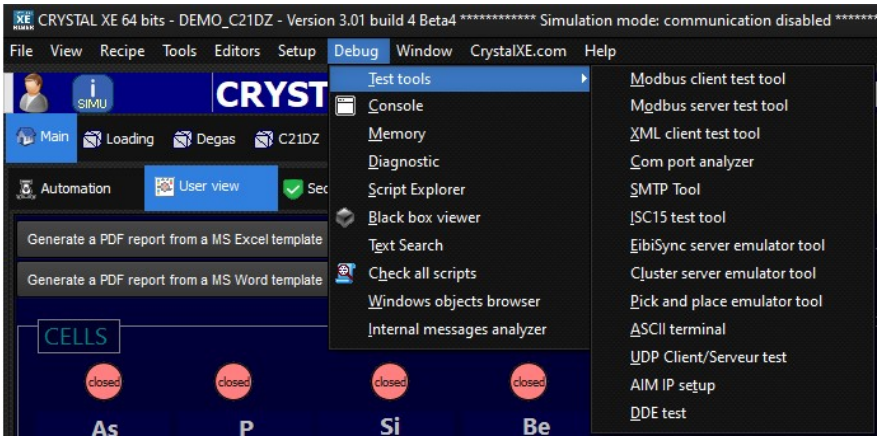


Figure 109: tools

Crystal XE provides a lot of tools which can be useful to check communication with your devices.

Before using it, make sure that the communication port or the socket is not already used by your application.

6.9 FAQ

At startup, Why some tags are written to the device?

In the device template, unselect the option of the tag ***[Write to device at startup]***
If the tag is linked to a sub-equipment, then in the sub-equipment, uncheck the option ***[Save and load value in data file]***