

QUICK START

MasterSCADA 4D. Version 1.3.10

Table of Contents

QS. Lesson 1

- QS. Lesson 1. Creating a project
- QS. Lesson 1. Configuring the system tree
- QS. Lesson 1. Creating the logical part of the project
- QS. Lesson 1. Creating a control window
- QS. Lesson 1. Launch

QS. Lesson 2

- QS. Lesson 2. Configuring a pop-up window
- QS. Lesson 2. Launch

QS. Lesson 3

- QS. Lesson 3. Object-oriented approach to project creation
- QS. Lesson 3. Launch
- QS. Lesson 3. Library operations

The purpose of **Quick Start** (hereinafter referred to as QS) is to help those who are taking their first steps in learning **MasterSCADA 4D** and do not yet have experience working with this environment.

QS. Lesson 1

Let us consider creating a project that implements device on/off control using two buttons — **Start** and **Stop**. When the **Start** button is pressed, the device should turn on. When the **Stop** button is pressed, the device should turn off.

- [QS. Lesson 1. Creating a project](#)
- [QS. Lesson 1. Configuring the system tree](#)
- [QS. Lesson 1. Creating the logical part of the project](#)
- [QS. Lesson 1. Creating a control window](#)
- [QS. Lesson 1. Launch](#)

QS. Lesson 1. Creating a project

To start creating a project, we need to launch the **MasterSCADA 4D** development environment,

by running the default Windows application at `C:\Program Files\MPSSoft\MasterSCADA 4D 1.3\`

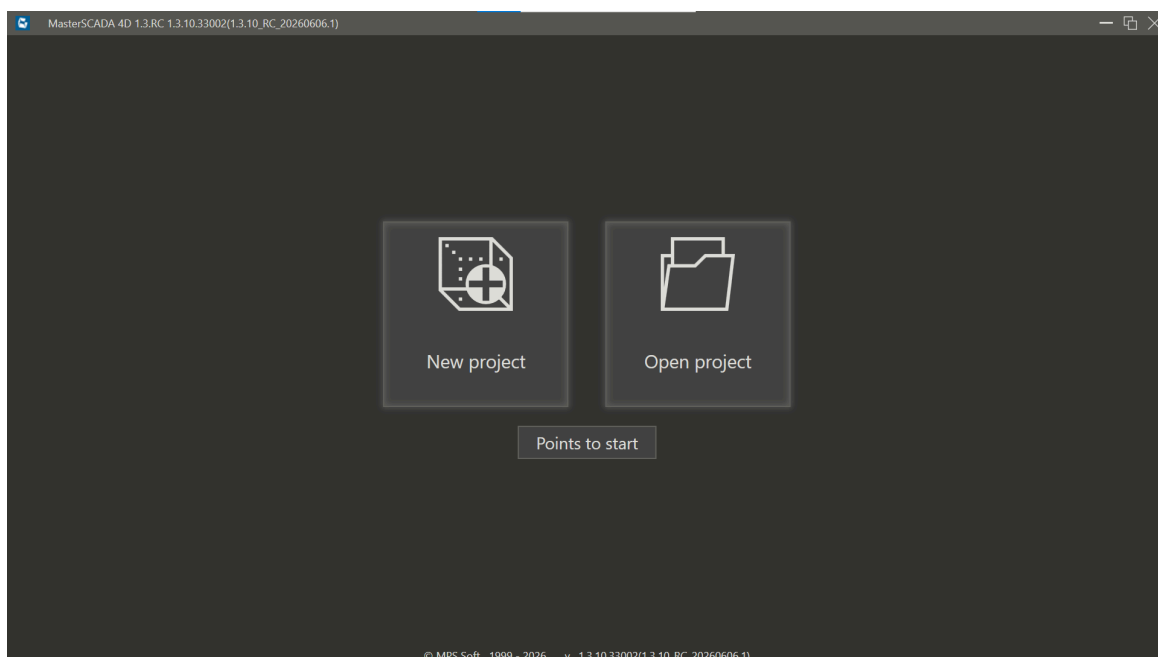


Figure 1

After launch **MasterSCADA 4D** we will be prompted to either create a new project or open an existing one. Let us create a new project.

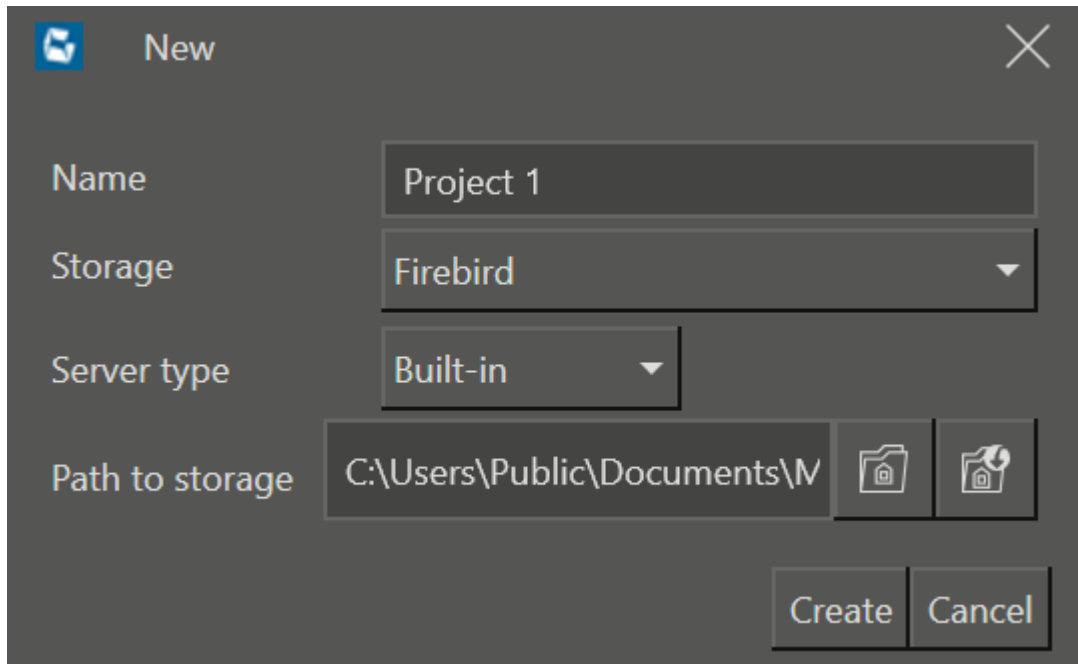


Figure 2

By default, the new project will be assigned the name **Project 1**. We will leave it unchanged — this will be our very first project in **MasterSCADA 4D**.

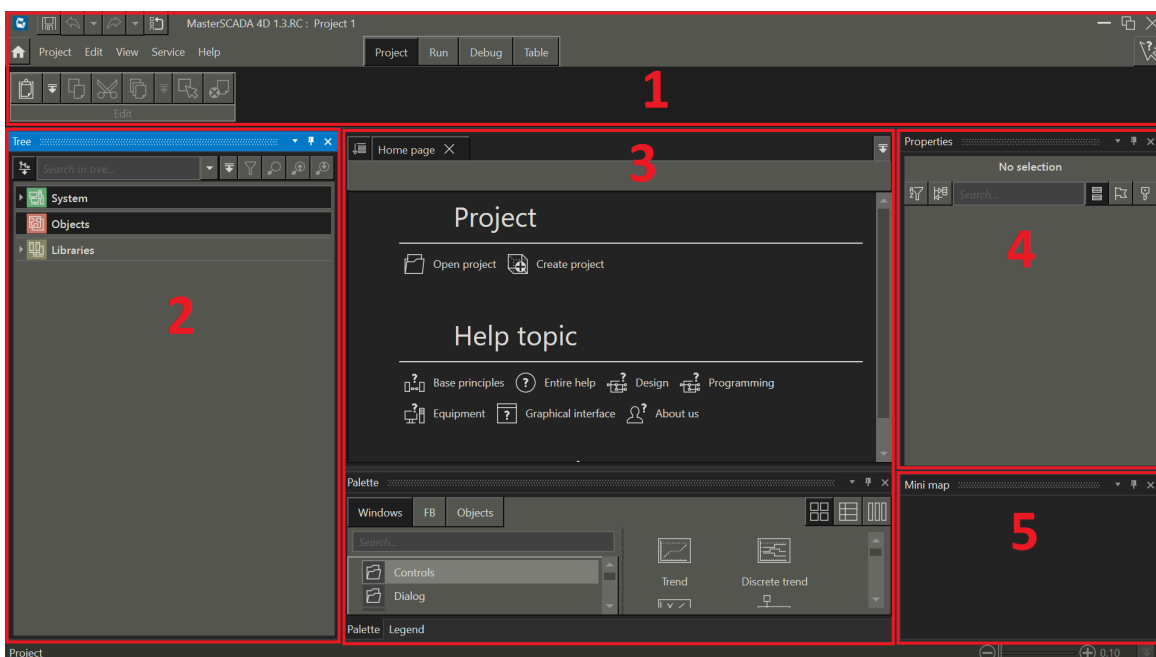


Figure 3

The development environment interface consists of several items. At the top of the interface is the **toolbar** (1). On the left is the **project tree** (2), and in the

center are the **editors panel** and **workspace** (3). On the right is the **properties panel** (4). In the lower right part of the interface is the **Mini-map** (5). For more information about the environment interface, see **interface description**.

After launch, this area displays the start page with icons for opening demo and training projects. This start page can be closed:

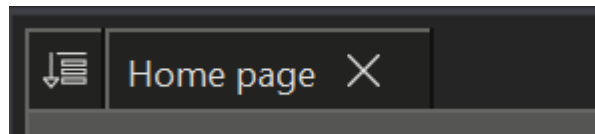


Figure 4

Project tree consists of three parts.

The first part is the **system tree**. It defines the physical configuration of your system, i.e., devices, protocols, and communication channels are specified.

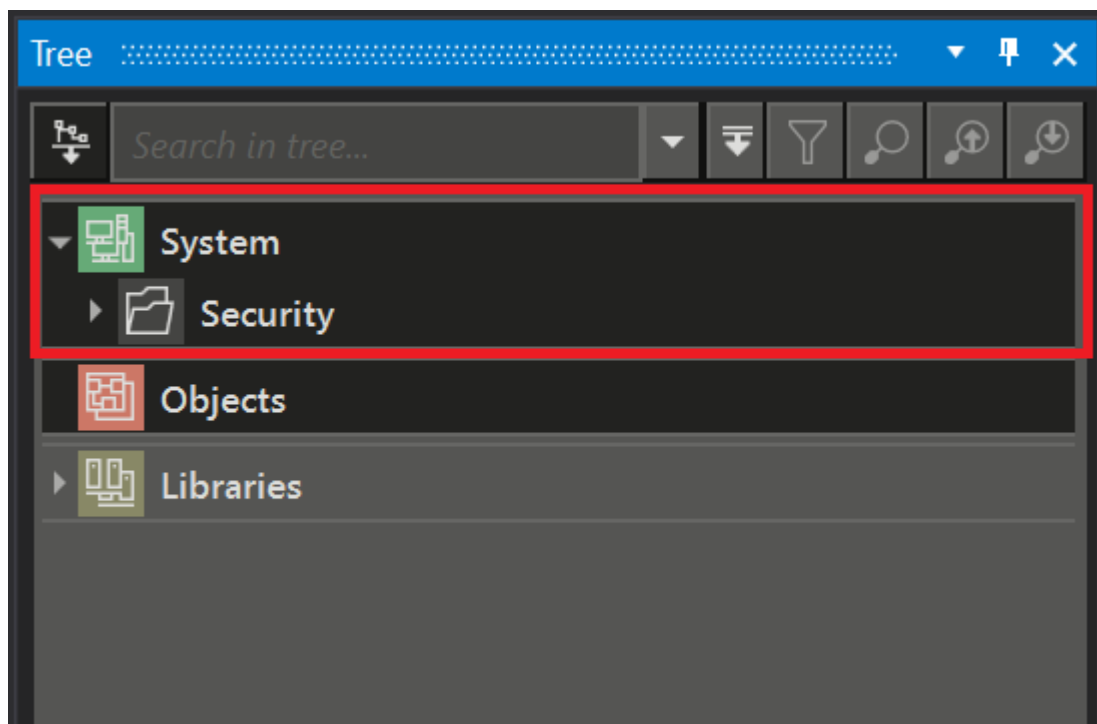


Figure 5

The second part is the **objects tree**. The logical structure of the project is created in this tree. By default, it contains no child items.

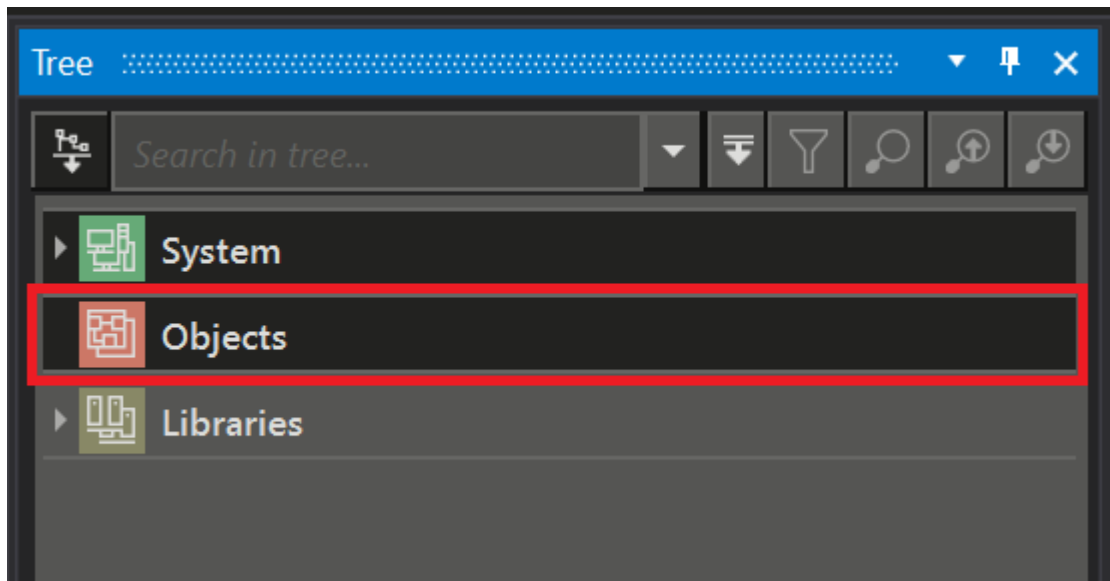


Figure 6

The third part of the tree contains libraries, both those supplied with the development environment and those created by the user. For more information about the **library tree**.

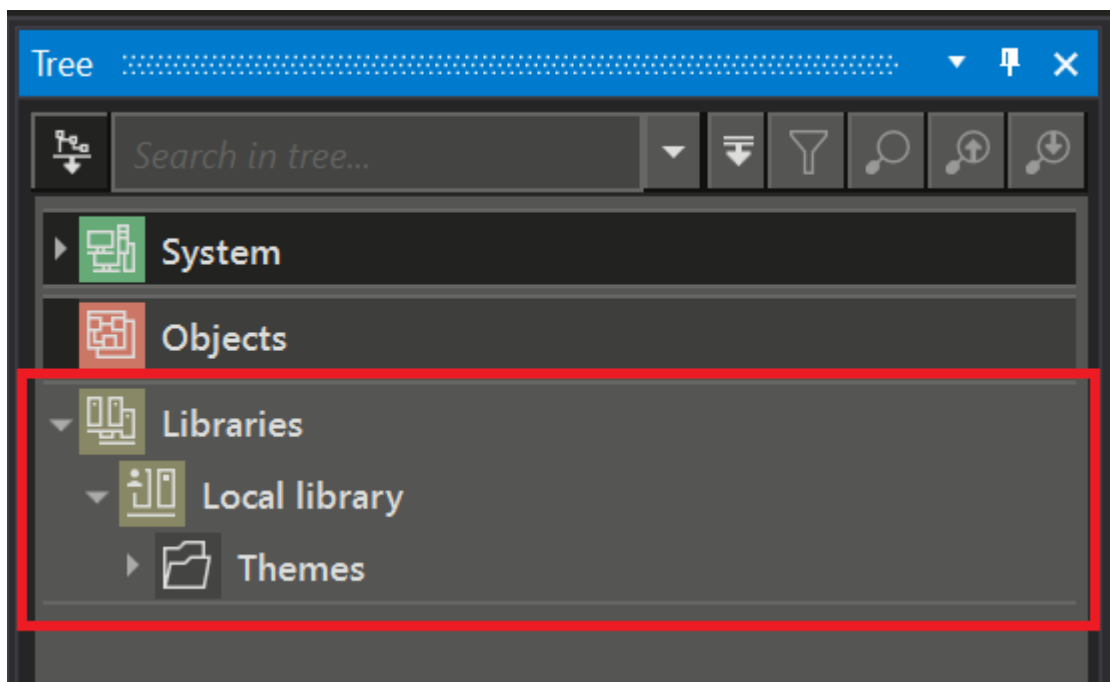


Figure 7

QS. Lesson 1. Configuring the system tree

Let us start building the project by configuring the **system tree**.

To do this, right-click anywhere in the **system tree**, and from the **context menu** select the device on which our project will run. Select **Workstation**.

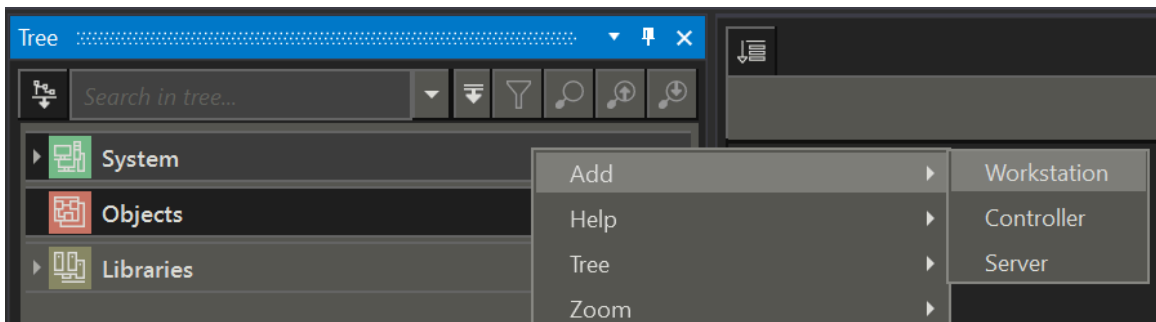


Figure 8

A **Workstation** will be added to the tree, containing its characteristic items:

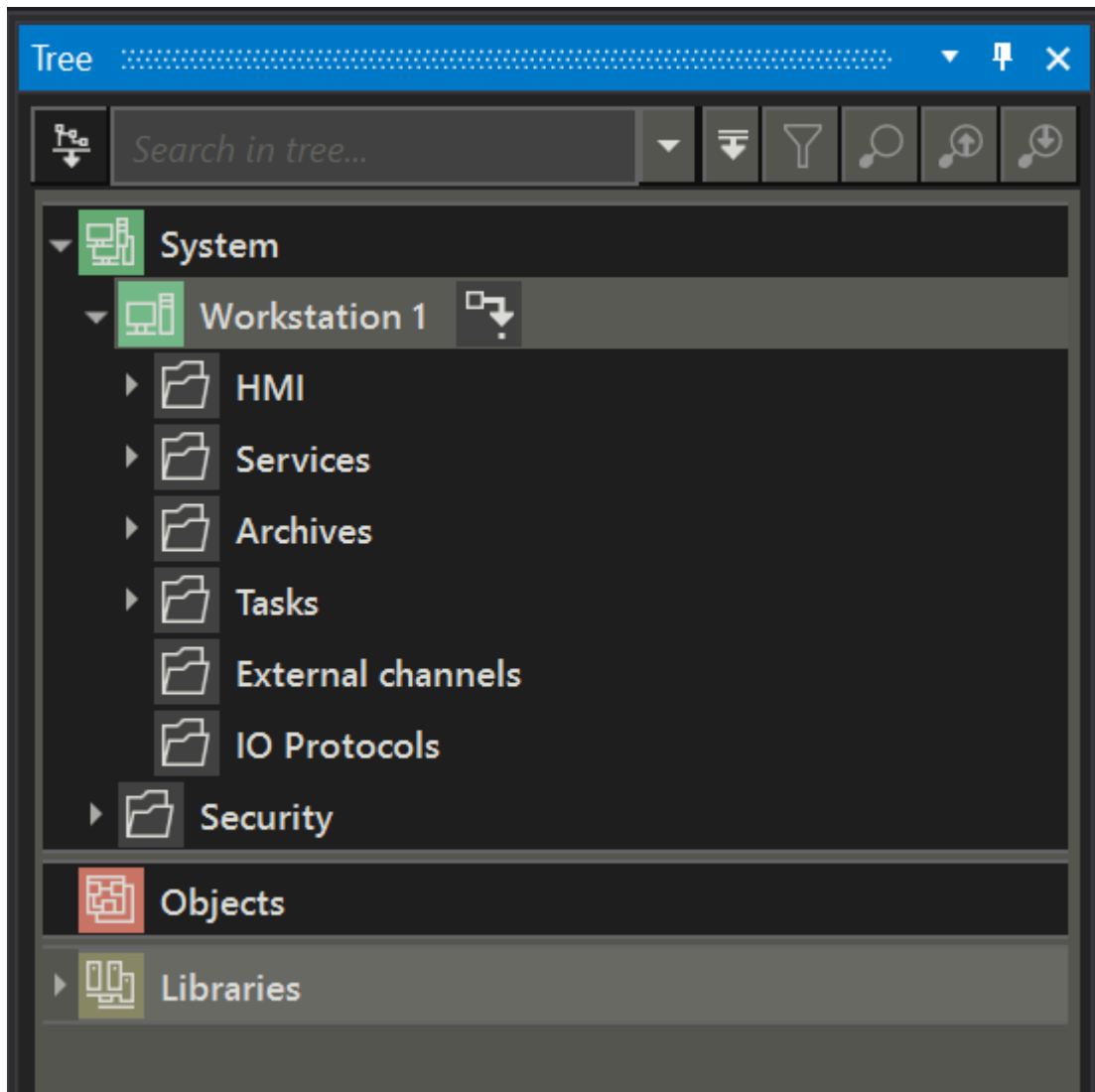


Figure 9 – A Workstation will be added to the tree, containing its characteristic items

For the task at hand, we will need one discrete output to control the device. Some Workstation devices have built-in inputs/outputs. However, we will use an external discrete output module connected to the **Workstation** via a serial port.

To do this, right-click the **Protocols** item and from the context menu select the protocol we will use. In this case, let it be the **Modbus TCP** protocol.

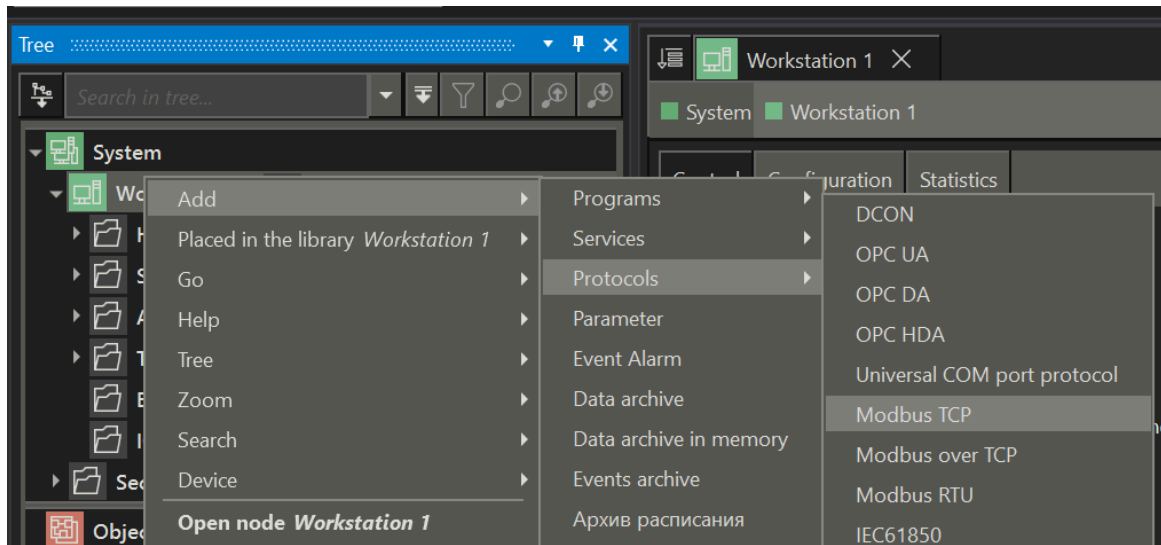


Figure 10

As a result, we get:

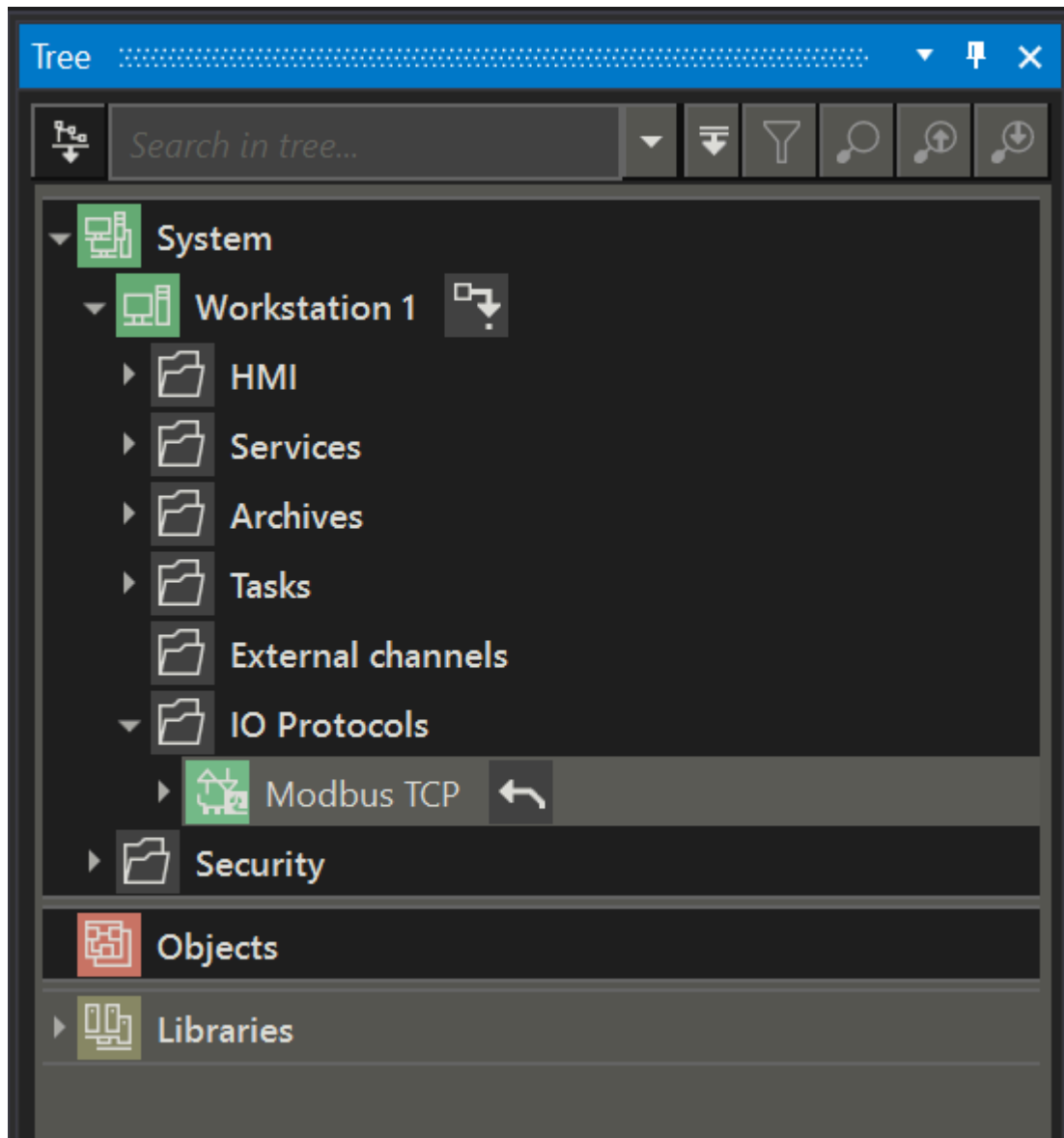


Figure 11 – As a result, we get

The properties panel looks as follows; for more information about properties, see **Modbus TCP protocol properties**.

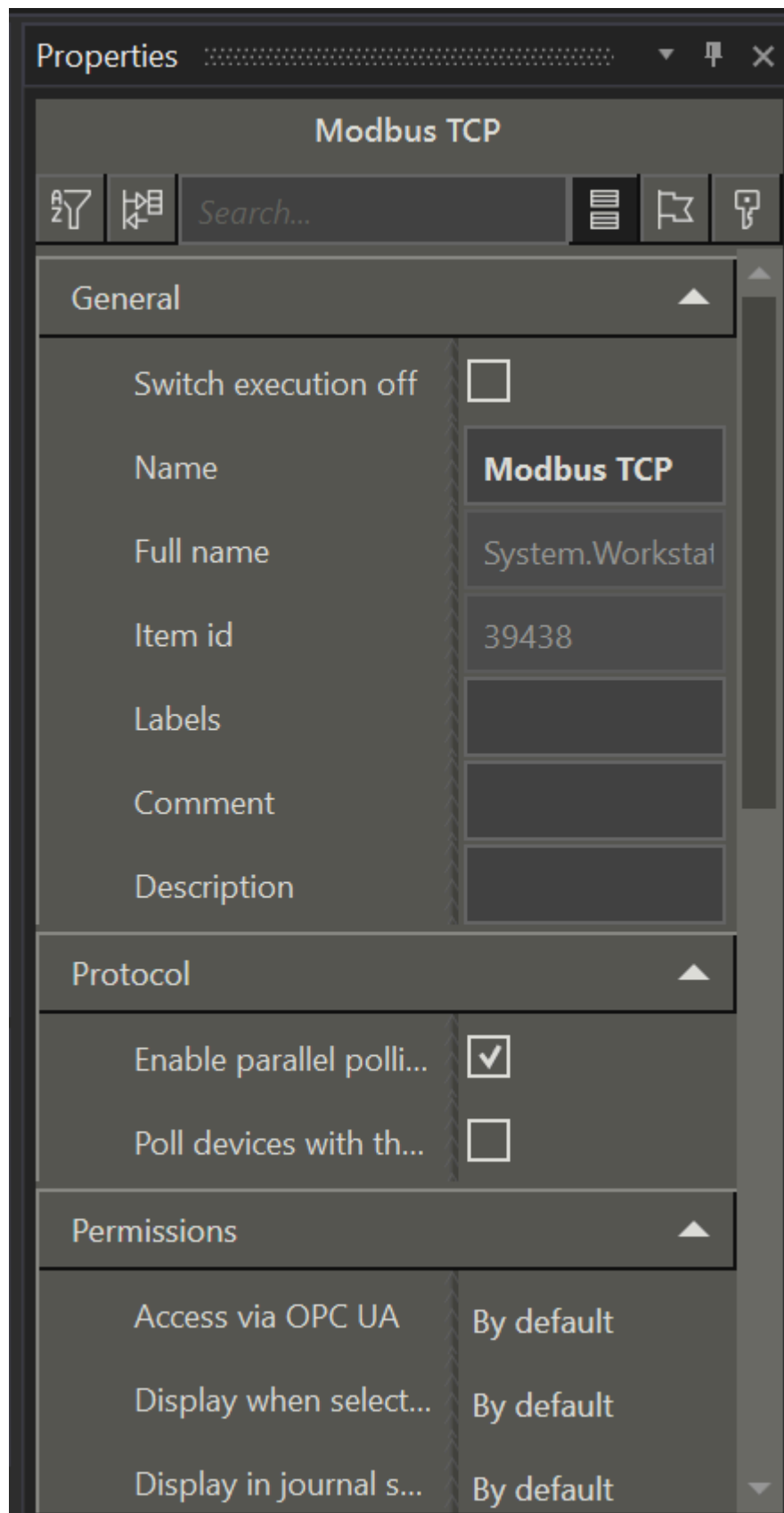


Figure 12

Next, we need to specify the module itself.

Right-click the **Modbus TCP** protocol and from the context menu select **Add - ModbusTCP Device** - this is the discrete input/output module that we will control. For more information about **ModbusTCP Device**.

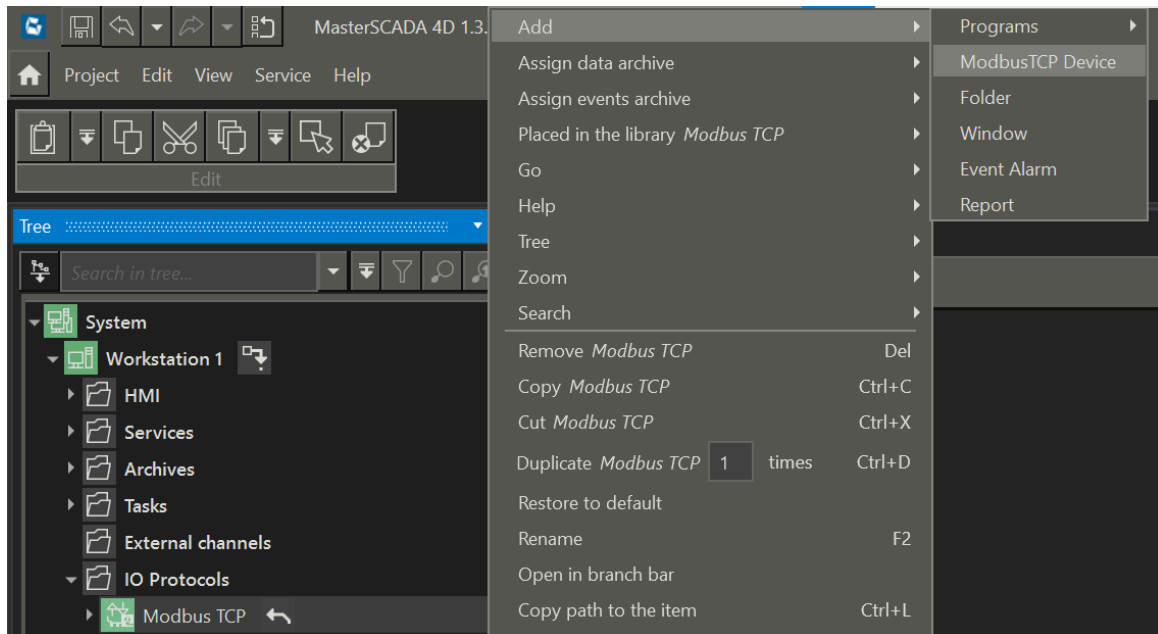


Figure 13

In our project we used **ModbusTCP Device** as an input/output module, so the required number of channels must be added manually. In the first lesson, we will add 1 channel.

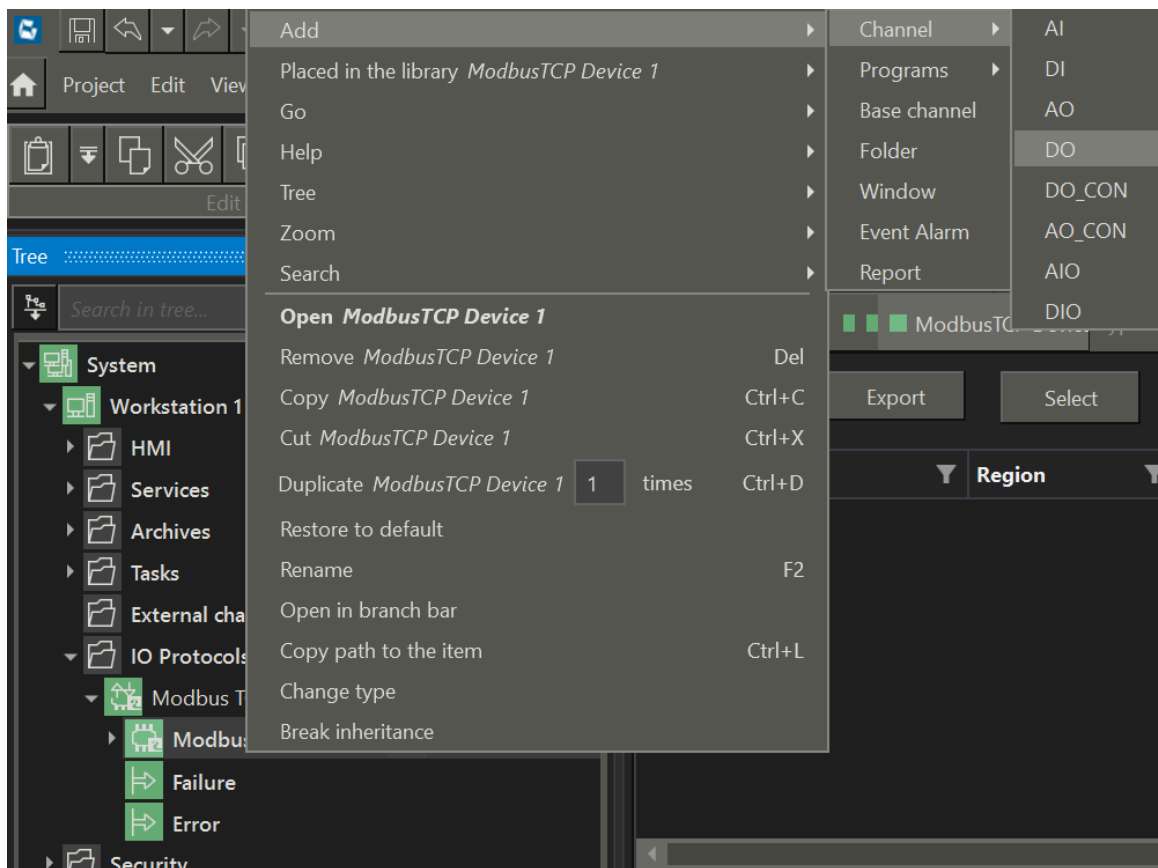


Figure 14

The corresponding item will be added to the system tree.

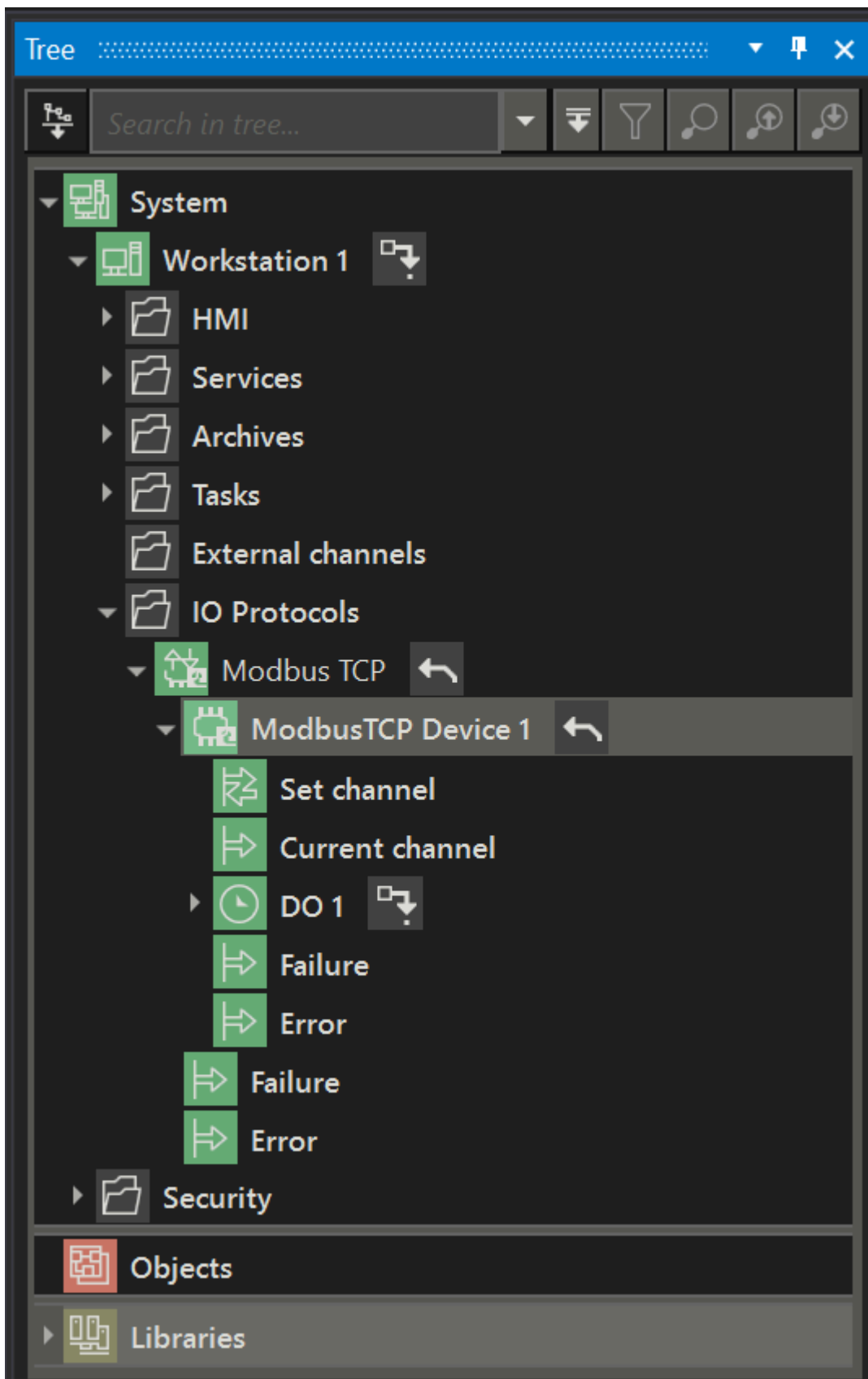


Figure 15

At this point, configuration of the physical system structure is complete. Let us proceed to creating the logical part of the project.

QS. Lesson 1. Creating the logical part of the project

First, open the objects tree context menu by right-clicking and execute the Add - Object.

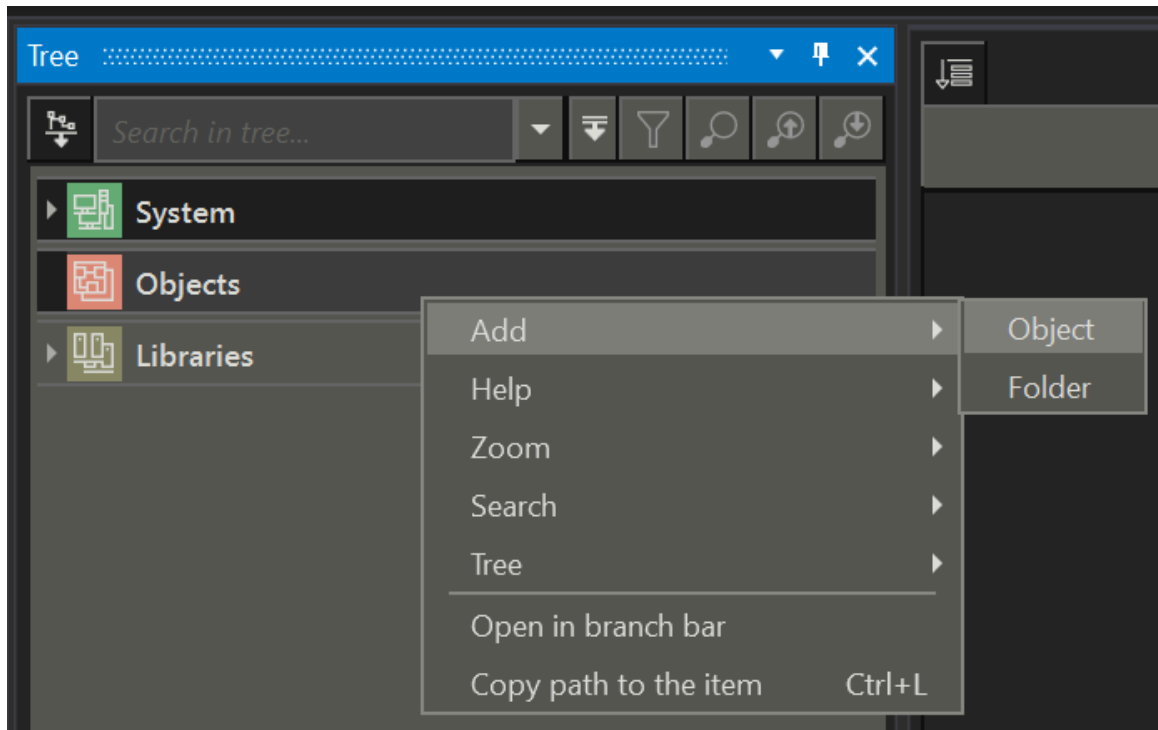


Figure 16

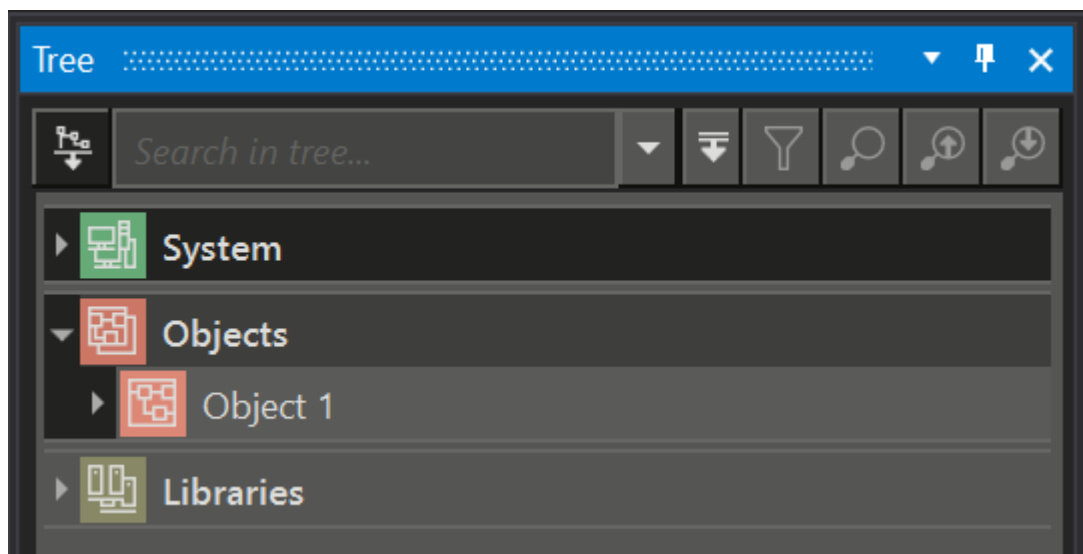


Figure 17

In addition to parameters, an object can contain programs for data processing, graphic windows, and other items.

For the task at hand, we will have one control signal. Right-click the object and from the context menu select **Add - Parameter**.

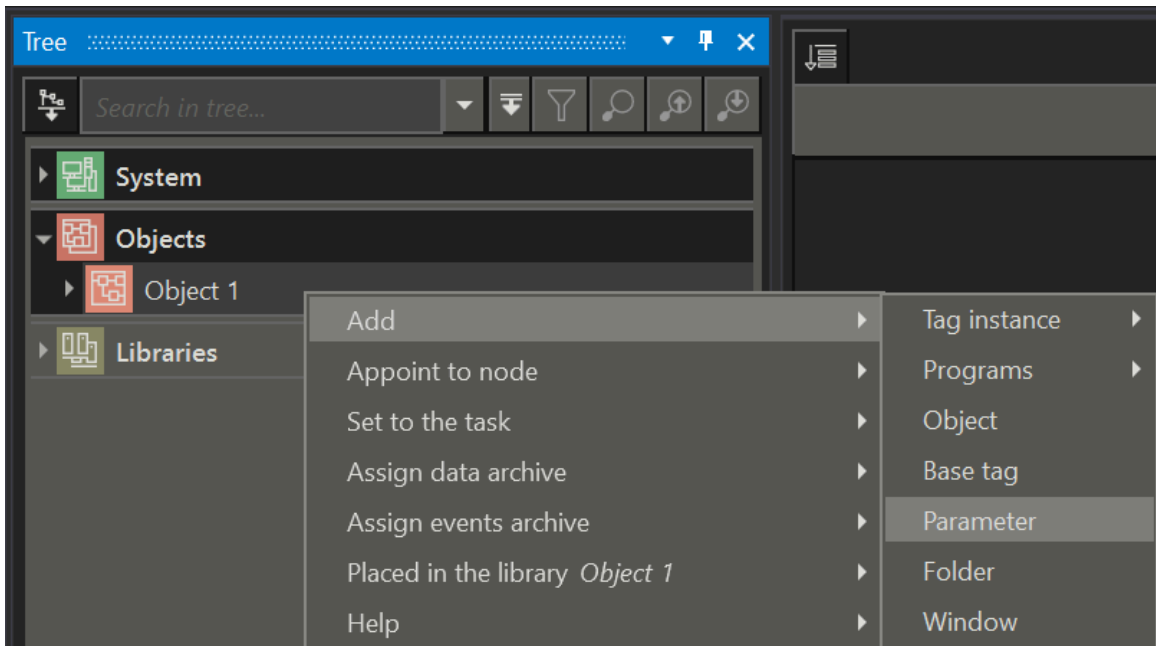


Figure 18

In the parameter properties window that opens, assign it a name, for example, **Control output** and specify the parameter type — **BOOL** (Boolean).

Important! In this example, for simplicity, we will disregard the rule that the parameter data type must match the channel parameter data type.

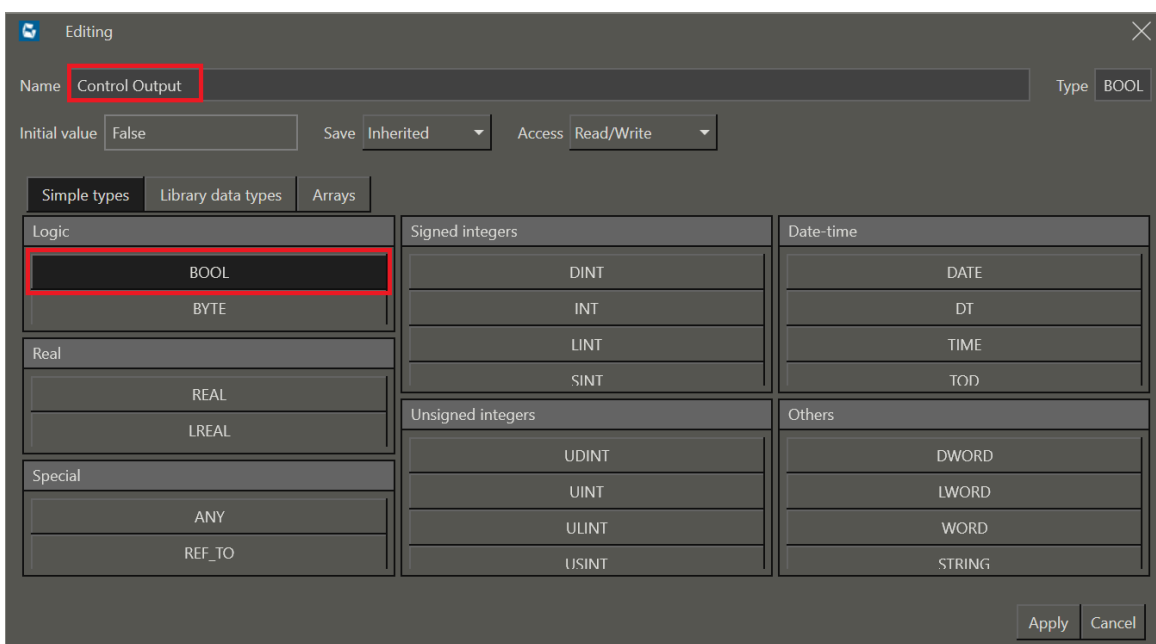


Figure 19

As a result, we get:

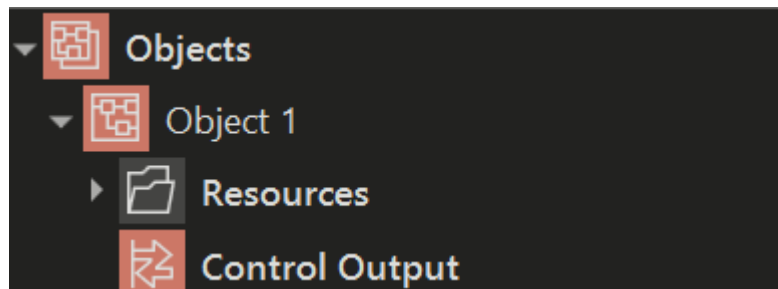


Figure 20 – As a result, we get

Now let us create the control program. To do this, use the **context panel** of the tree item **Object 1**. The figure below shows that this panel allows you to add a program in one of the four languages of the IEC 61131-3 standard. Let us select a program in the function block diagram language — **FBD**.

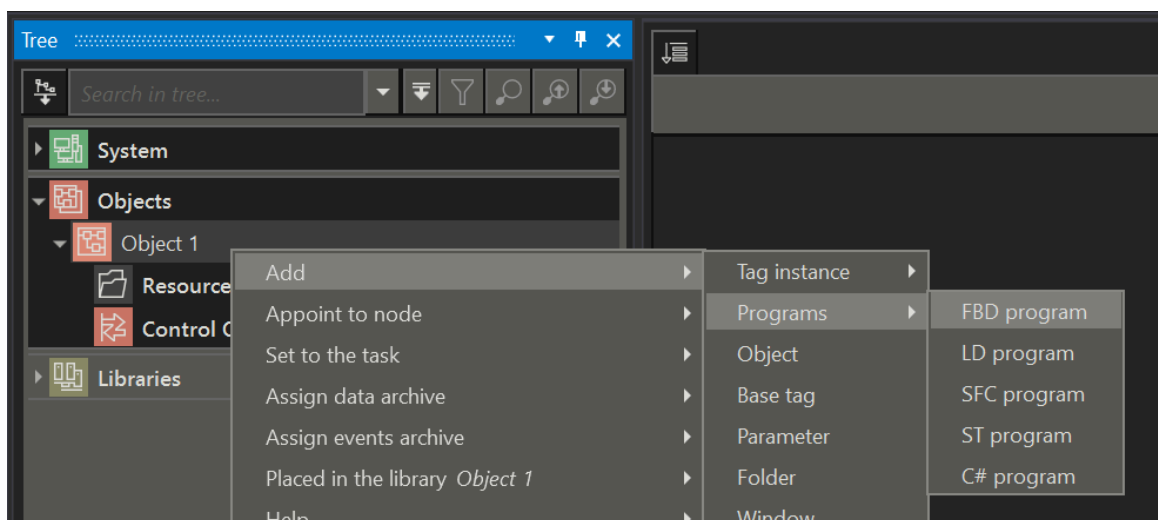


Figure 21

The editor window for this language will open.

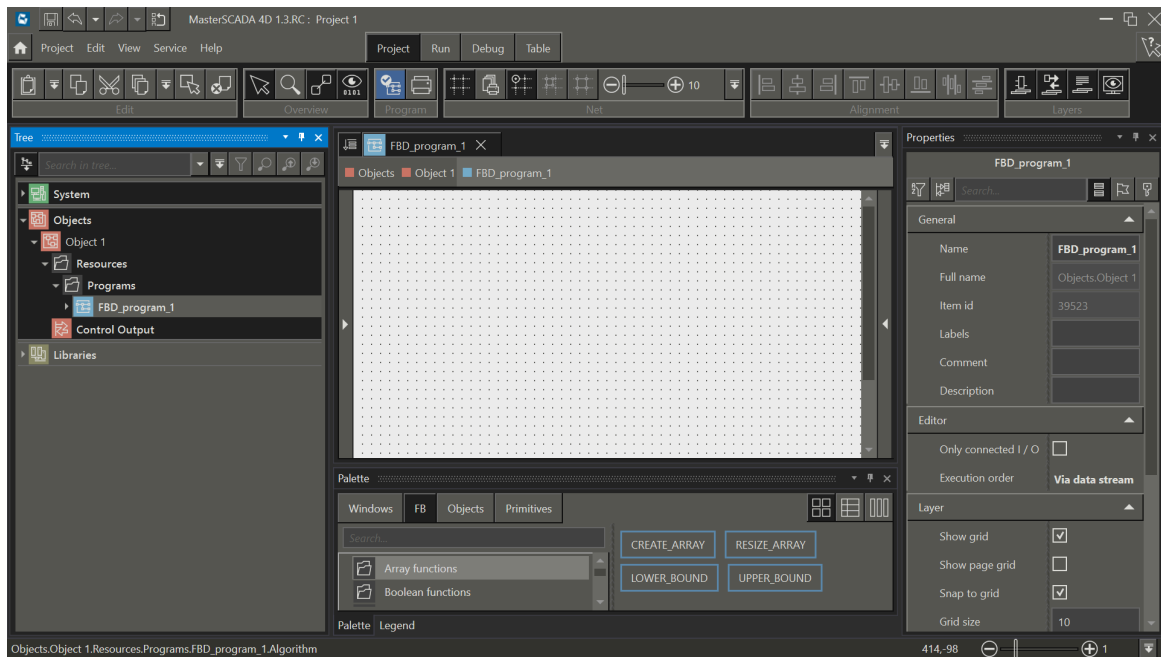


Figure 22

On the left and right of the editing area, similar to electrical diagrams, are **terminal blocks**. The left terminal block contains input signals, and the right one contains outputs. To expand or collapse the terminal blocks, click the arrows at the edges of the editing area.

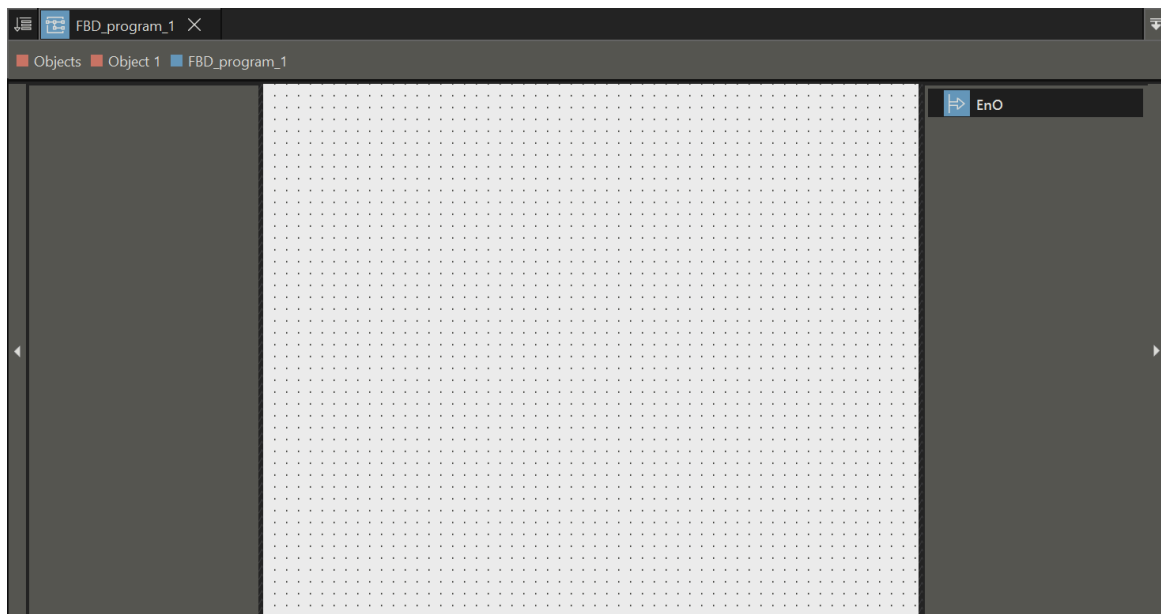


Figure 23

Let us create two input parameters in the program: **Start** and **Stop**. To do this, right-click anywhere on the input terminal block and from the context menu select **Add - Parameter**.

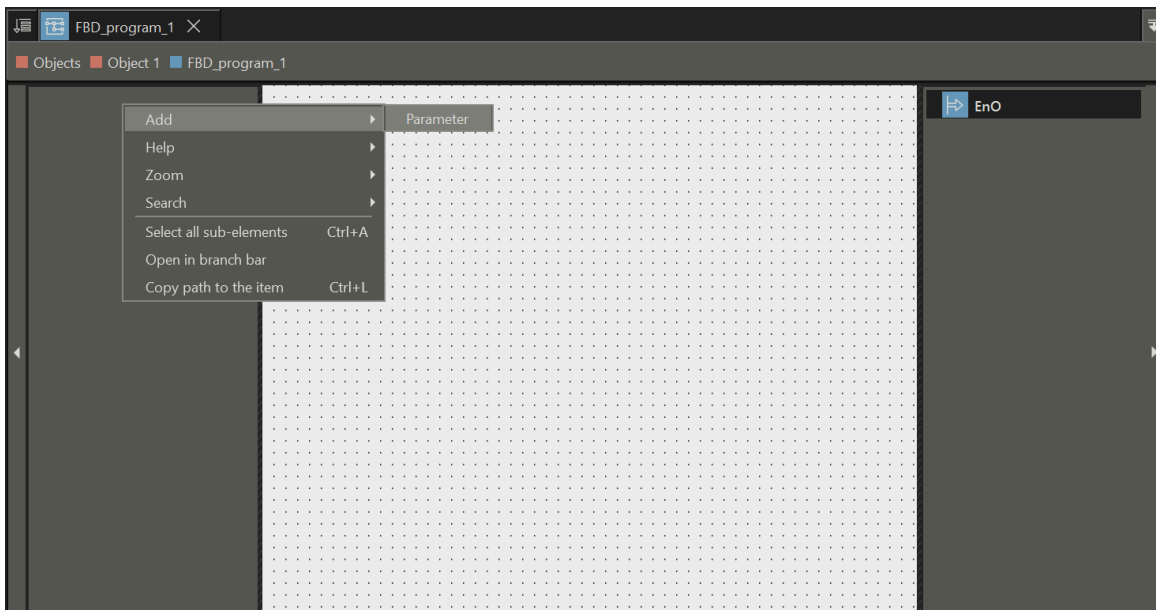


Figure 24

In the parameter properties window that opens, change its name to **Start** and set the type to **BOOL**.

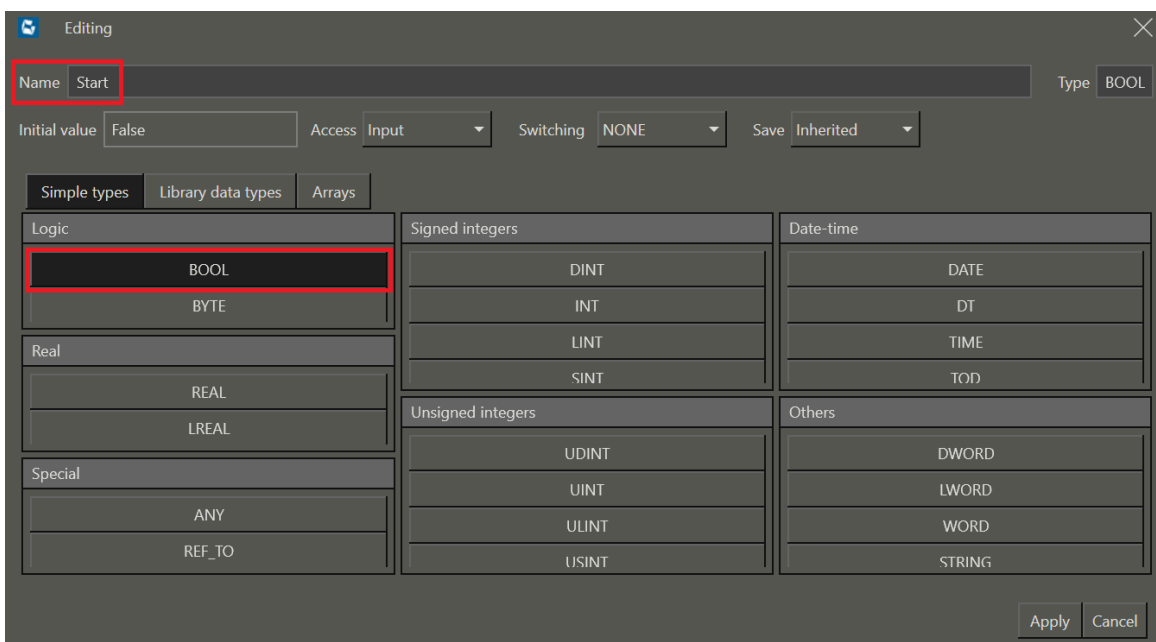


Figure 25

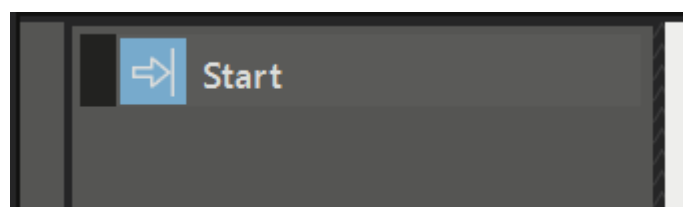


Figure 26

Similarly, create the **Stop** parameter.

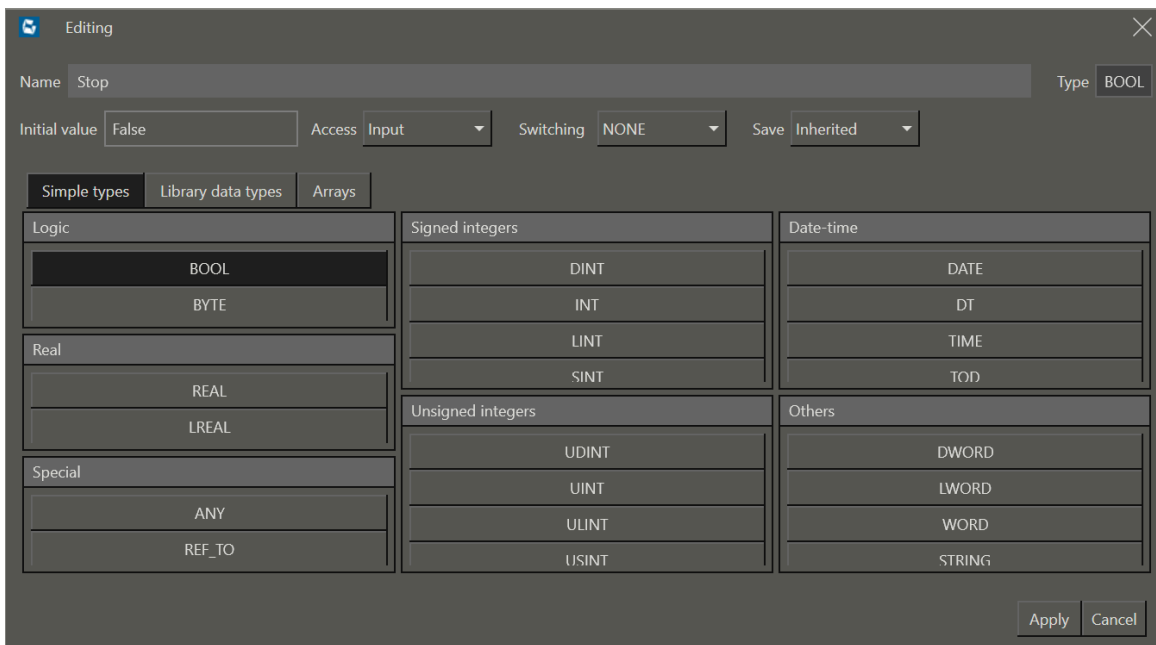


Figure 27

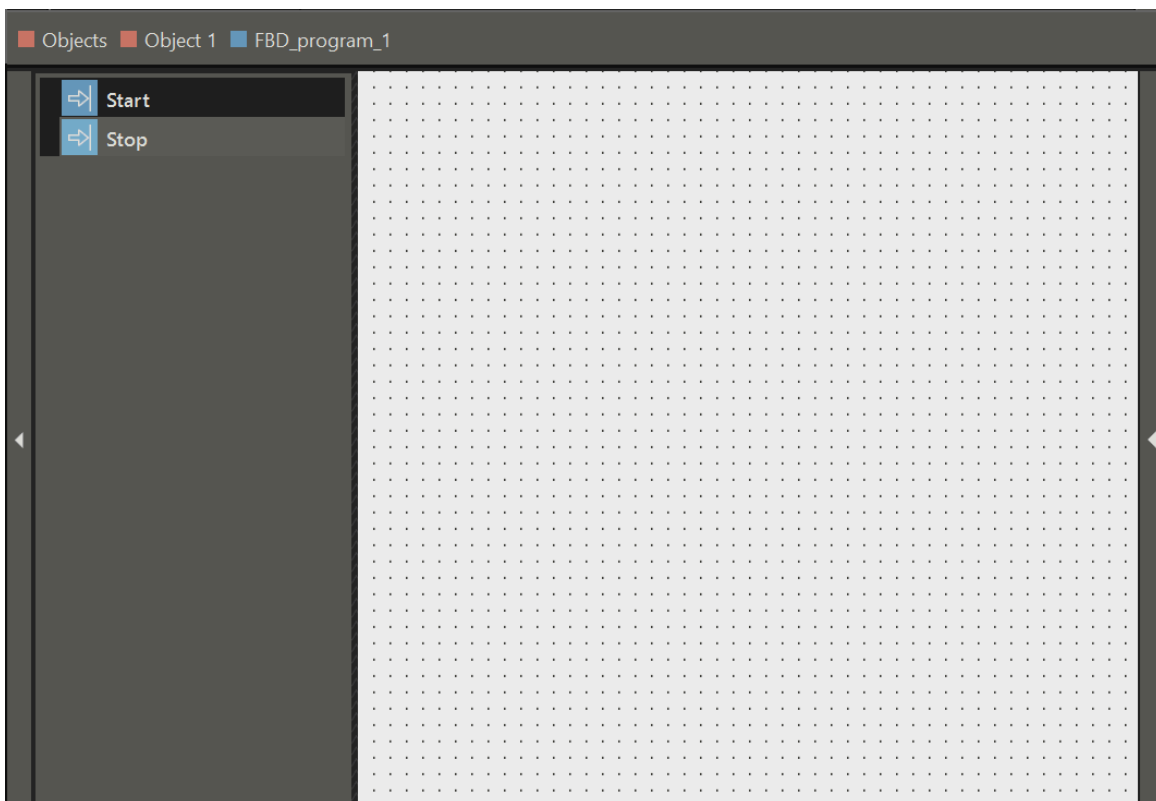


Figure 28

To form the on/off logic, we will use a standard RS trigger. To insert a function block onto the diagram, simply drag it from **FB** into the editor area. In the **Palette** find the section **Triggers** and drag the **RS** item onto the diagram.

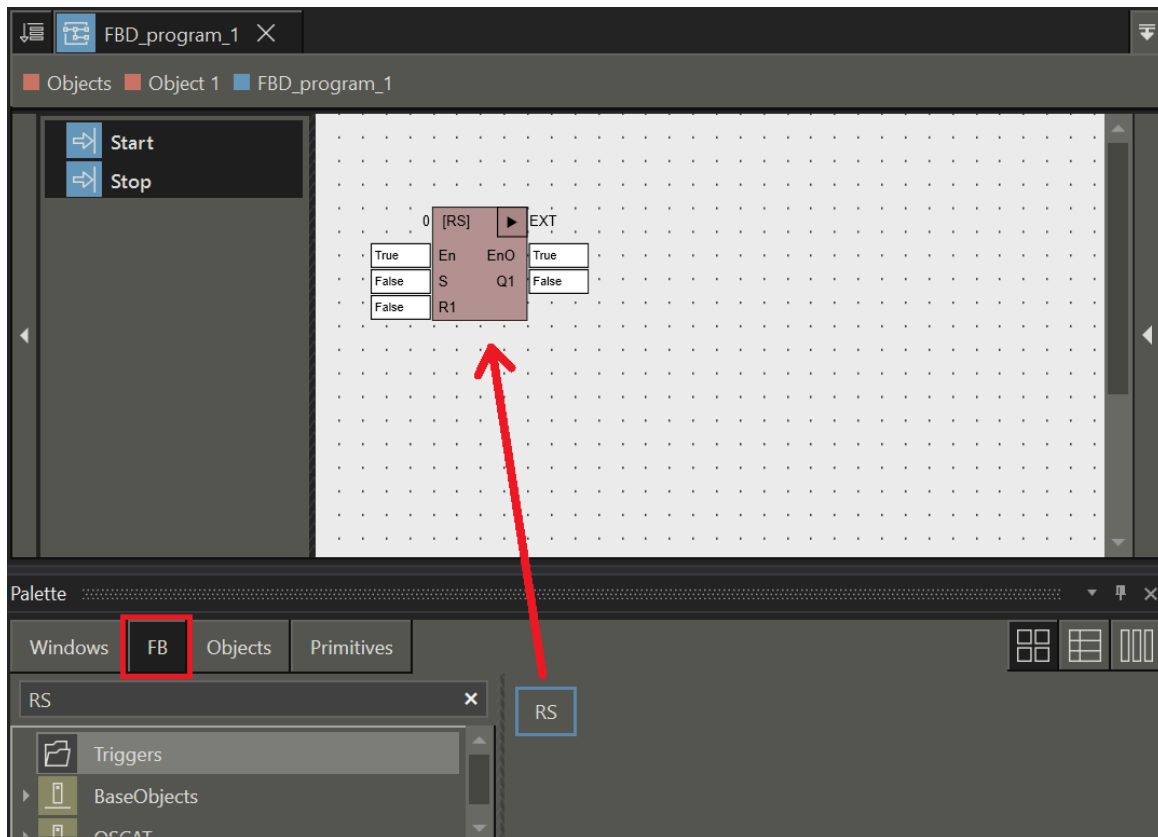


Figure 29

Now we need to connect the program inputs to the RS trigger inputs (or, as it is also said, "link" or "establish a connection"). To do this, left-click above the parameter name (for example, **Start**), and, holding the mouse button down, move the pointer to the parameter you want to link it to (**S**), and release the mouse button over it.

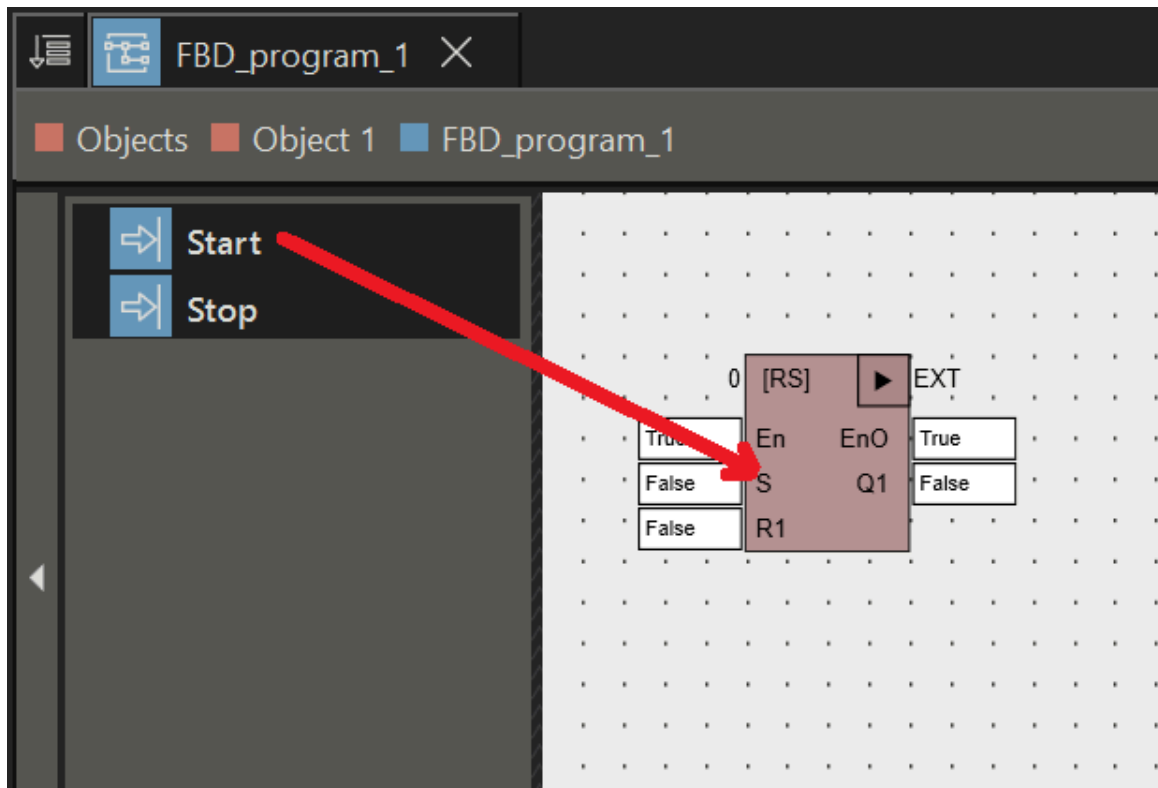


Figure 30

In the same way, link the program input **Stop** to the trigger input **R1**. The result:

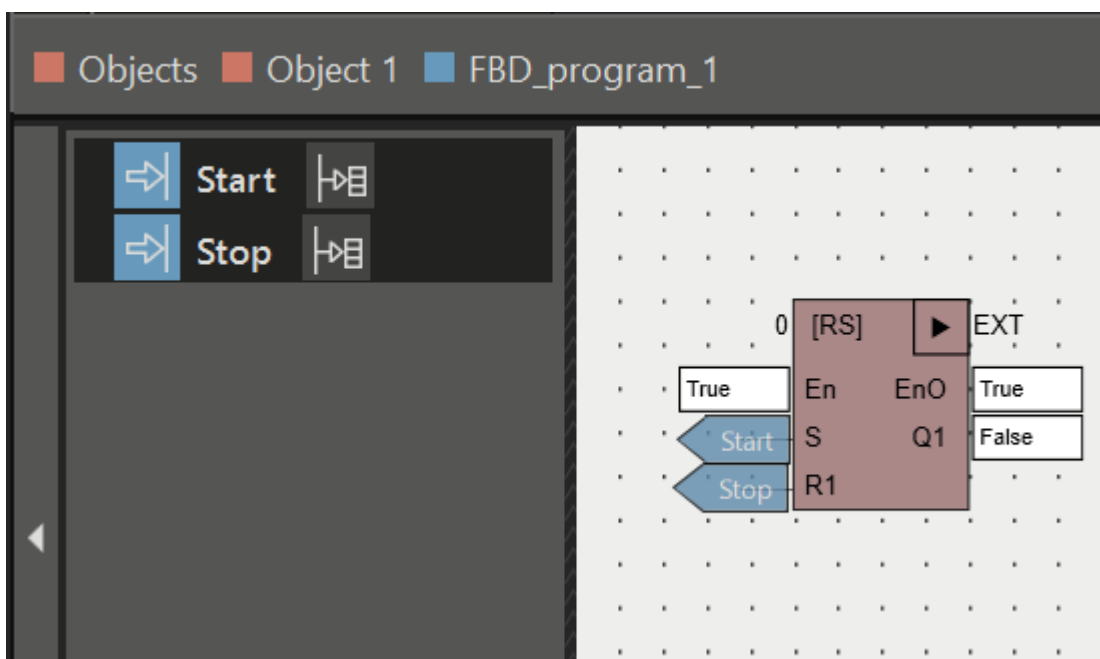


Figure 31– In the same way, link the program input **Stop** to the trigger input **R1**. The result

We will return to the program input parameters later; for now, let us create an output parameter.

To do this, drag the object parameter **Control output** to the program output terminal block — an output program parameter will be created (by default with the same name), along with a link to it.

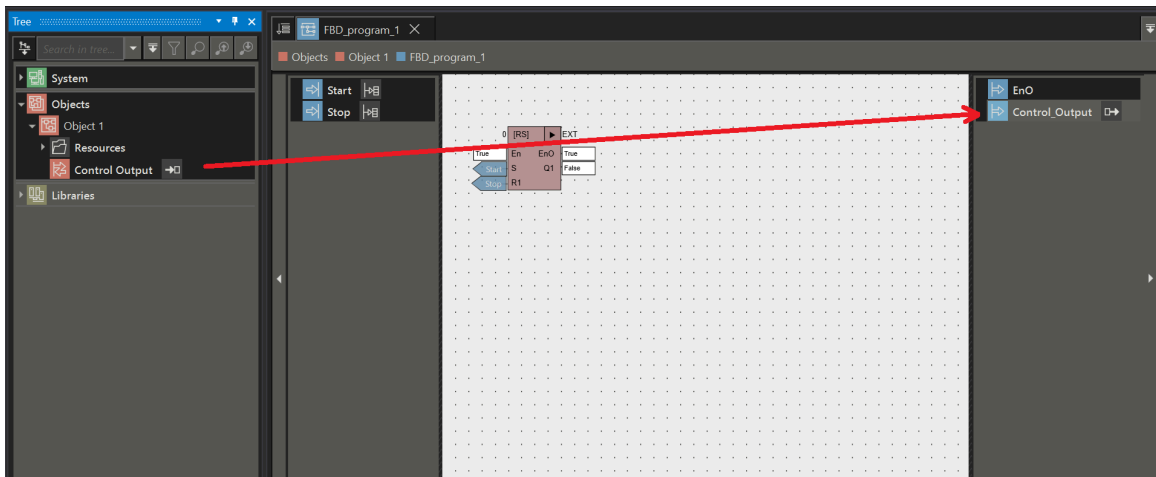


Figure 32

Finally, link the trigger output **Q1** to the output parameter **Control output**.

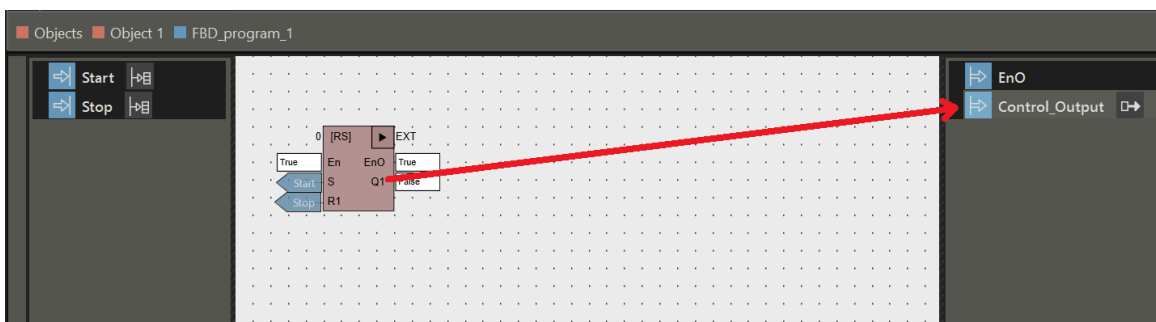


Figure 33

Thus, we have formed the output path from the trigger output **Q1** to the **Control output** object parameter.

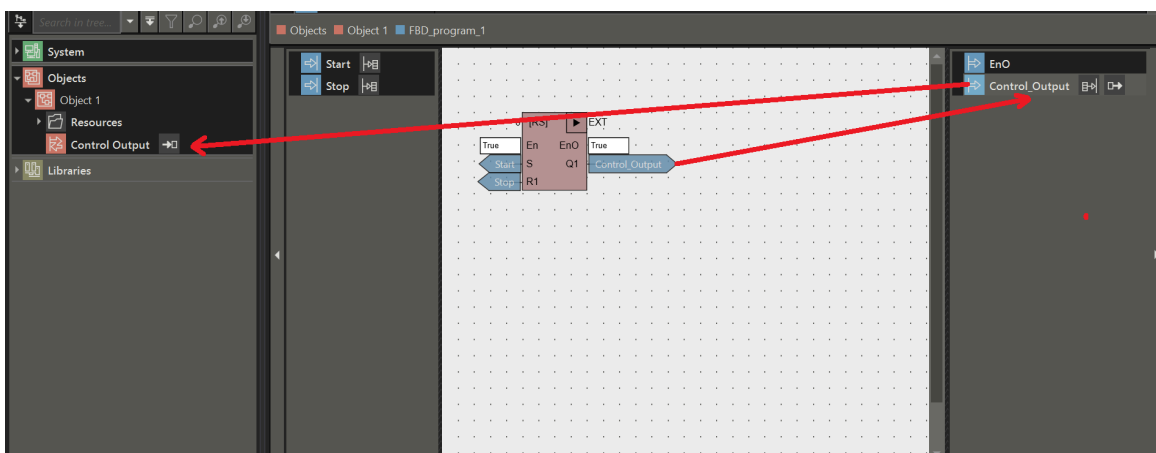


Figure 34

At this point, control logic formation is complete. Just in case, click the **Compile** button and make sure compilation completes successfully.

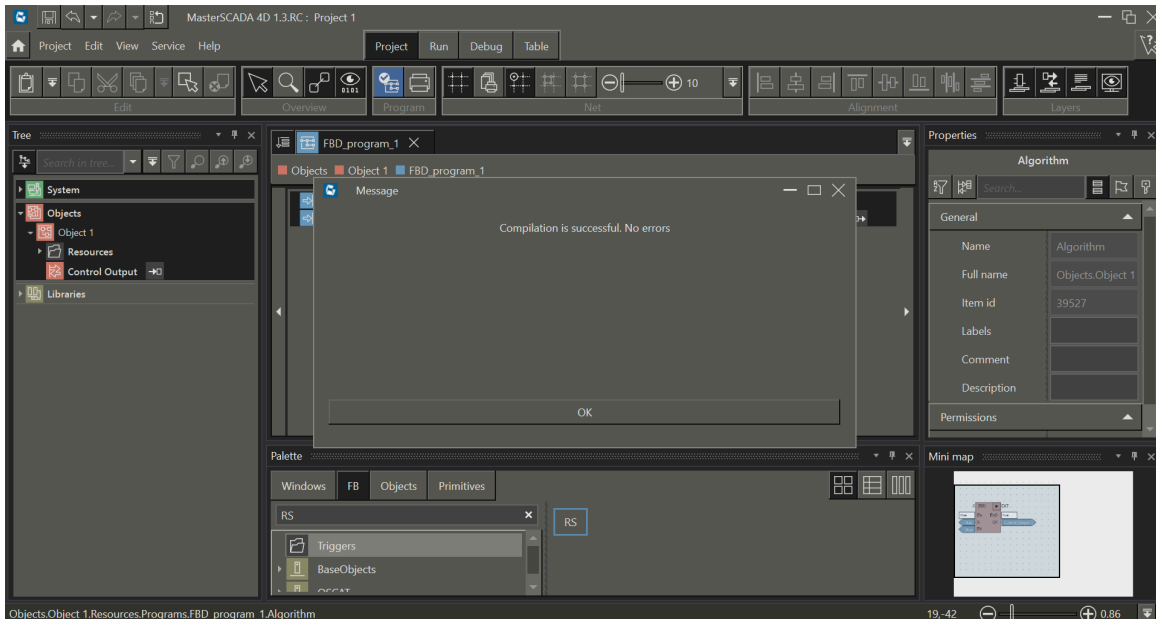


Figure 35

The next step is creating visualization.

QS. Lesson 1. Creating a control window

Let us start by selecting the **Add - Window** item in the object context menu.

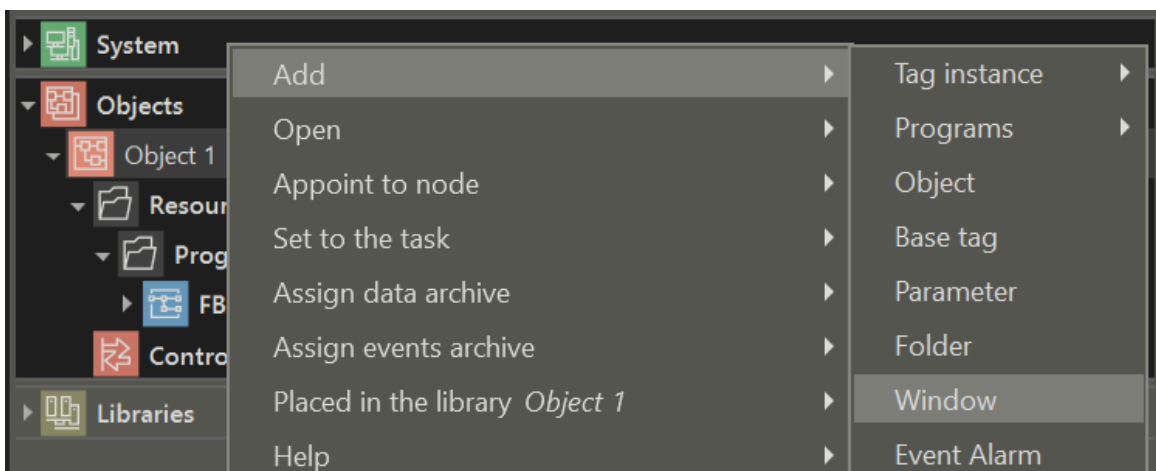


Figure 36

A window will appear for the object and will immediately open in the graphic interface editor area.

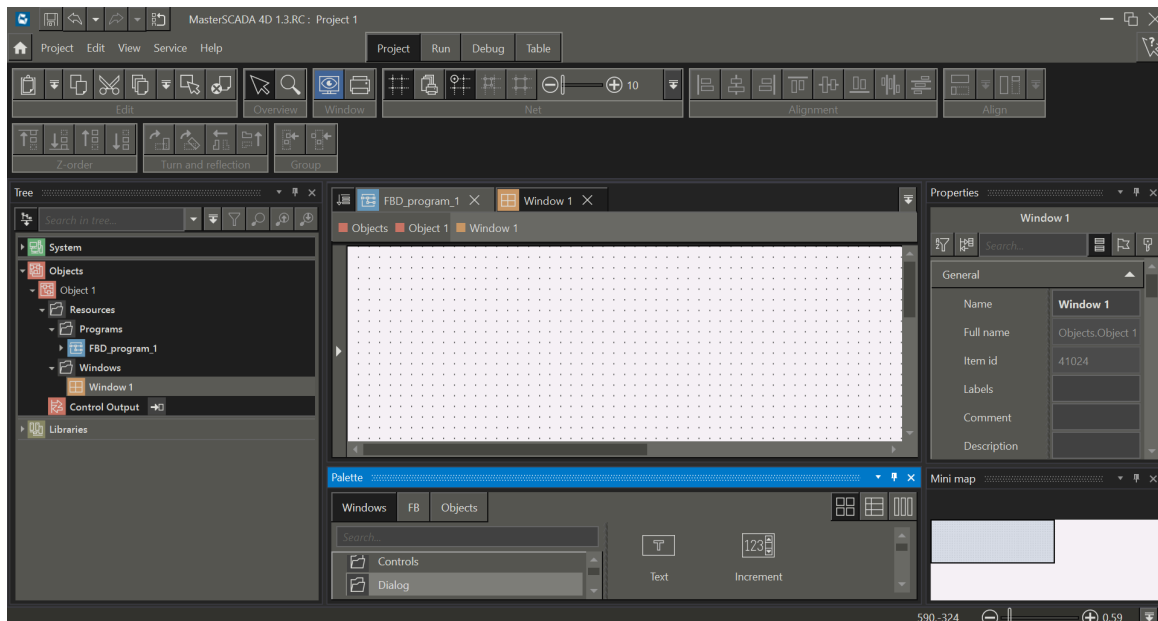


Figure 37

Our task is to place two buttons in this window — **Start** and **Stop** — whose presses will be passed to the corresponding program input parameters, and also create an indication of the **Control output** state.

From the **Parameters** group of the program we created, drag the **Start** parameter into the window, holding the right mouse button. From the context menu that opens, select **Create linked control.Button.Bidirectional link**.

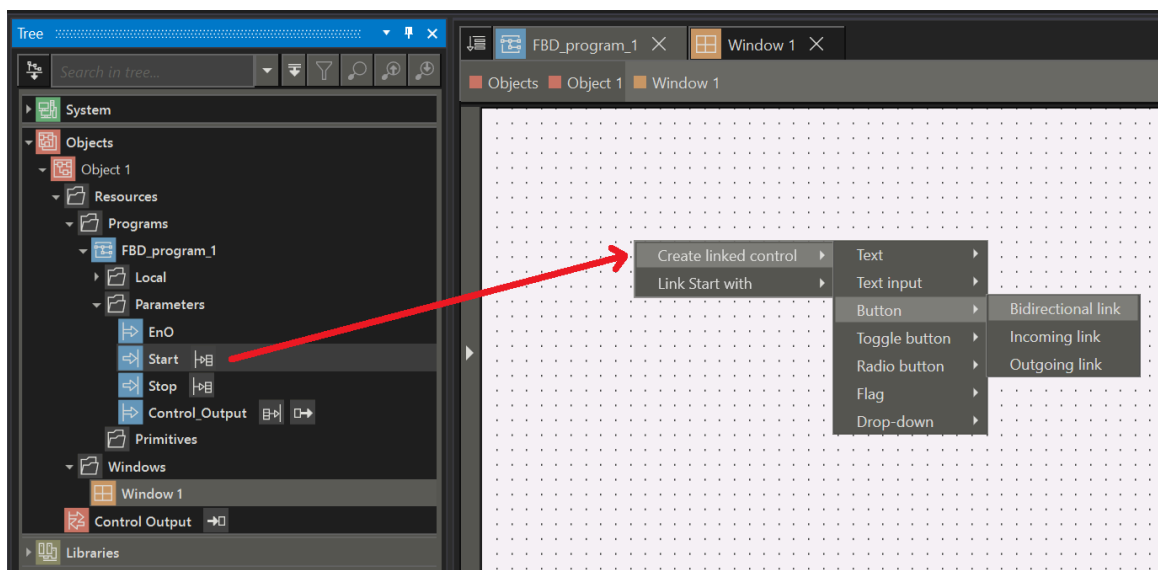


Figure 38

A button with links already established between the **Start** parameter and the button **Pressed** property will be inserted into the window. When the button is pressed, the **Start** parameter will be set to **TRUE**, and when the button is

released, this parameter will be set to **False**. Thus, the **Start** parameter will reflect the button state.

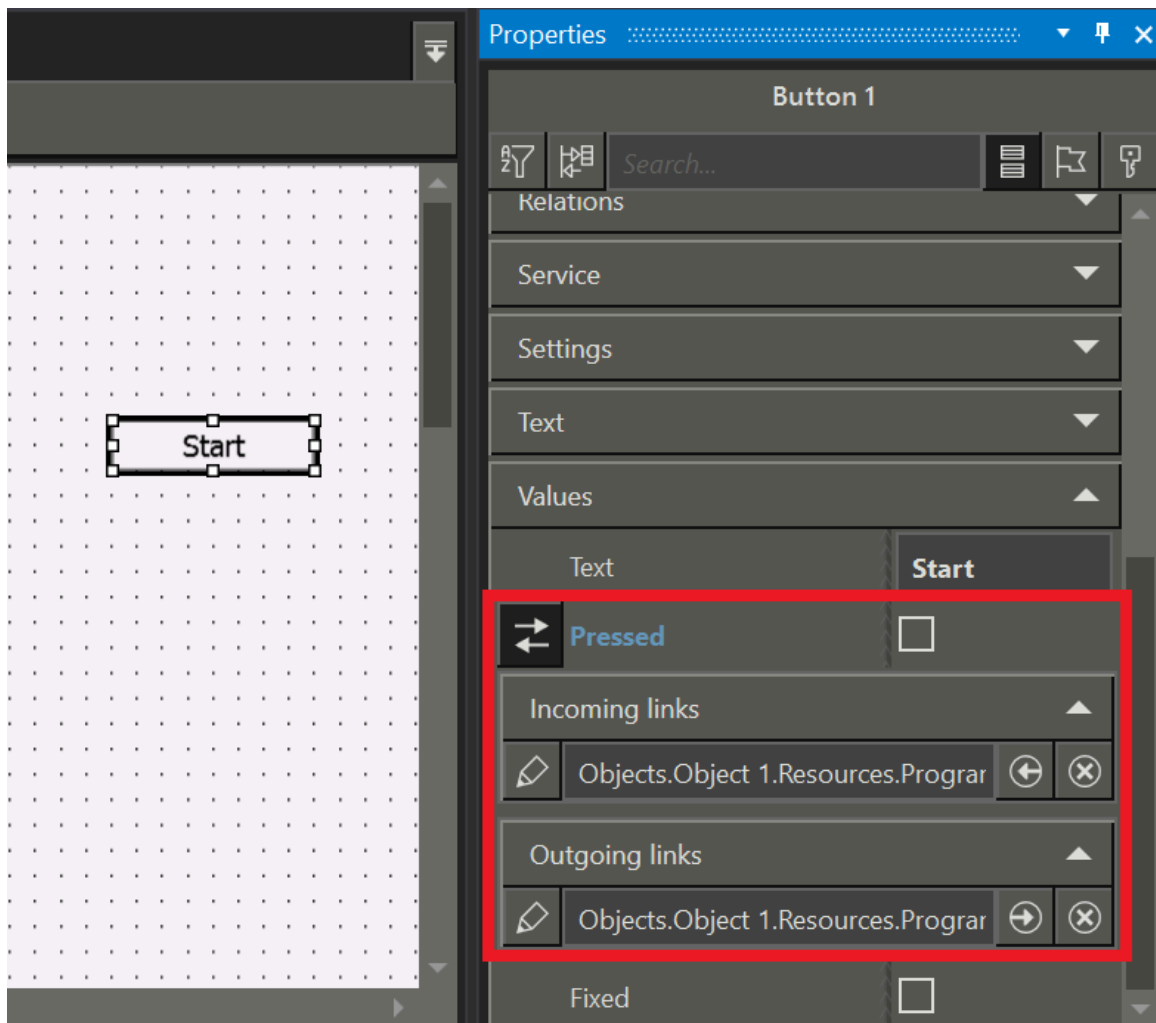


Figure 39

Perform the same actions for the **Stop** parameter.

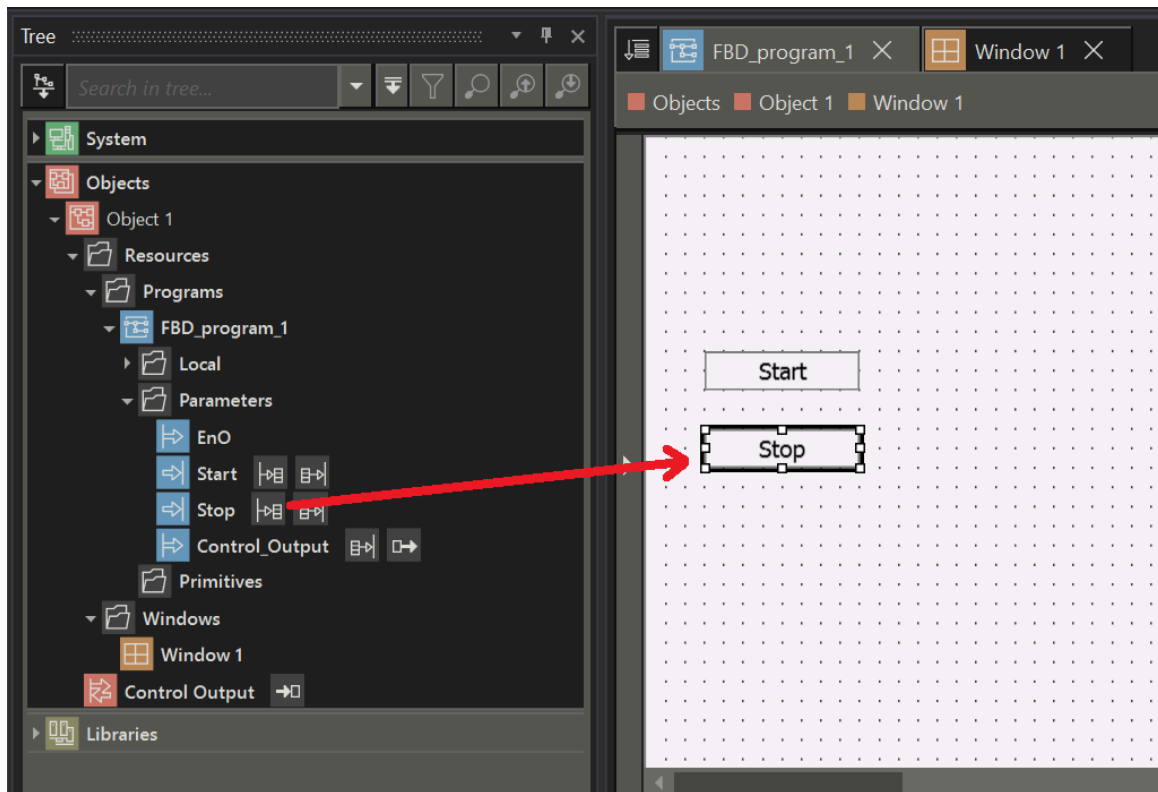


Figure 40

Now let us implement display of the **Control output** state. To do this, go to the **Legend** of the graphic editor, open the **Palette** tab, select the **Primitives**, category, and drag the **Oval** element into the window holding the left mouse button.

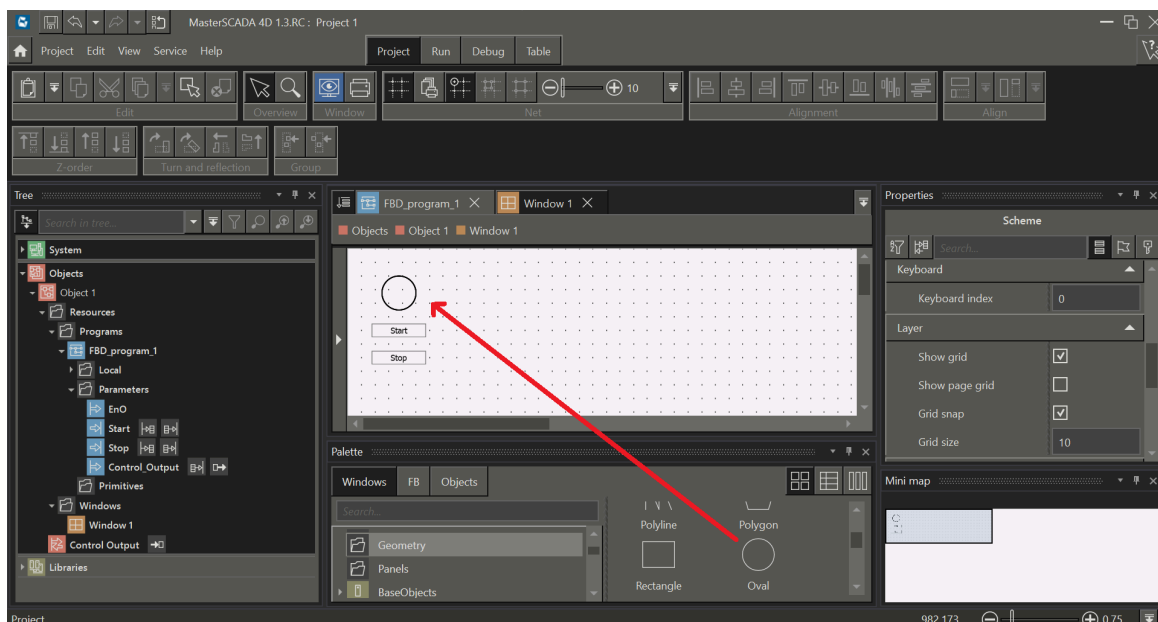


Figure 41

Let us make the oval color reflect the current output signal state: on or off.

To do this, drag the object parameter **Control output** onto the oval **Background fill** property.

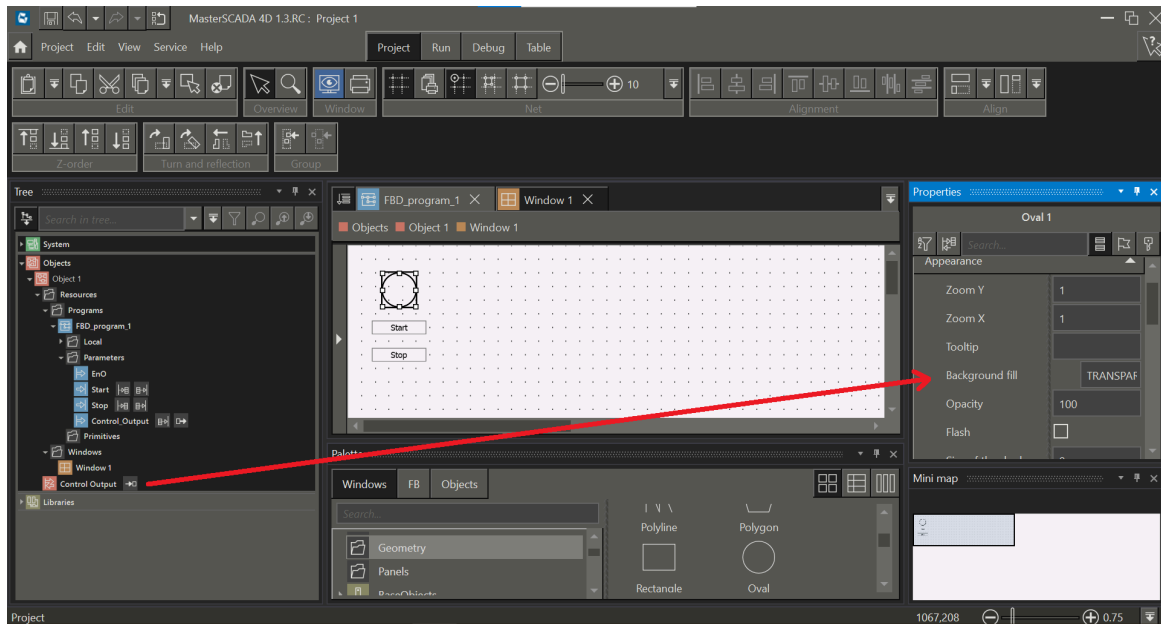


Figure 42

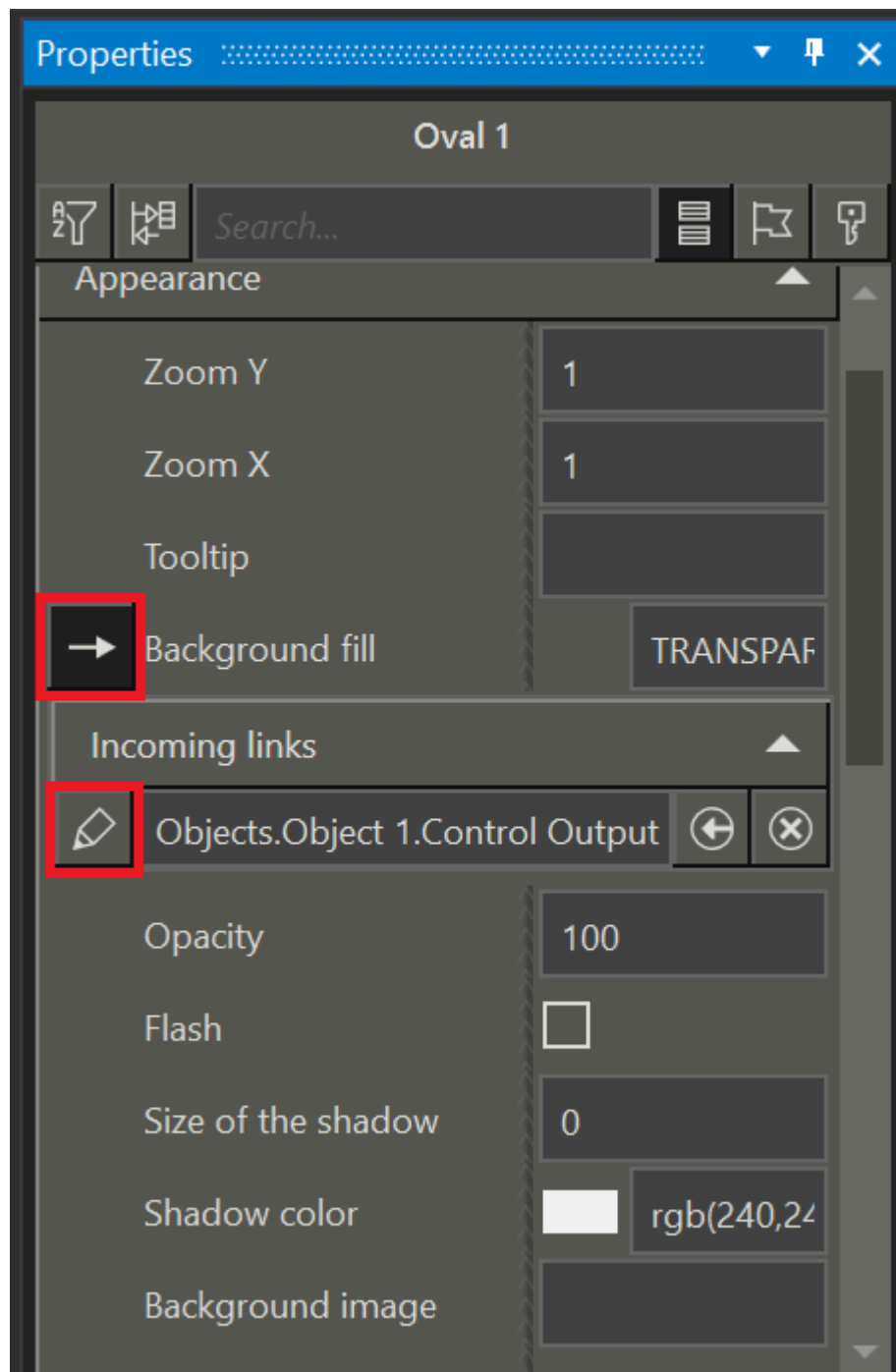


Figure 43

A dialog box will open:

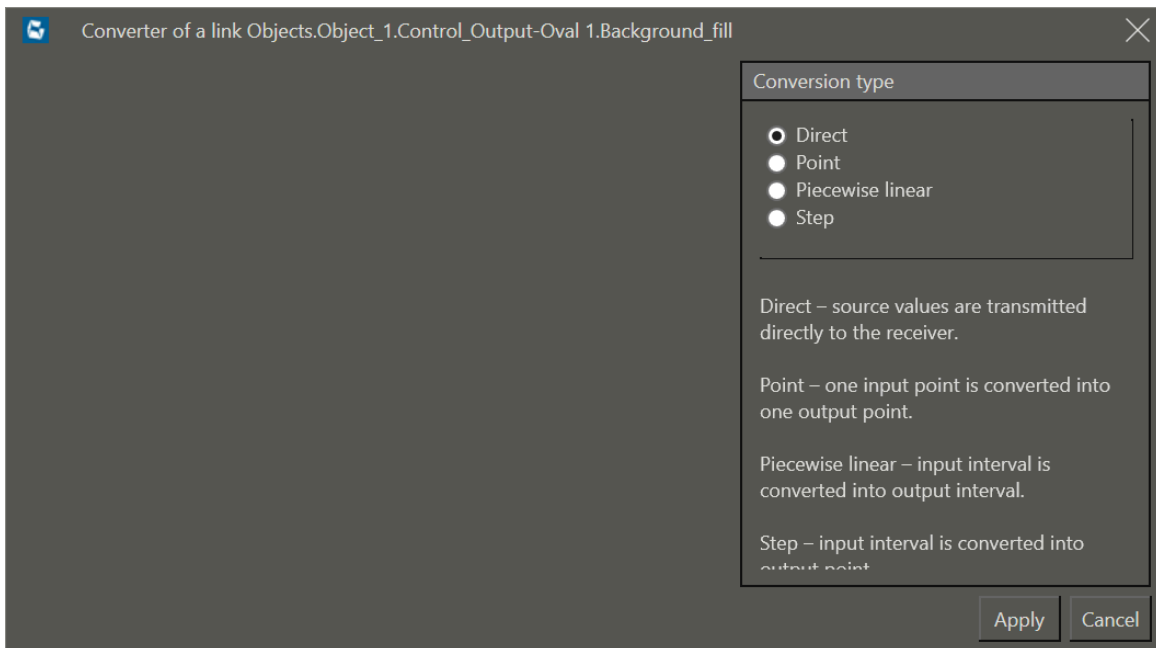


Figure 44 – A dialog box will open

A conversion can be assigned to any link in **MasterSCADA 4D**. Select **point conversion type**. Configure it so that the **False** value corresponds to white (select WHITE from the standard colors). To enable the selection buttons, right-click next to the required **Source** value in the **Target** field.

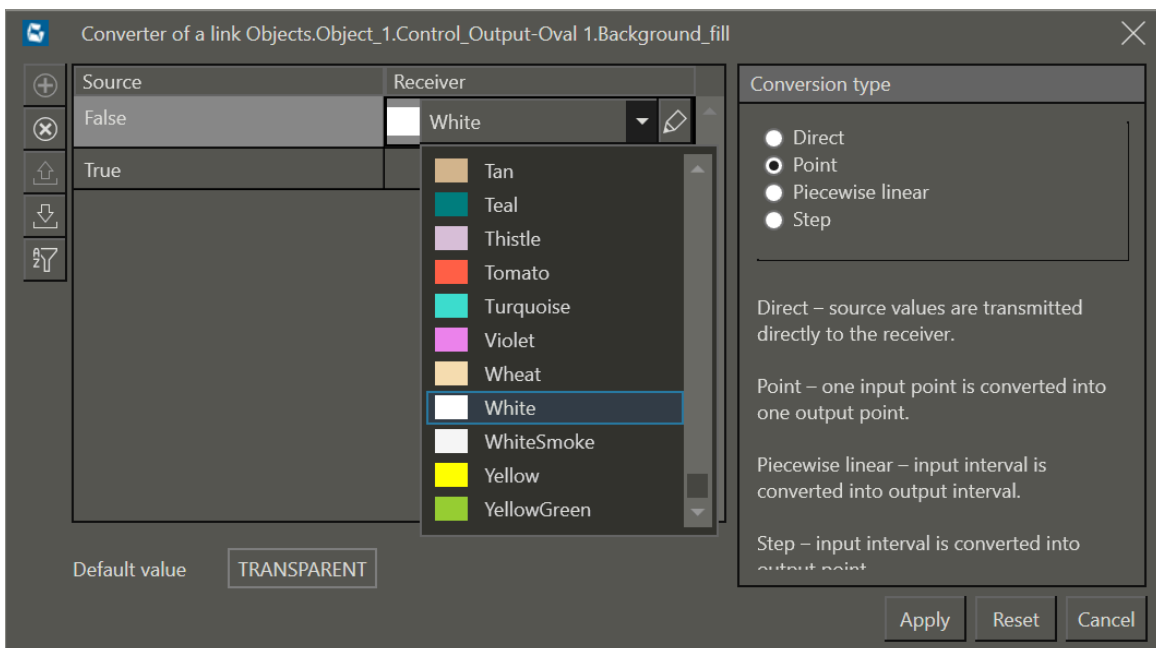


Figure 45

For the **TRUE** value, select green from the palette.

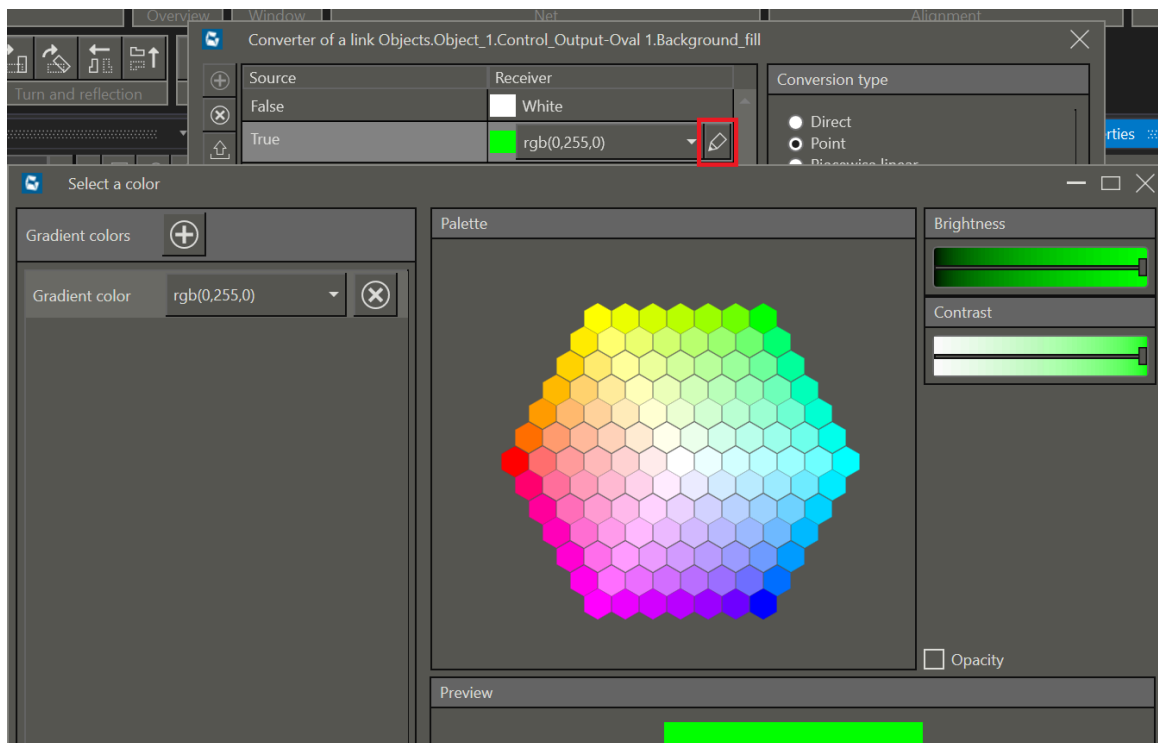


Figure 46

Then close the conversion window.

Now let us add a pump image. To do this, select any pump from the pump library included in the **Common**, library and drag it into the window.

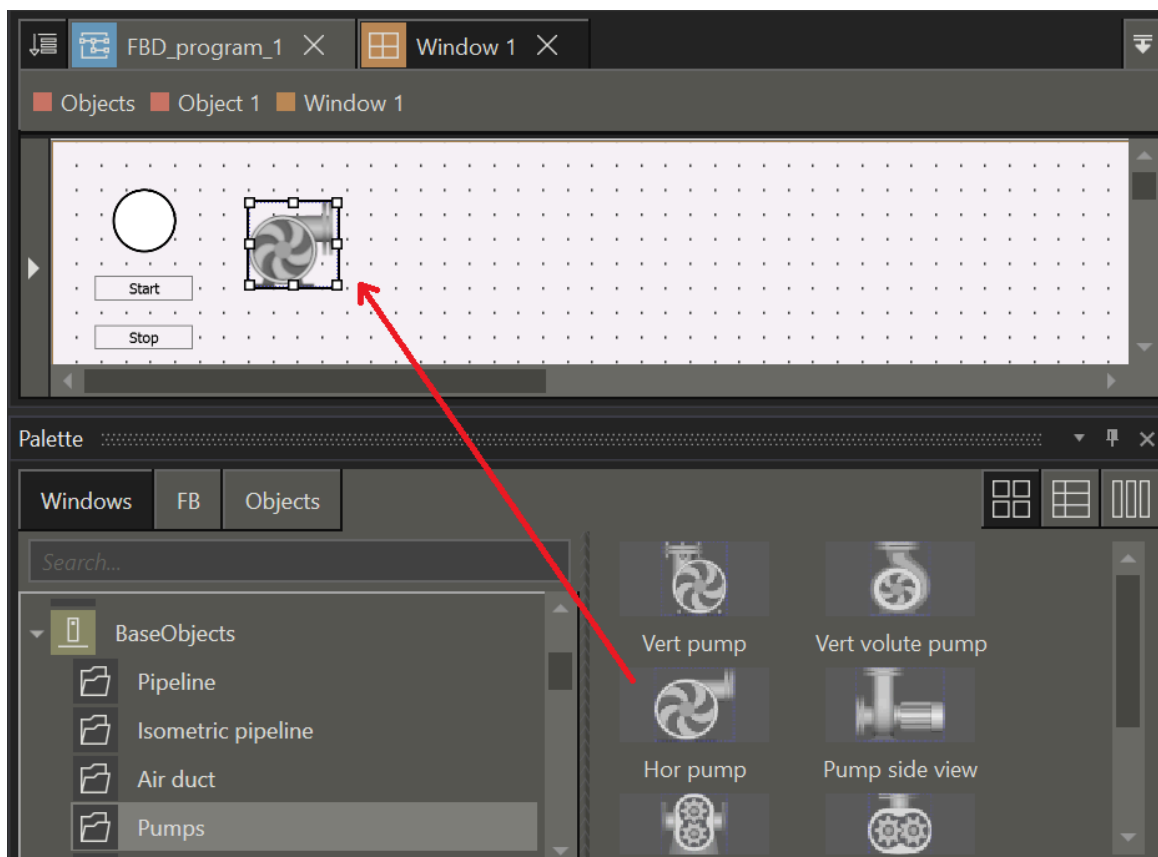


Figure 47

Finally, link the object parameter **Control output** to the pump **Rotation** and **Running** properties.

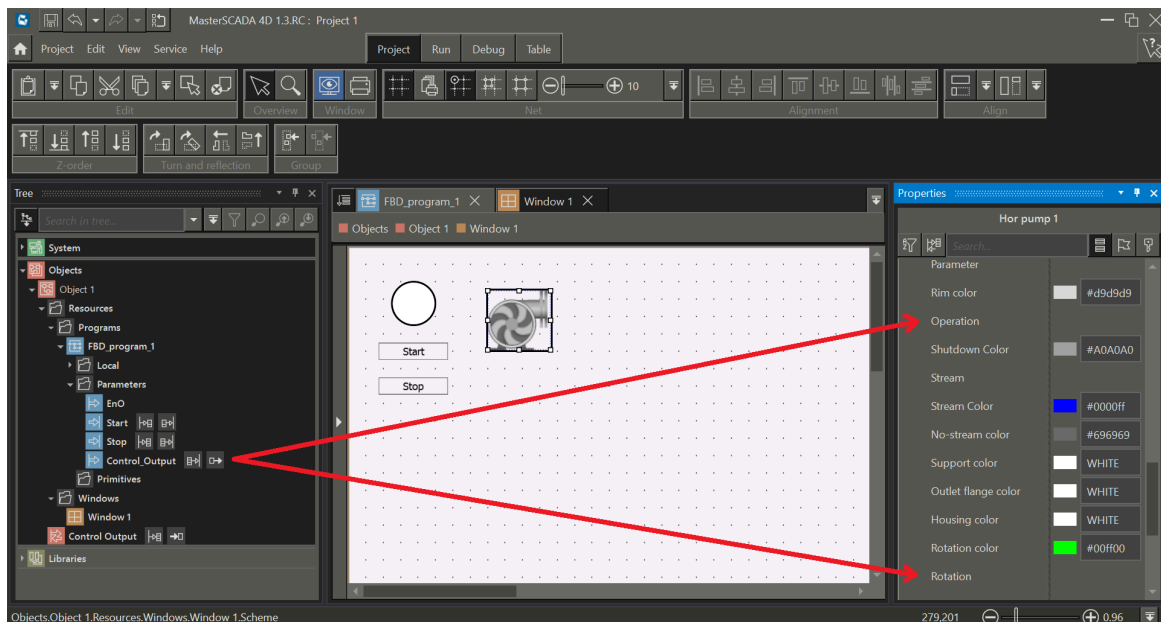


Figure 48

At this point, visualization formation is complete.

The next step is to assign the **startup window**. Our project currently has only one window. However, there can be many such windows, and it is necessary to specify which one will be the startup window. To do this, right-click this window, from the context menu select **Set as startup window**, and specify where the startup window will run. In our case, there is only one possible option.

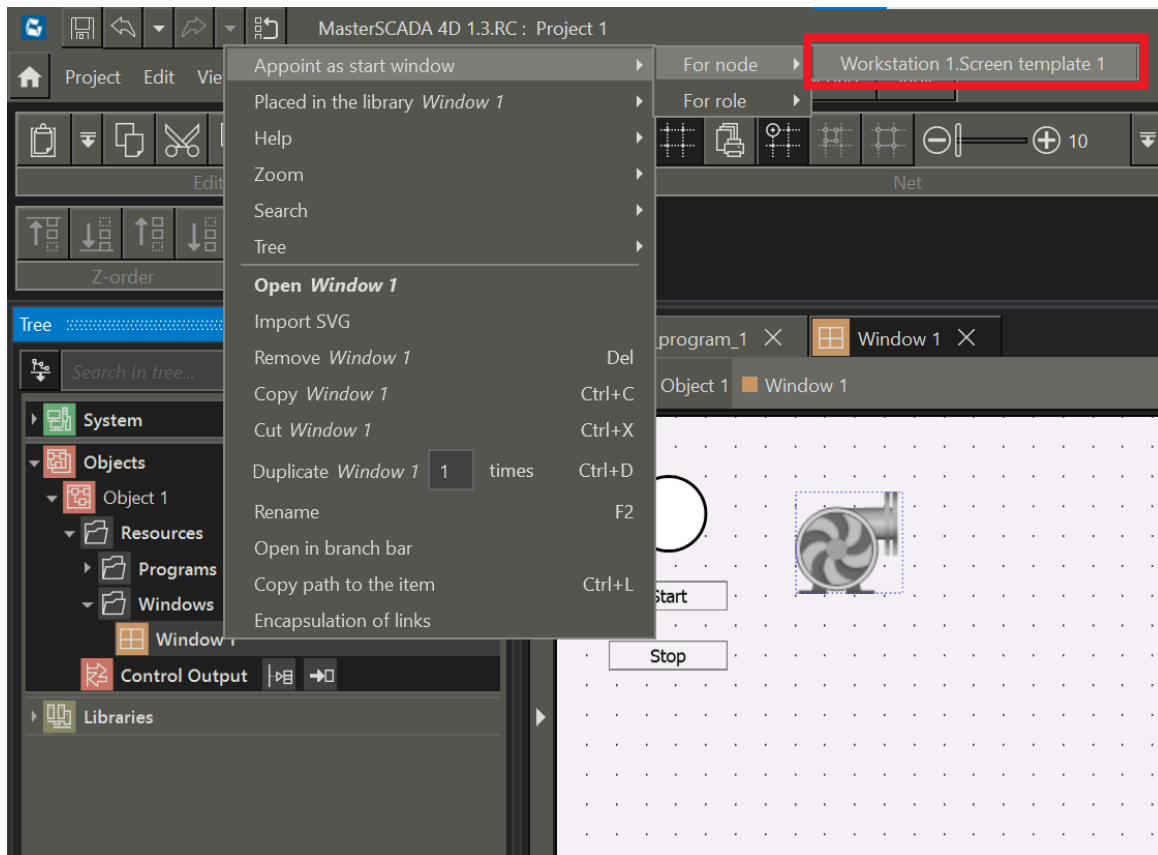


Figure 49

Thus, we have a functionally complete object containing a control program, visualization means, and one parameter for linking this object to the external world. In our case, we need to use this parameter to control the module discrete output. To do this, left-drag the object parameter **Control output** to the discrete output **DO 1**. A link will be created that passes the value from the **Control output** parameter to the discrete output channel.

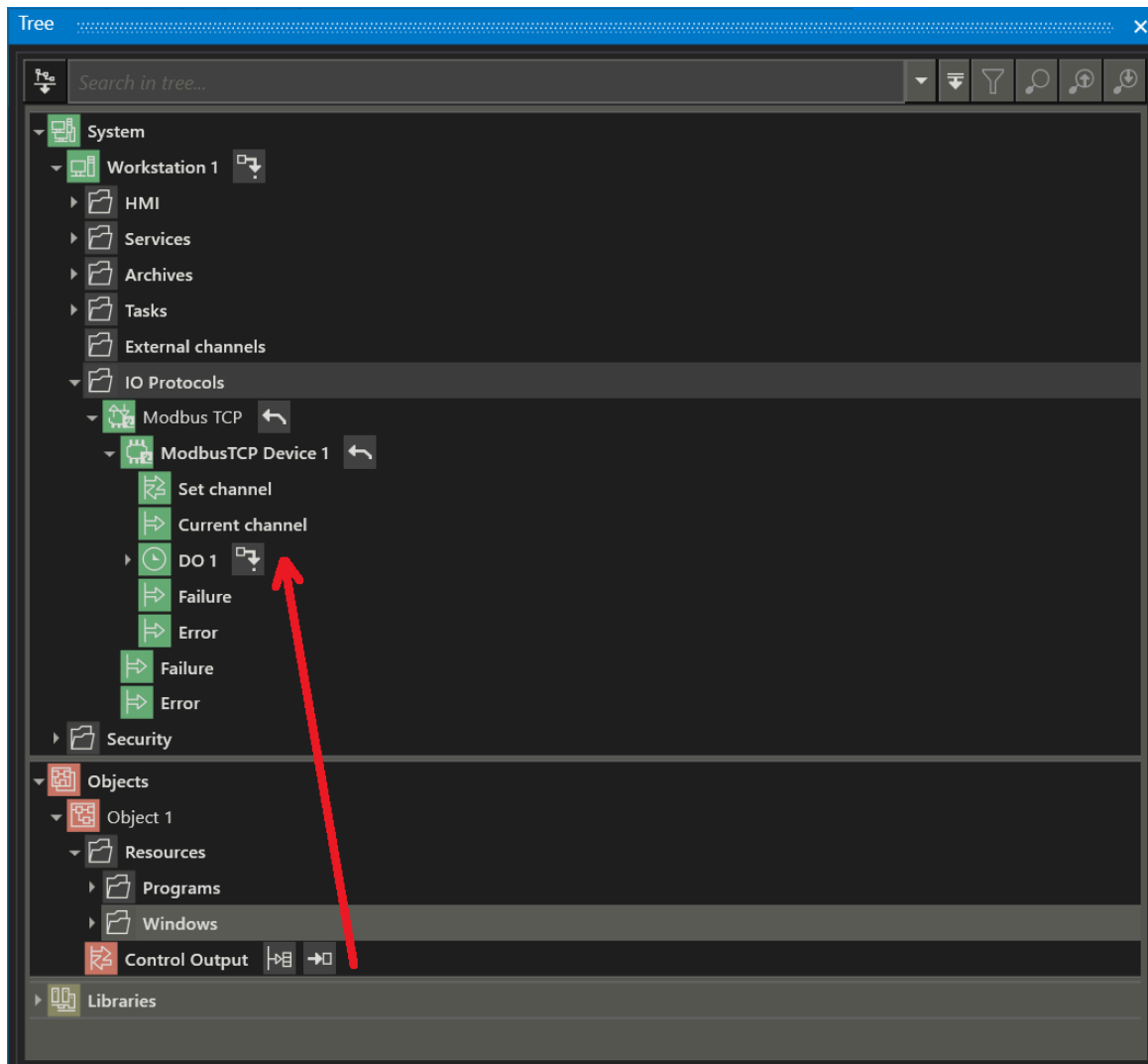


Figure 50

At this point, project creation is complete.

QS. Lesson 1. Launch

First, switch to the **Runtime** tab on the toolbar and click the **Compile**:

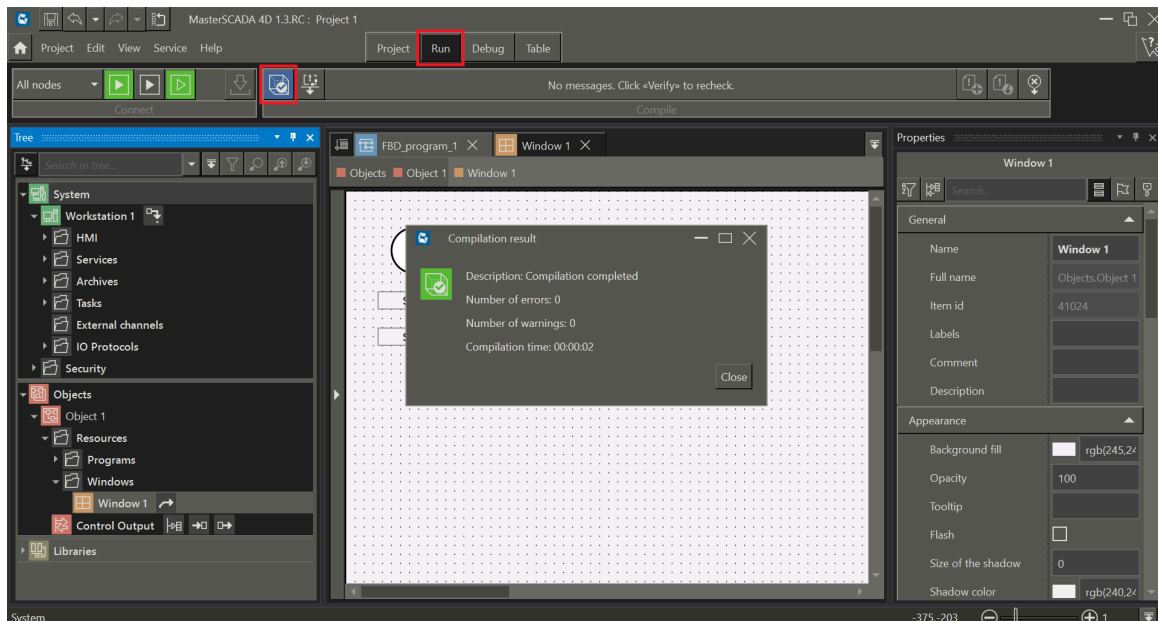


Figure 51 – First, switch to the Runtime tab on the toolbar and click the Compile

Then, after making sure there are no project errors, start the project in emulation mode. In this case, no download to the controller will occur; the runtime will be emulated on the same computer where the project was developed.

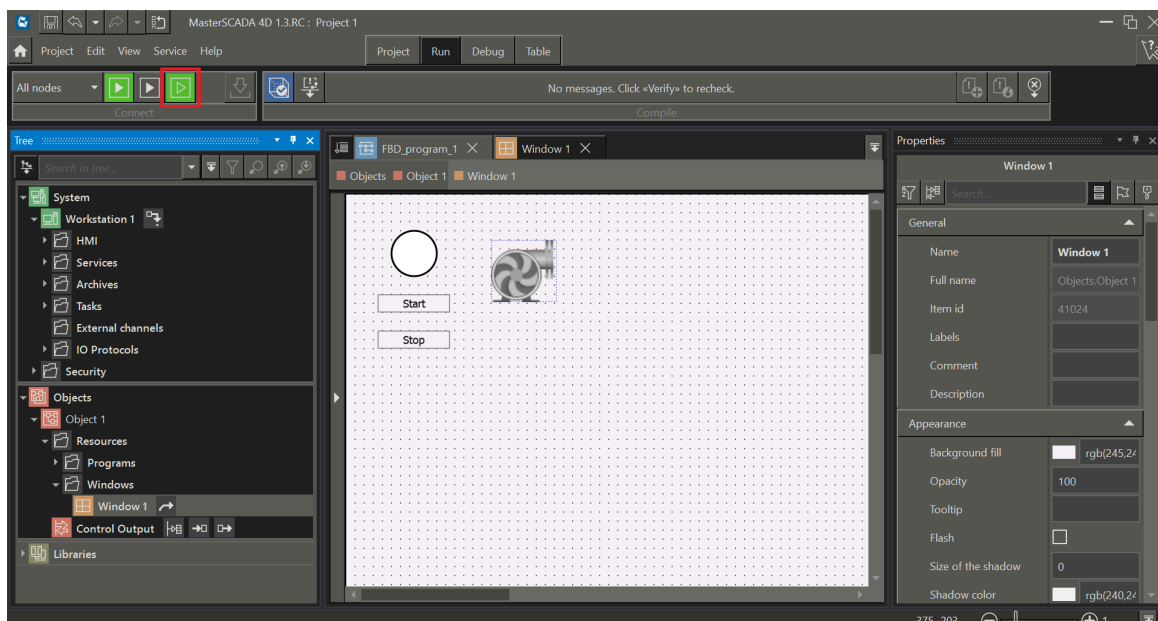


Figure 52

If runtime mode is started for the first time, a window will appear offering to choose for display in runtime mode either the HMI client developed by "IEK DIGITAL", or the browser assigned as default on the computer.

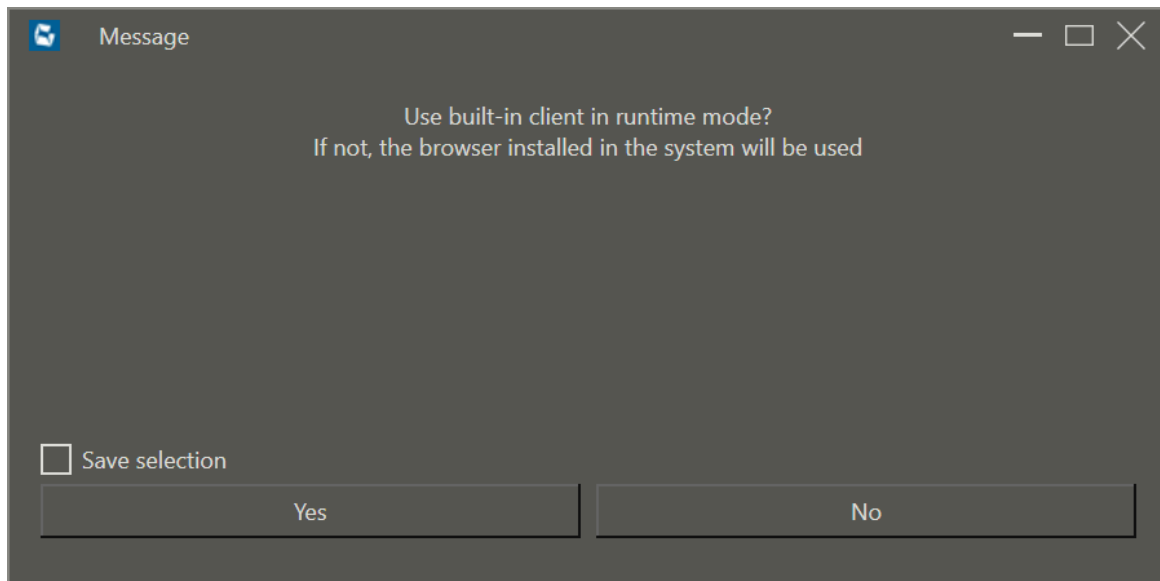


Figure 53

In our case, either choice is acceptable.

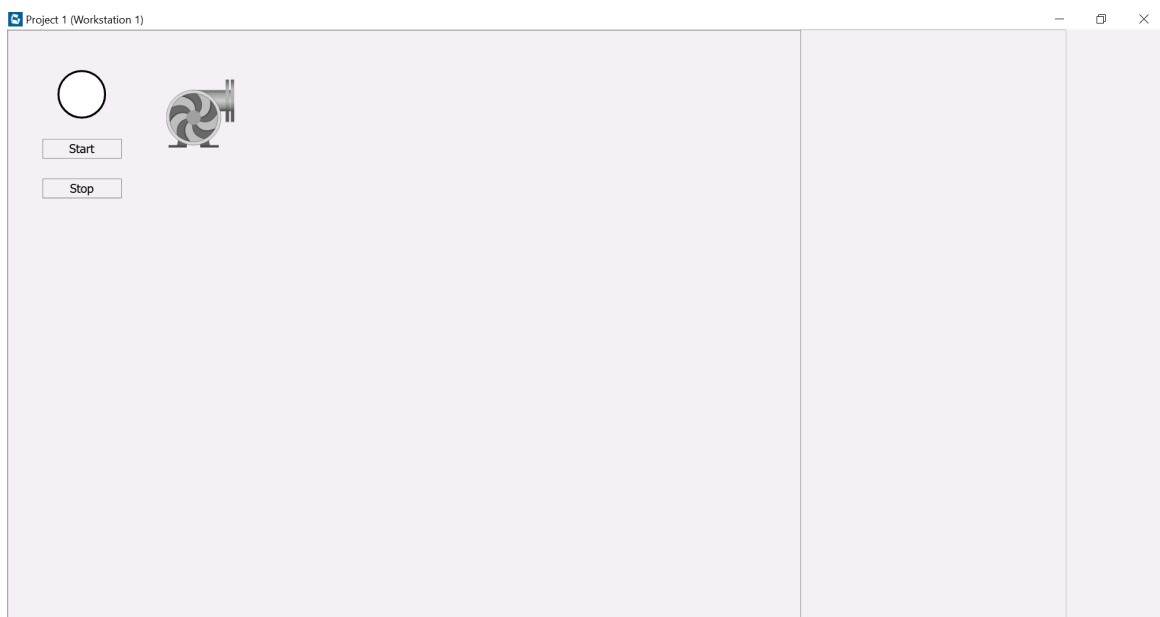


Figure 54

In the development environment, current values will appear next to parameter names.

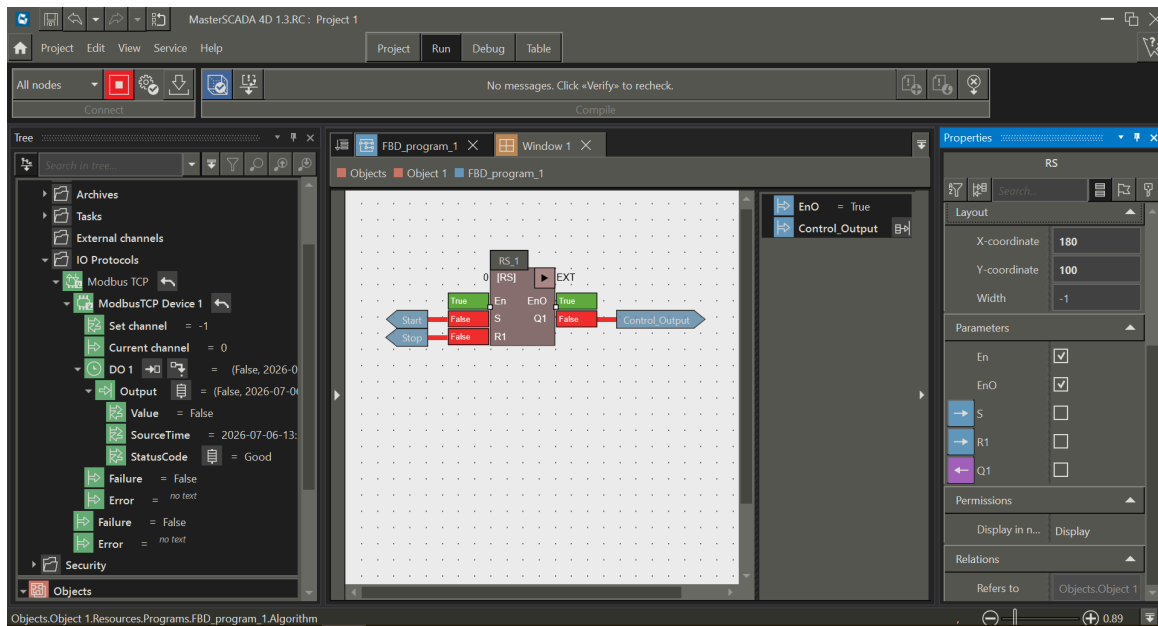


Figure 55

Click the **Start** button in runtime mode. **Control output** will take the value **TRUE**; this same value will be sent to the module discrete output, the indicator on the mimic diagram will turn green, and the pump will start rotating.

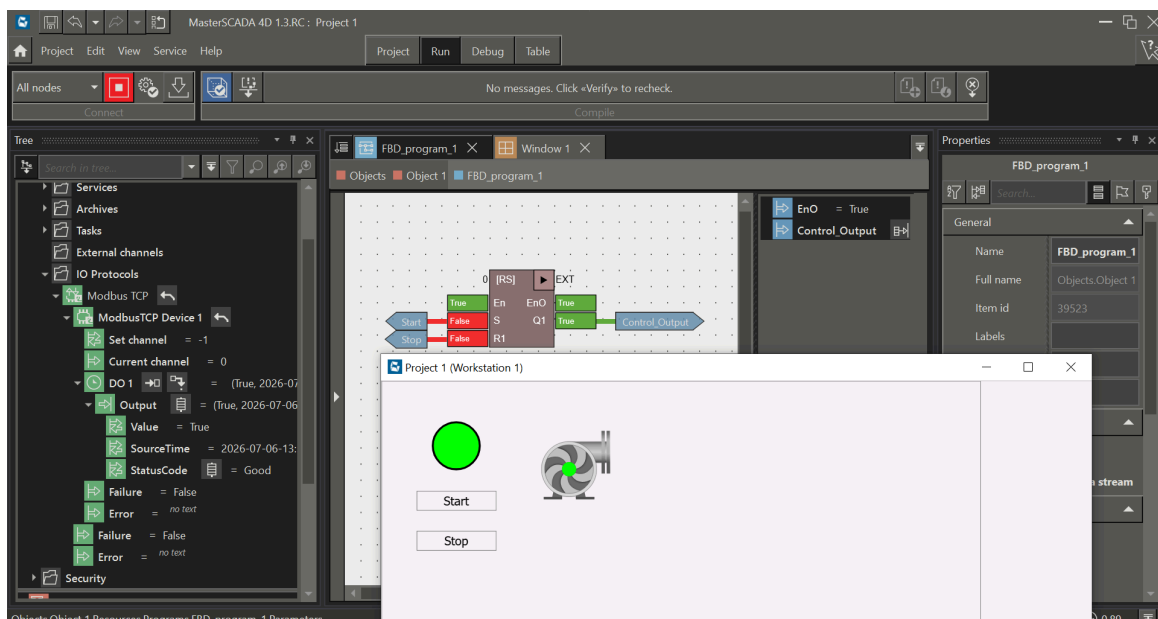


Figure 56

Click the **Stop** button. **Control output** will take the value **False**, the indicator on the mimic diagram will turn white, and the pump will stop.

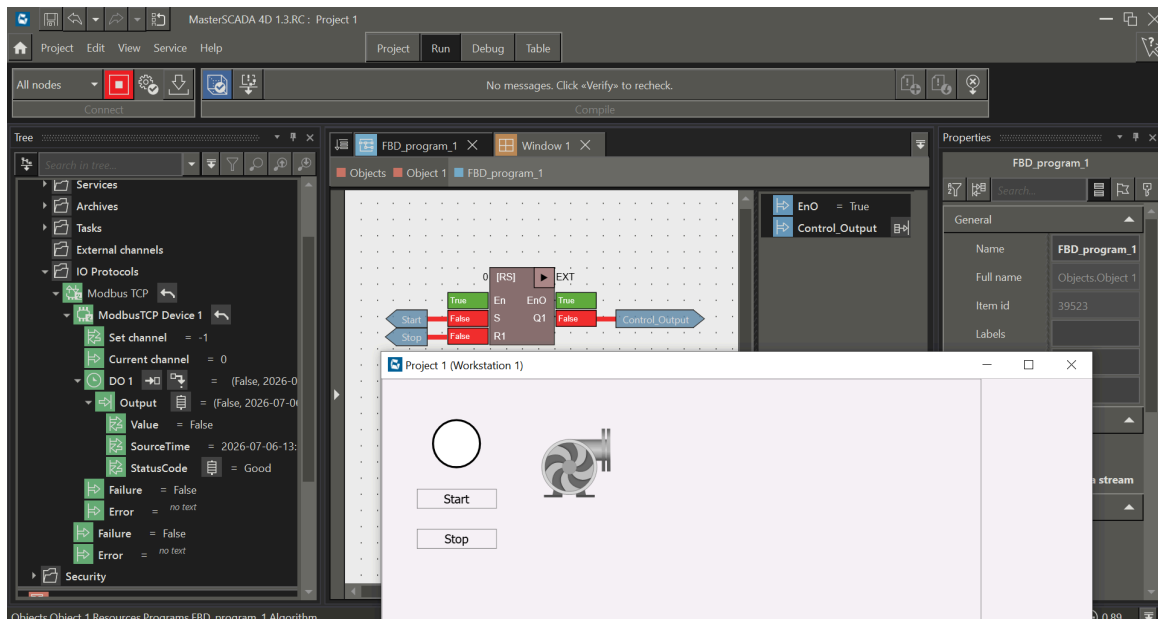


Figure 57

Why does this happen? While the **Start** button is pressed, the **Start** parameter value at the program input equals **TRUE**. This value is sent to the **S** trigger input, setting the trigger output to **TRUE** (or, as it is also said, "sets the trigger"). After the button is released, the trigger remains in the same state until the **Stop** button is pressed — this will cause **TRUE** to appear at the **R** trigger input and, accordingly, set the trigger output to **False** (or, as it is said, "reset the trigger").

Now, if your computer is accessible over the network from other devices, open a browser on another device, enter the IP address of the computer where you started runtime mode in the address bar, and verify that you can control the pump identically and in any sequence from any device.

If you have a controller connected to the network, you can download the program to it.

To do this, stop runtime mode, select the controller in the tree, make sure its IP address is specified, and instead of the **Emulation** button, click **Connect**.

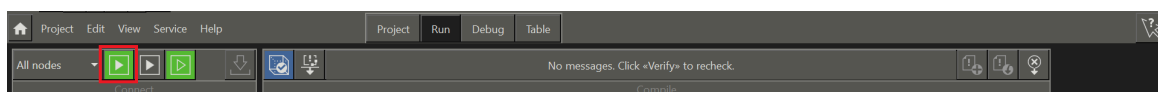


Figure 58

After that, the project will be downloaded to the controller and started.

QS. Lesson 2

In [Lesson 1](#) we created a pump control project using two buttons and a visual display of the pump state. In **Lesson 2** we will modify the previously created project and implement control of the same pump in a separate pop-up window.

We will continue from where **Lesson 1** left off. Therefore, if you have not completed it yet, we strongly recommend performing all the actions described in it first — it is not difficult. If you have already completed **Lesson 1**, simply open that project!

- [QS. Lesson 2. Configuring a pop-up window](#)
- [QS. Lesson 2. Launch](#)

QS. Lesson 2. Configuring a pop-up window

First, add another **window** to **Object 1**. To do this, right-click to open the context menu, then select **Add - Window**:

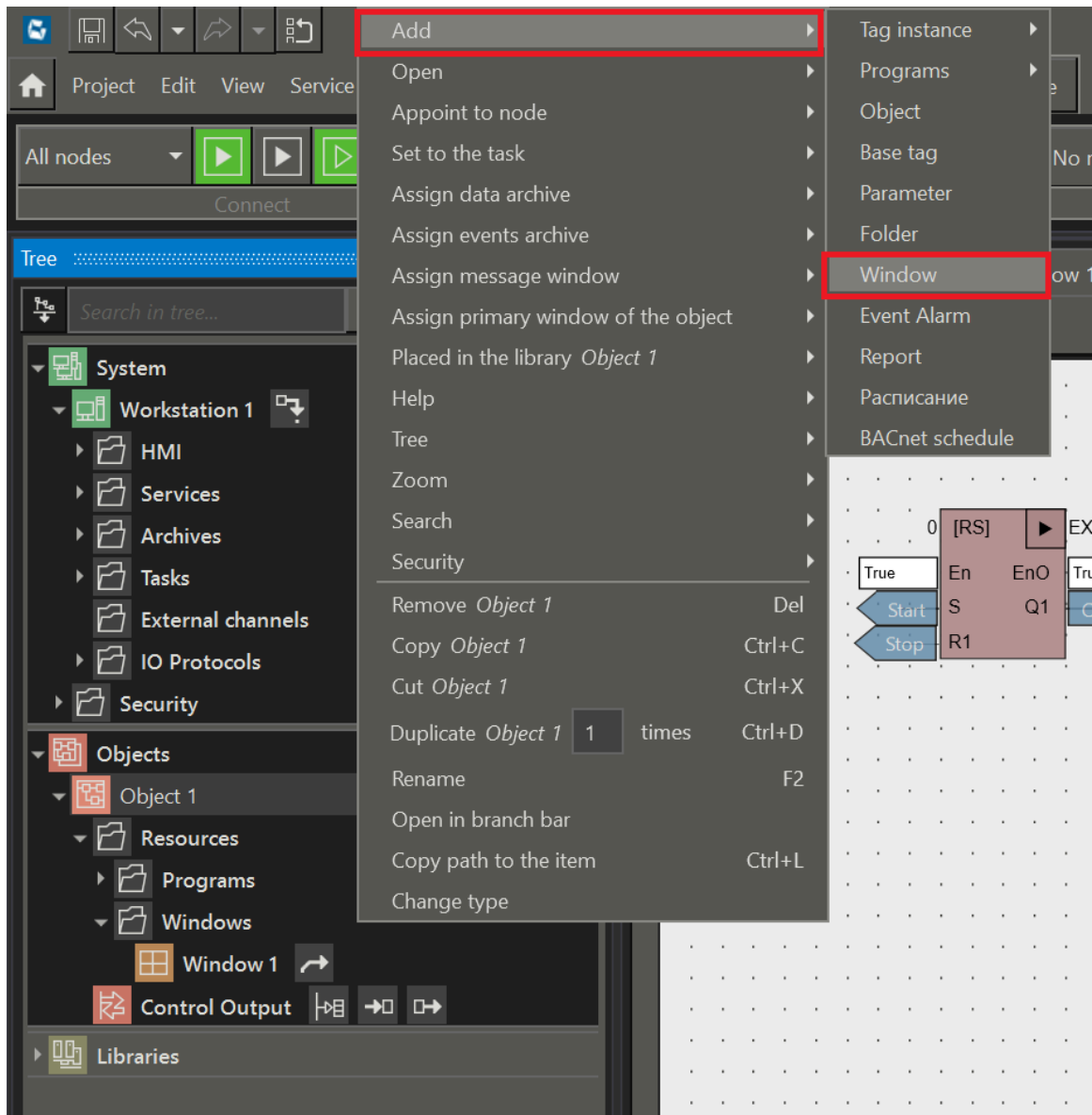


Figure 59

Let the new item be called **Control window**. To change the name assigned to the added item by default, you can use the **properties panel**, or open the context menu and select **Rename**, or double-click the item in the tree and enter a new name using the keyboard in the field that appears:

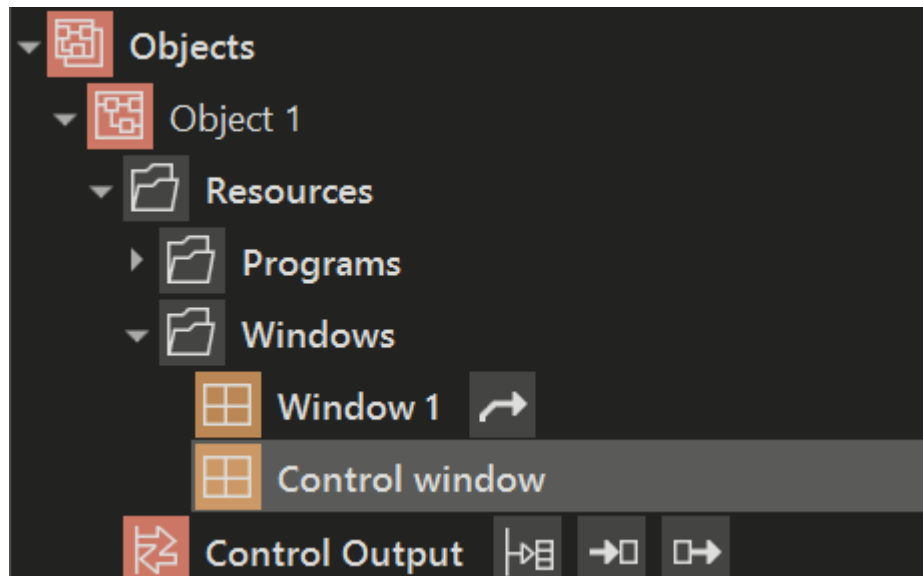


Figure 60

After the window is created, the graphic editor will open automatically:

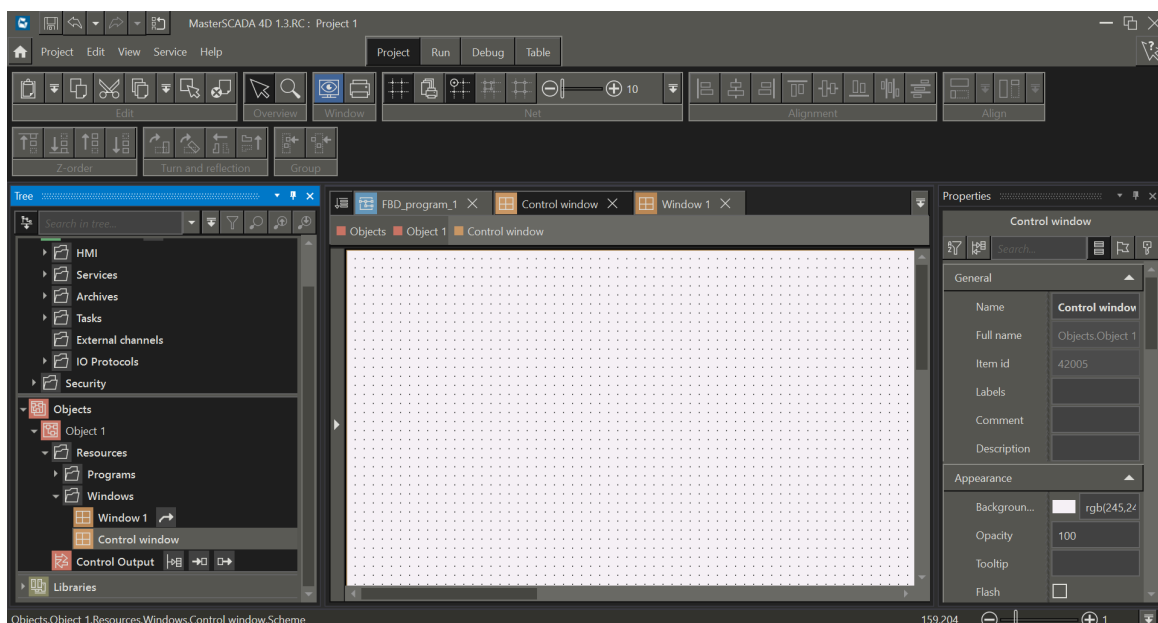


Figure 61 – After the window is created, the graphic editor will open automatically

Move the control and indication items from the main window into it.

To do this, switch to the **Window 1** tab, select both buttons and the indicator with the mouse, and cut these items either by pressing **Ctrl+X**, or using the **Cut** button on the toolbar, or via the context menu of the selected items.

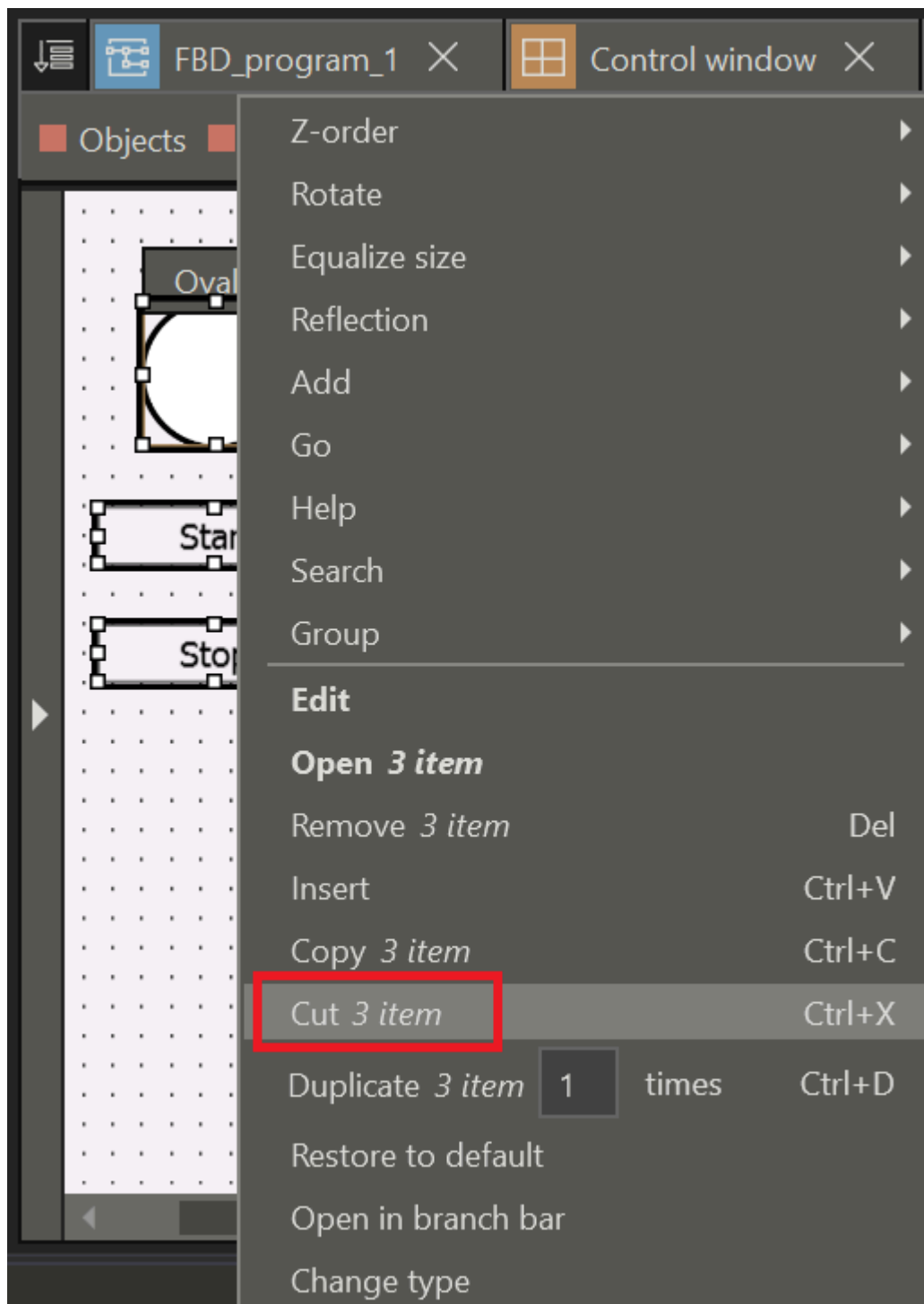


Figure 62

Now switch to the **Control window** tab and, either via the **Paste**, context menu or by pressing **Ctrl+V**, paste the cut items there.

A link restoration dialog box will appear:

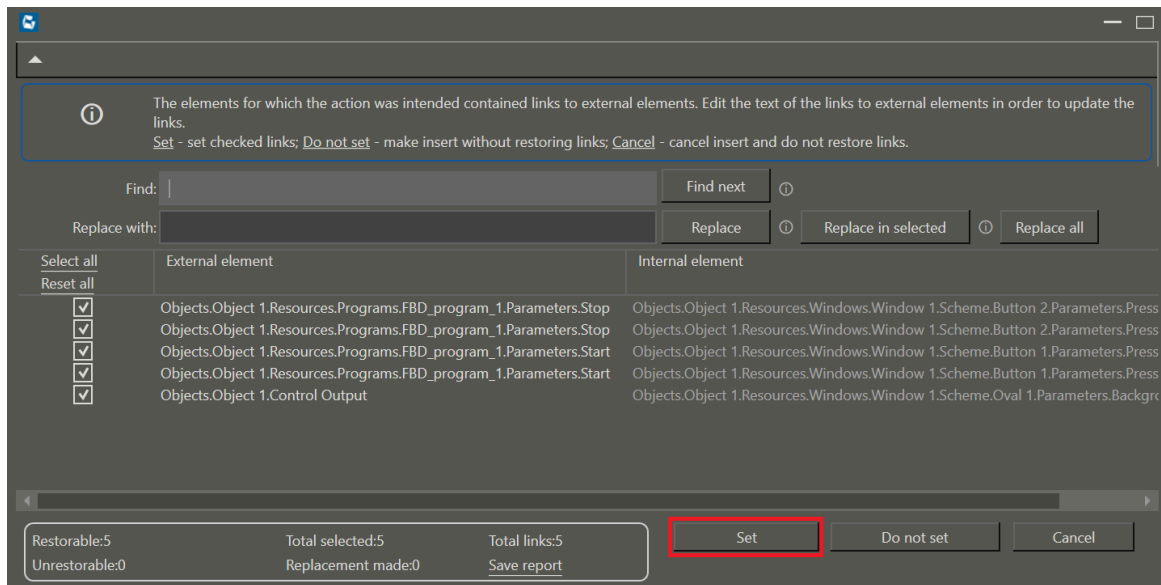


Figure 63 – A link restoration dialog box will appear

Click the **Set**, then reduce the window size by changing the window size parameters in properties or by dragging the right border to the left and the bottom border upward:

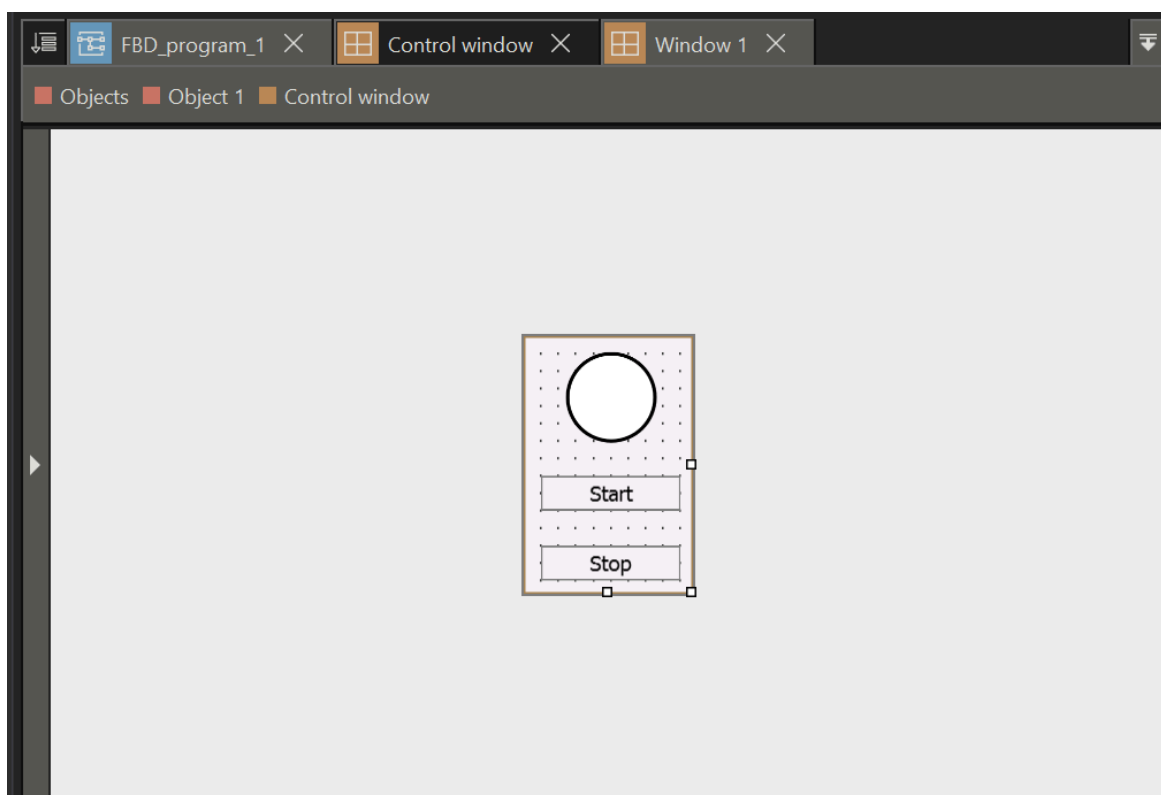


Figure 64

Now we need to make this window open when clicking the pump image.

To do this, switch to the **Window 1** tab, which is the main window. Select the pump image in the window by left-clicking it. **Properties** panel switch to event display mode:

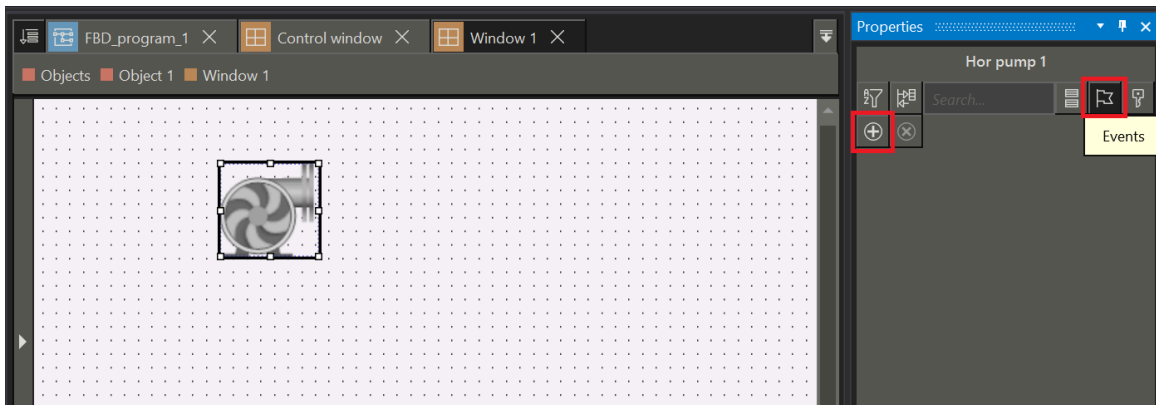


Figure 65

Right-click the **Mouse click** event. From the context menu select **Add**, and from the list of available actions — **Open pop-up window**:

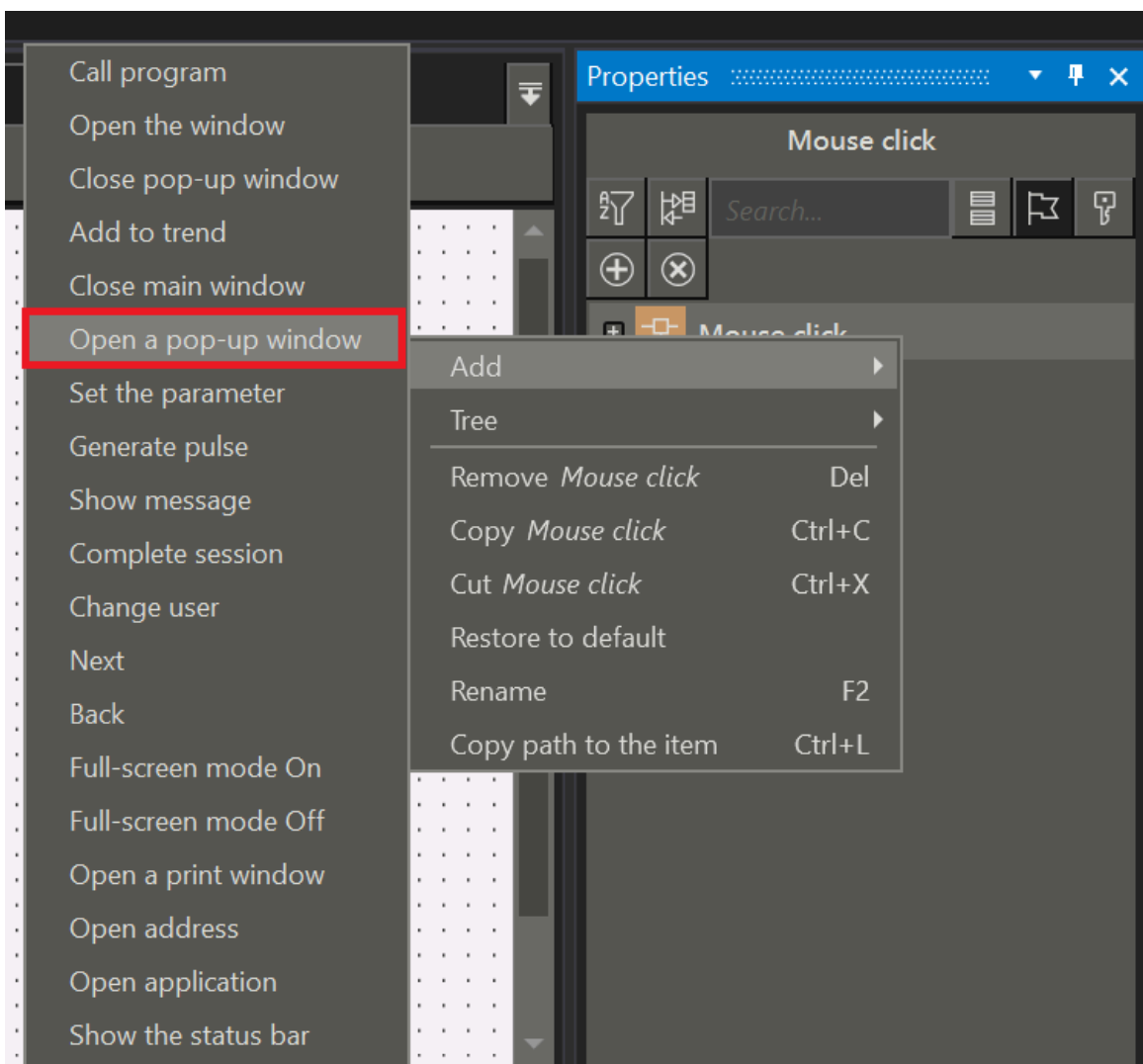


Figure 66

Expand the detailed parameters of this action. Specify which window will open on mouse click. To do this, simply drag the required window (in our case, this is **Control window**) to the **Window** action parameter. As a result, the window name will appear as the value of the **Window** parameter.

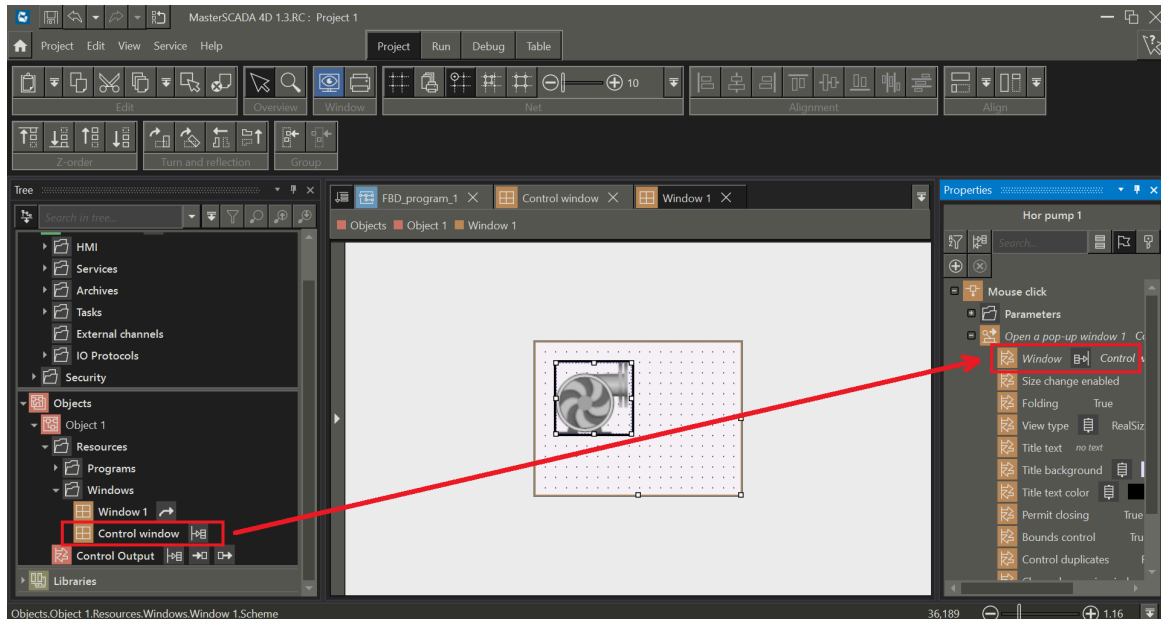


Figure 67

Also specify the position of the pop-up window relative to the item:

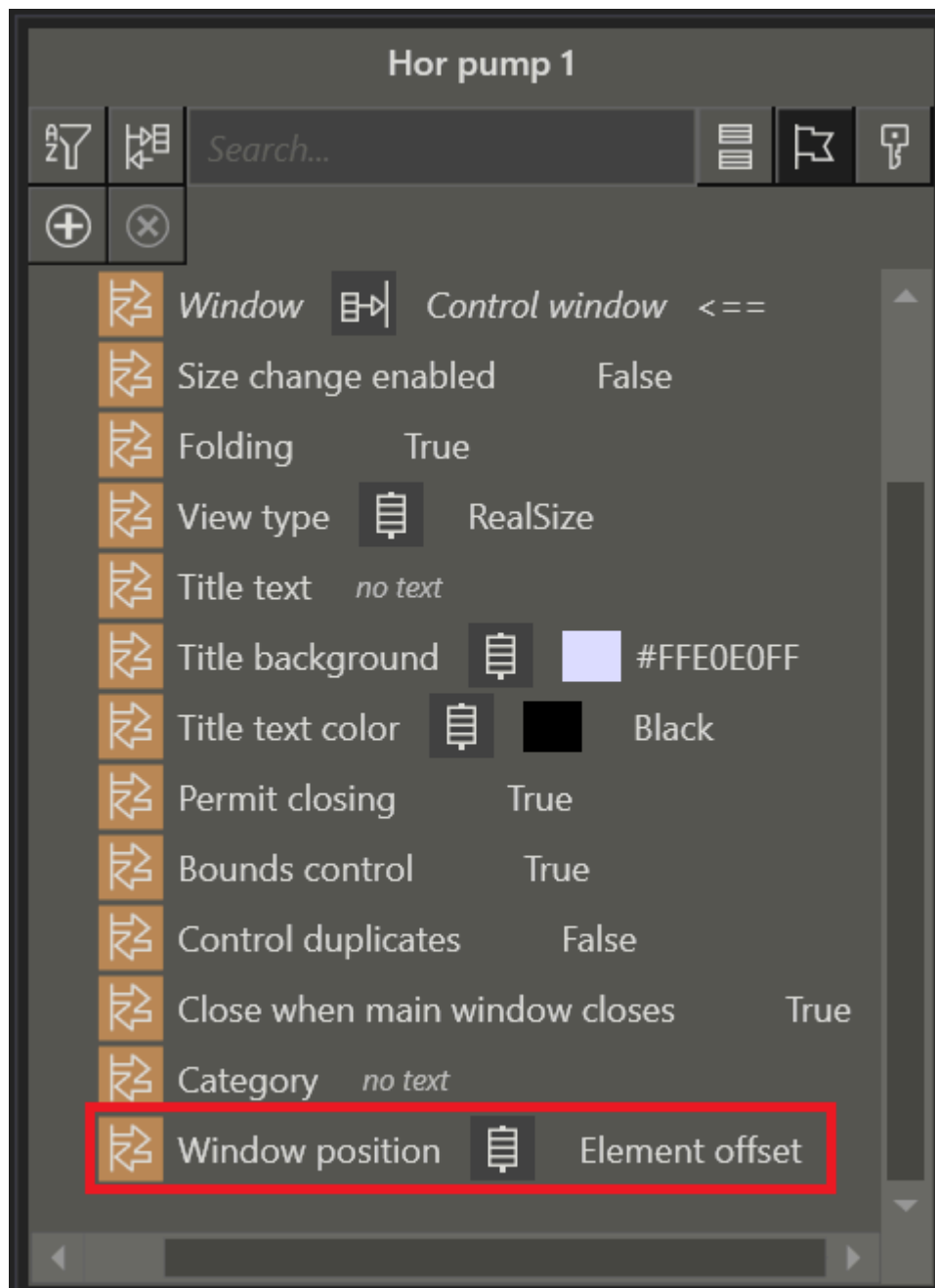


Figure 68 – Also specify the position of the pop-up window relative to the item

Move the **Pump** item to the left:

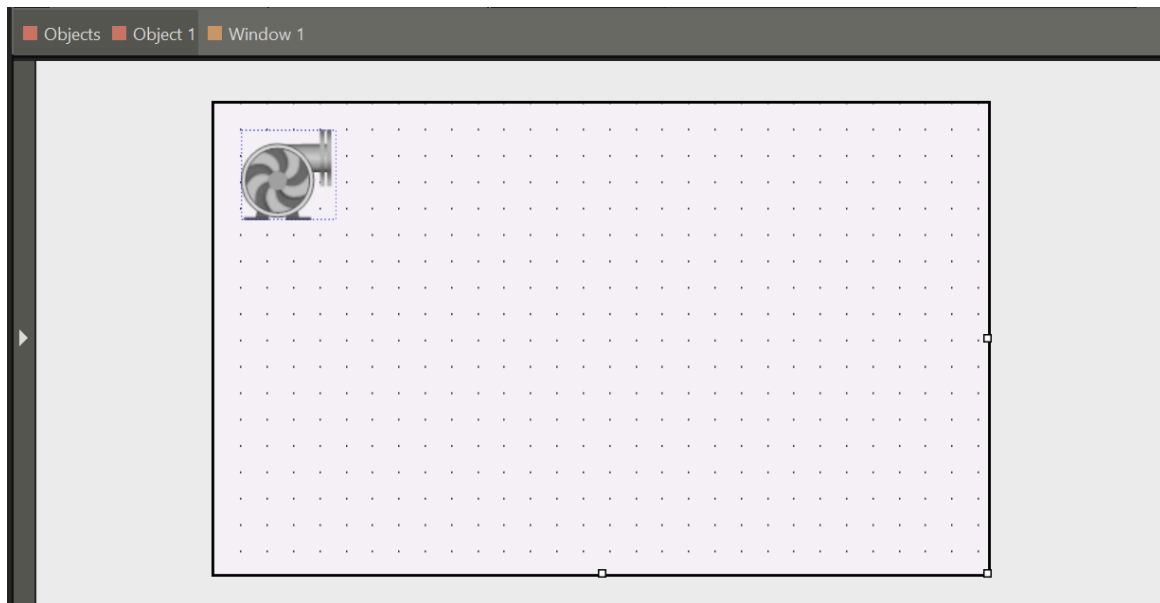


Figure 69 – Move the Pump item to the left

At this point, configuration of our simple task is complete.

QS. Lesson 2. Launch

First, switch to the **Runtime** tab and, by clicking the **Emulation** icon on the toolbar, check the result.

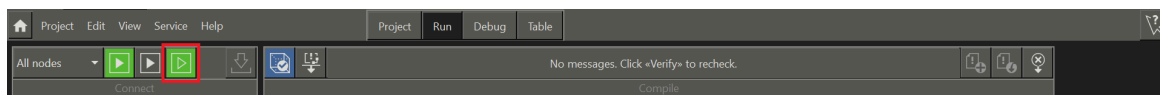


Figure 70

Emulation mode will start and the built-in HMI client (or browser) will open, displaying the familiar pump image. Left-click it — a pop-up window with **Start**, **Stop** buttons and a state indicator will open. Click the **Start** button — the pump will turn on.

 Project 1 (Workstation 1)

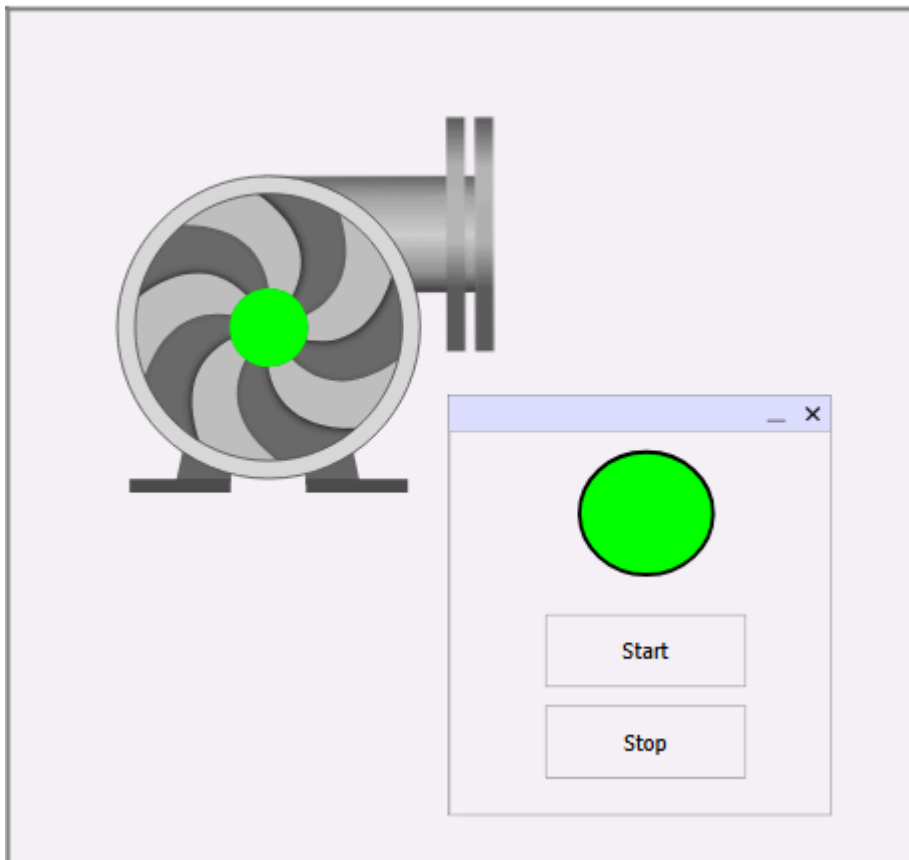


Figure 71

Click the **Stop** button — the pump will stop. Everything works as in Lesson 1, except that the control items are placed in a separate window.

QS. Lesson 3

In the previous lessons we practiced the object-oriented approach to project creation, but did not explain why it is needed. Now it is time to look at object-oriented design in more detail.

- [QS. Lesson 3. Object-oriented approach to project creation](#)
- [QS. Lesson 3. Launch](#)
- [QS. Lesson 3. Library operations](#)

QS. Lesson 3. Object-oriented approach to project creation

First, let us consider what an **object** is. In MasterSCADA 4D an **object** is a named collection that combines heterogeneous project items and the links between them.

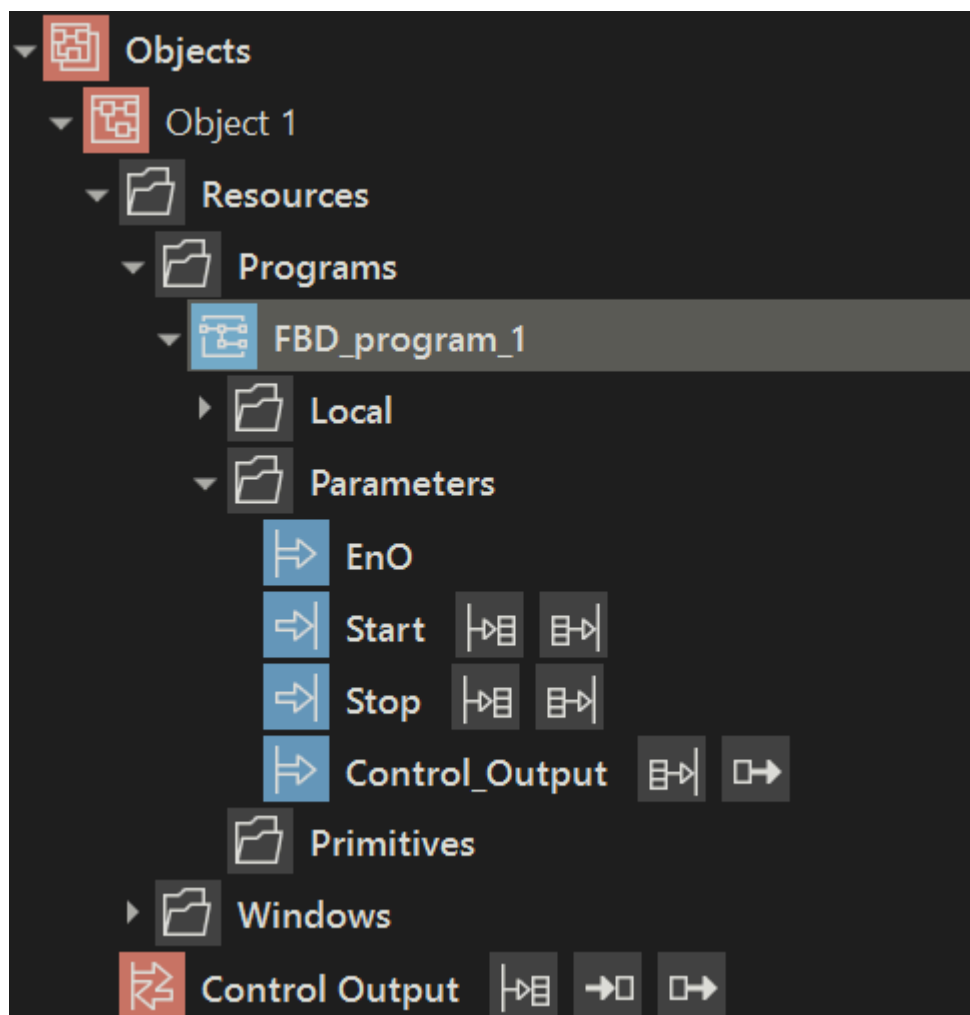


Figure 72

An **object** can contain programs, windows, reports, and other items, including nested objects. Various parameters of these items can be linked to each other. The object itself also has parameters — they are intended for linking it to the "external world", something like a terminal block. The main purpose of this object organization principle is to simplify replication. A copy of such an object is created with the entire internal structure preserved, including inter-item links within the object. The developer will only need to connect the external ones.

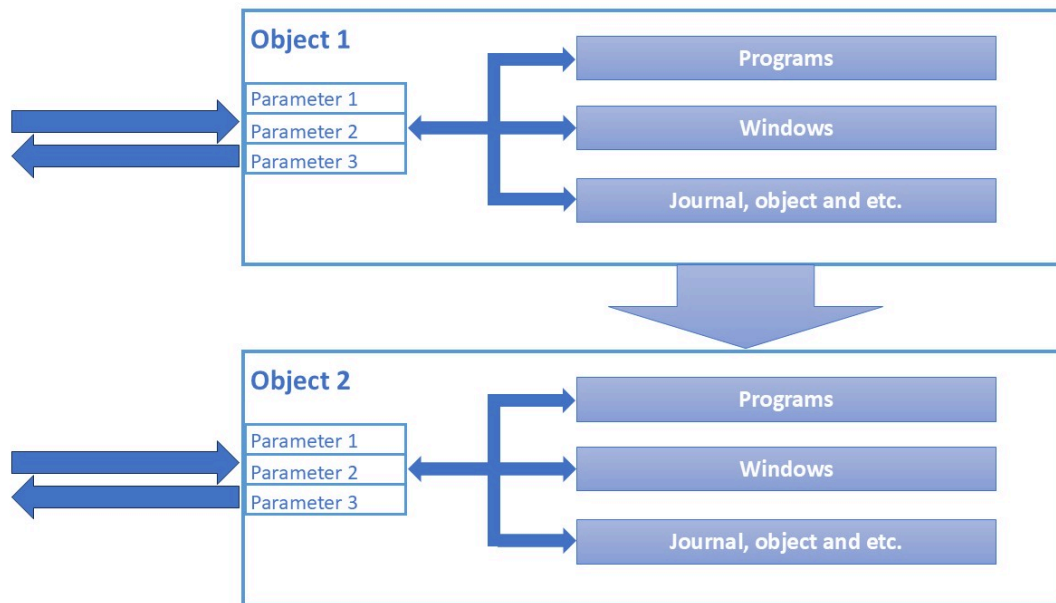


Figure 73

Now let us see how this can be applied in practice.

In the previous lesson we created an object consisting of a window with a pump image, a window for controlling this pump that opens on mouse click on the pump image, and a program implementing the pump control logic. The object has one parameter for linking to the external world — the control output, whose signal is sent to the physical discrete output of the controller module.

Now let us set the goal of creating a pump station consisting of several pumps.

Rename the previously created object to **Pump 1**.

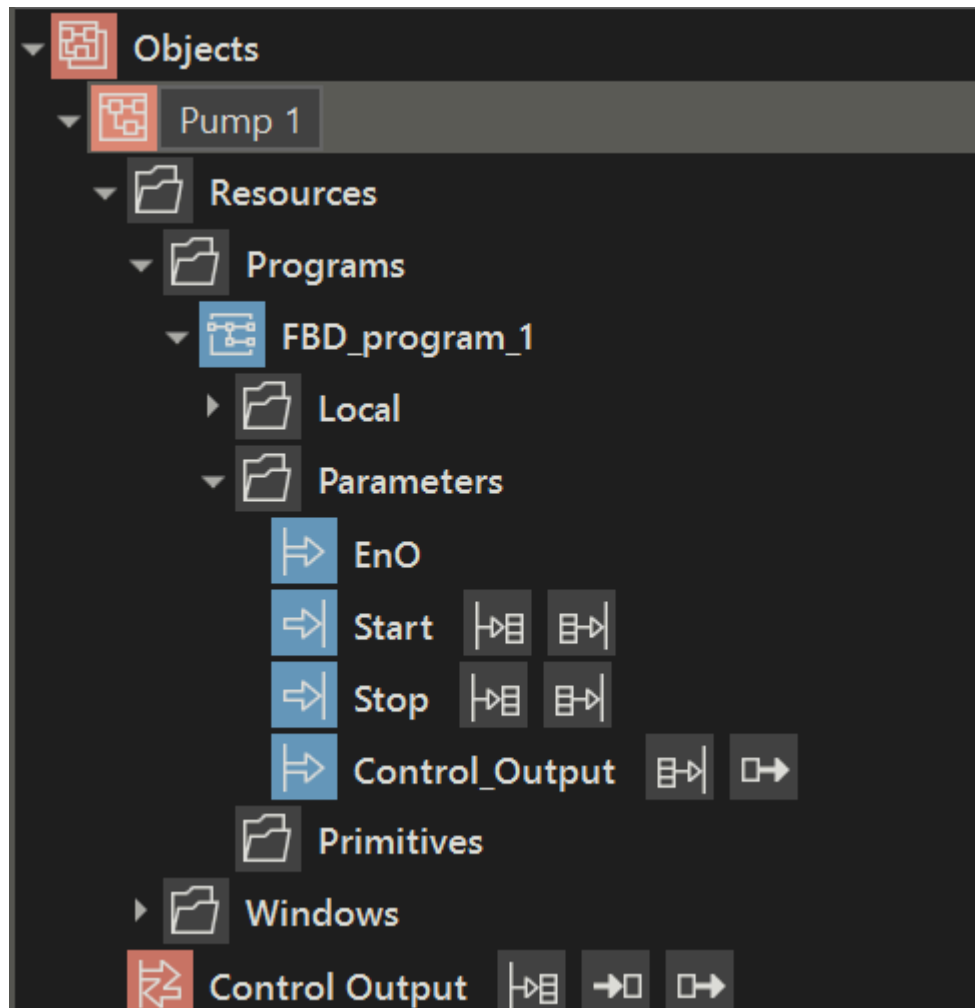


Figure 74

In the **objects tree** create another object and name it **Pump station**.

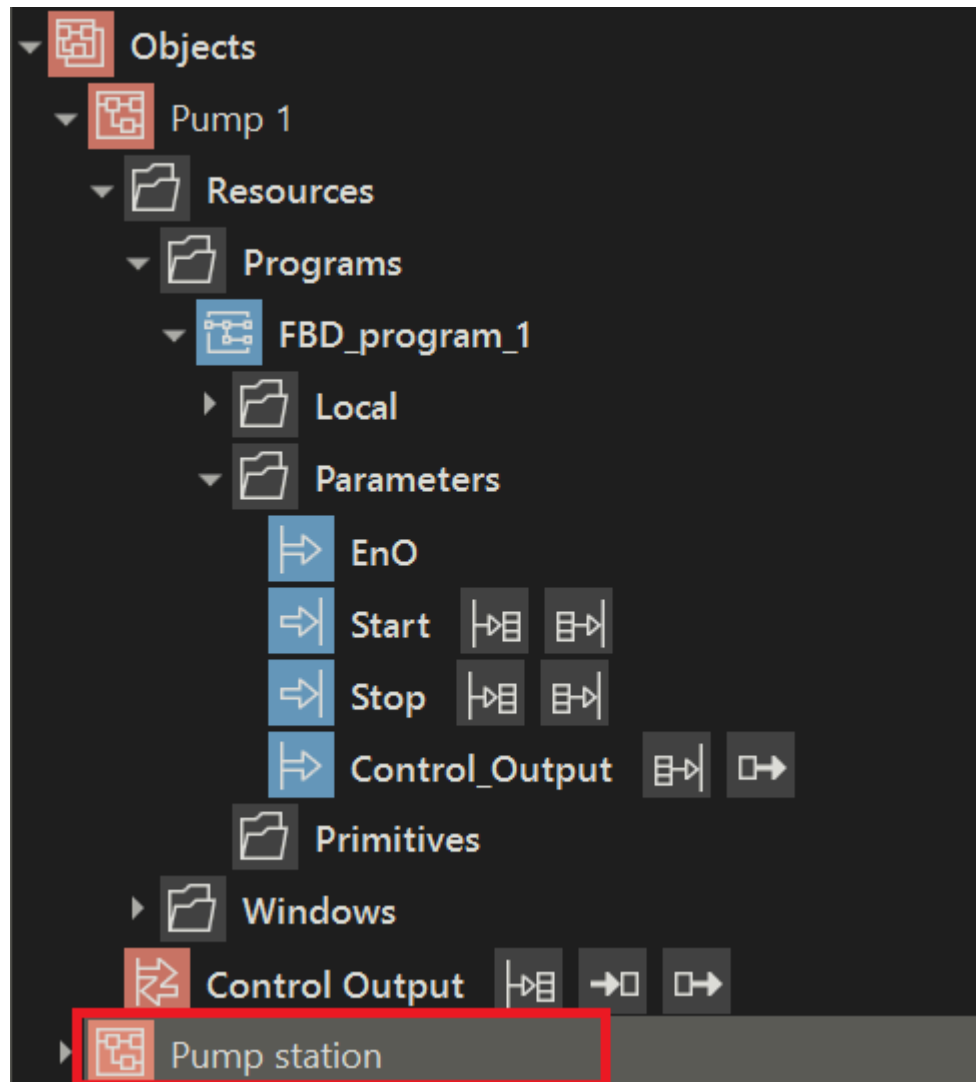


Figure 75

Now left-drag the pump into the pump station. We get a root object **Pump station** with a nested object **Pump 1**.

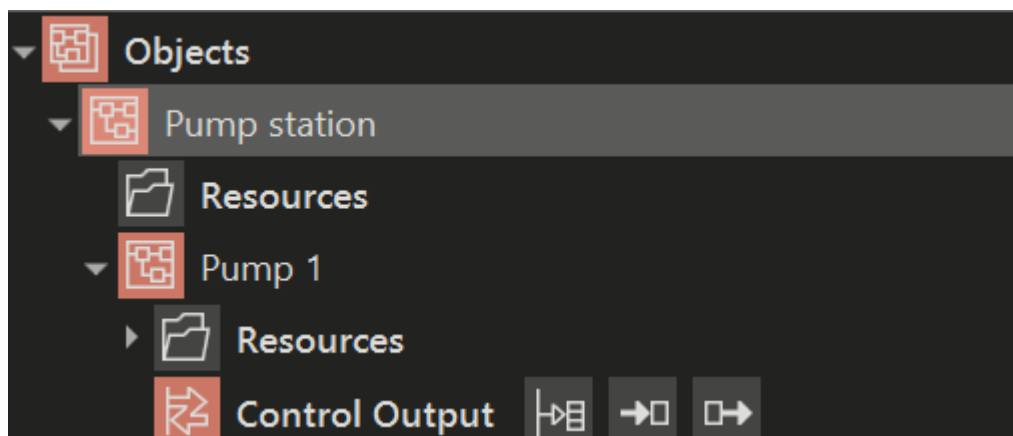


Figure 76

Let's place the **Pump 1** object into the **Local** library using the context menu.

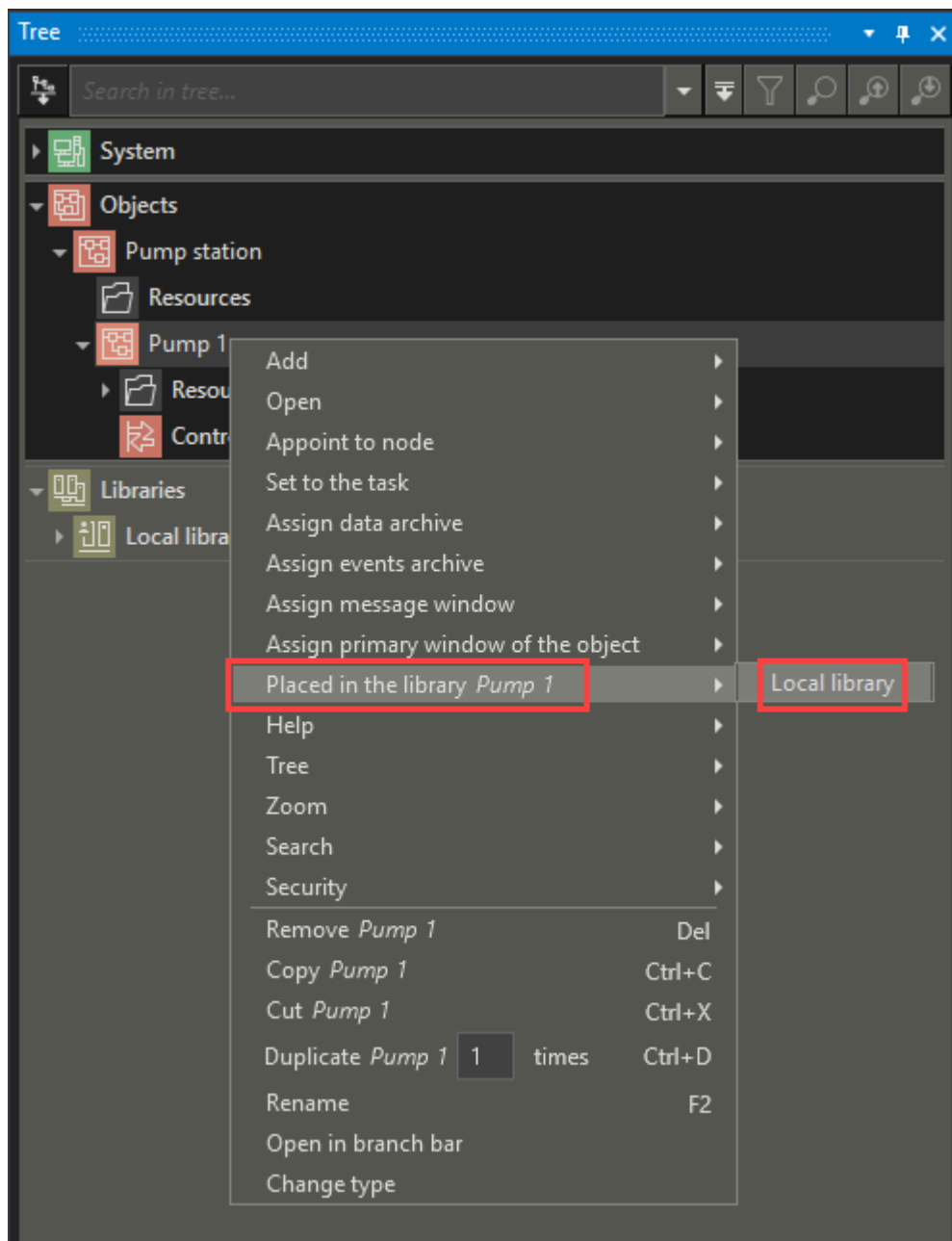


Figure 77

Now the pump has become a library item.

Next, we will replace the inheritors with instances. To do this, right-click on the library item **Pump 1** to open the context menu, and then select the **Convert Inheritors to Object Instances** item.

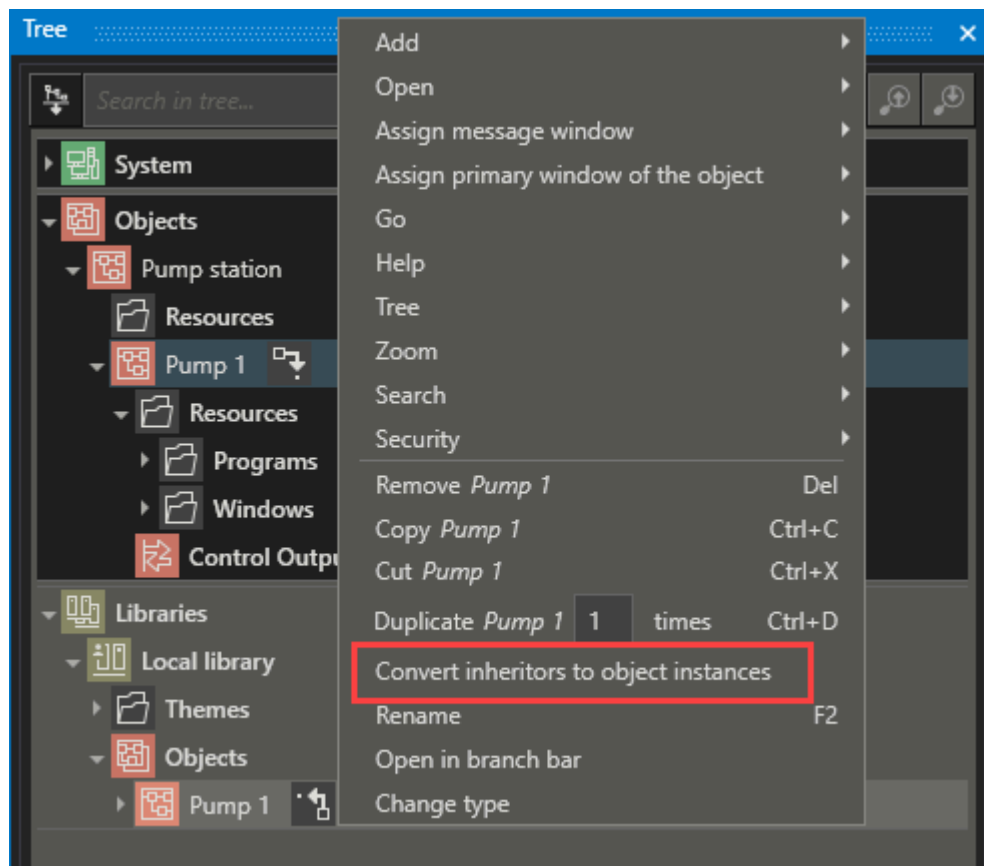


Figure 78

Now the pump in the object tree has become an instance of the library item, as indicated by the special icon next to its name and the tooltip.

Drag the library pump into the pump station. We now have two pumps.

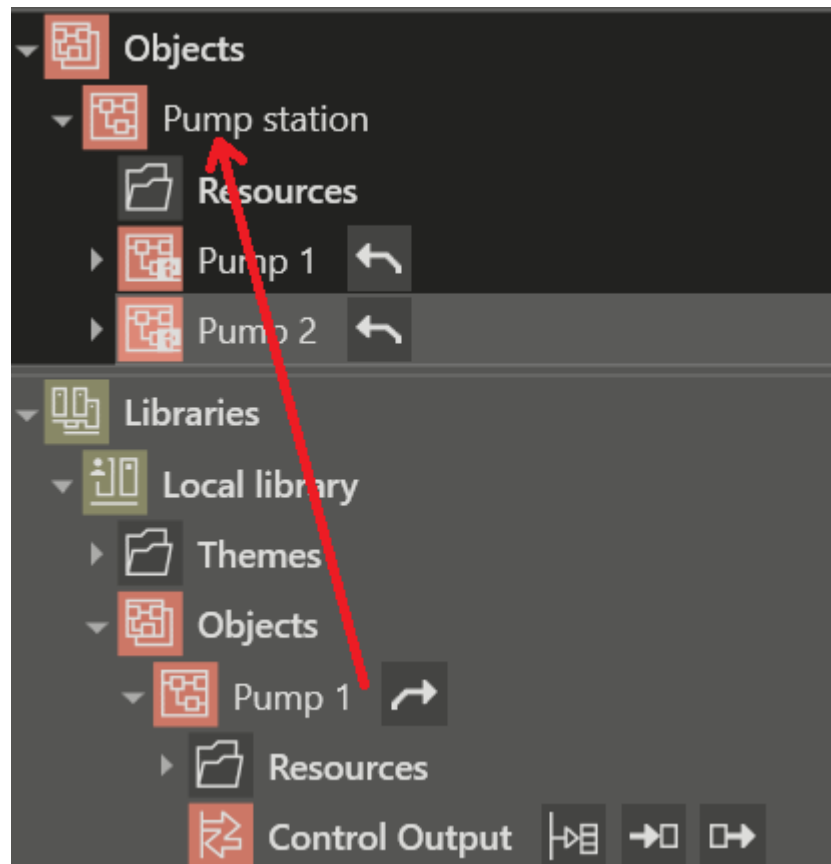


Figure 79

Repeat this procedure once more — the pump station now includes three pumps, each with one parameter for linking to external items or devices (**Control output**), and containing windows and a control program inside.

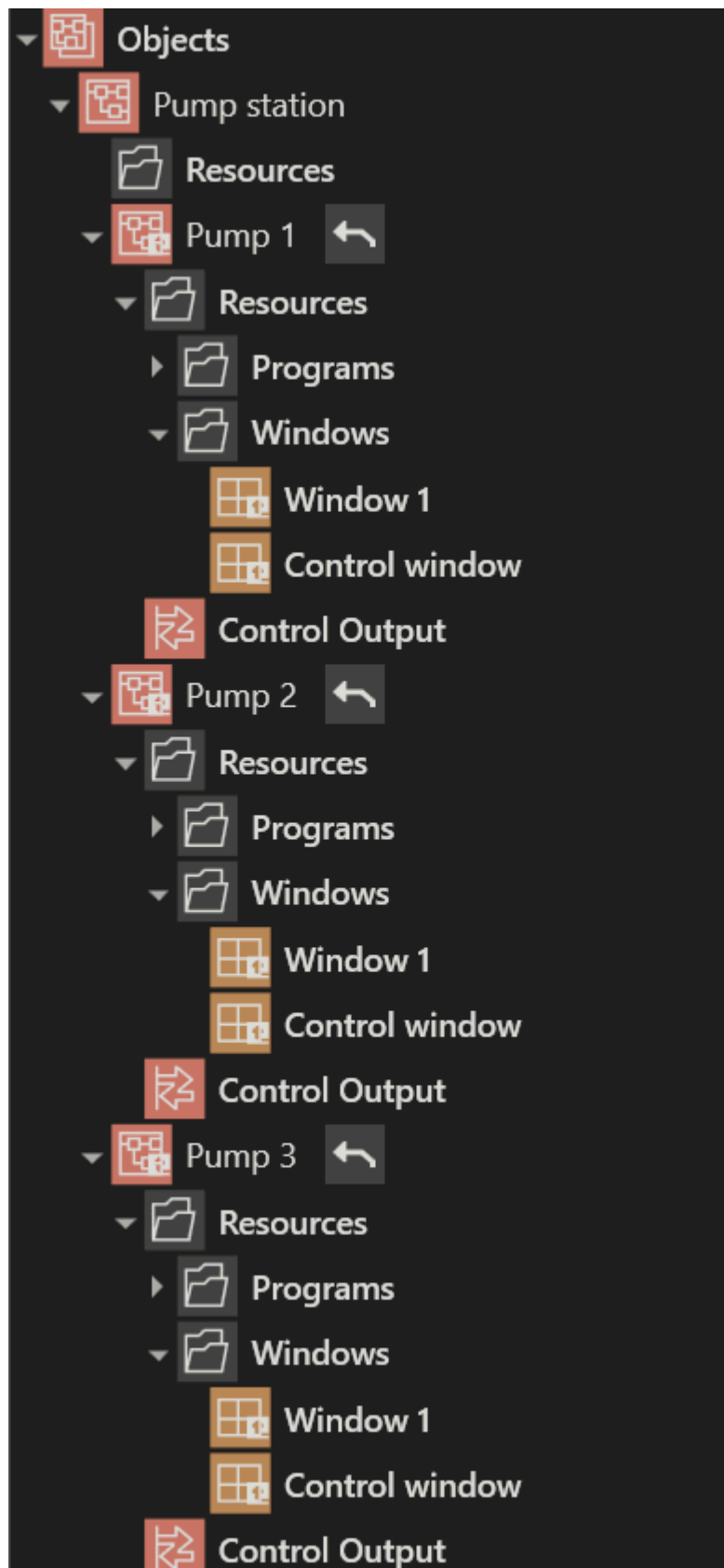


Figure 80

All pumps in the **objects tree** are instances of the library pump. This means that any change in the library pump will be reflected in all its inheritors, i.e., will be

inherited. For example, expand the library pump tree to the Windows folder contents and rename **Window 1** to **Pump image**:

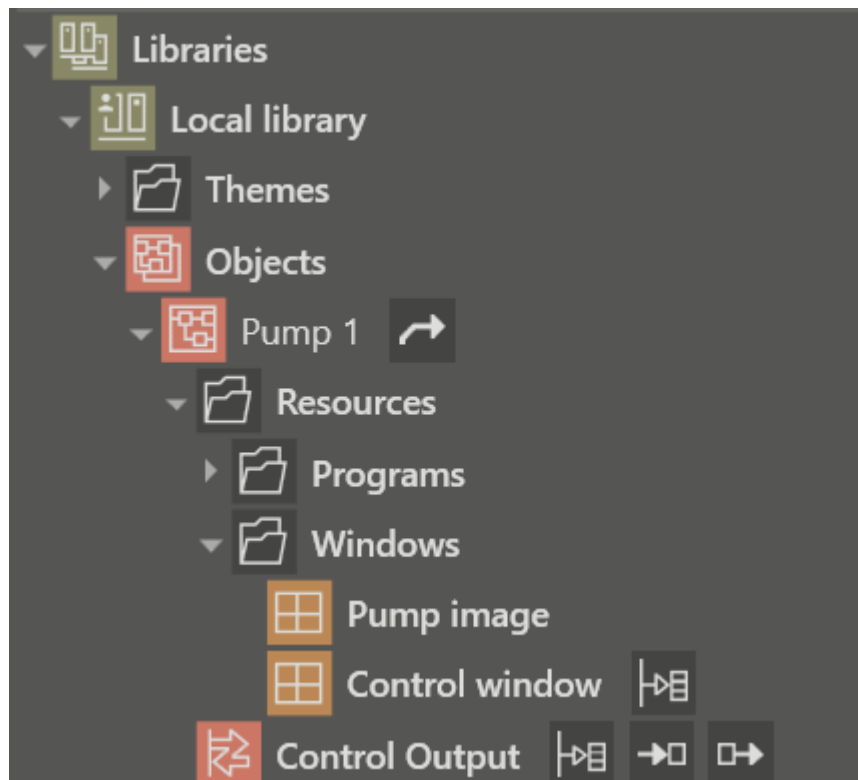


Figure 81

You will see that in the **objects tree** this window was also renamed:

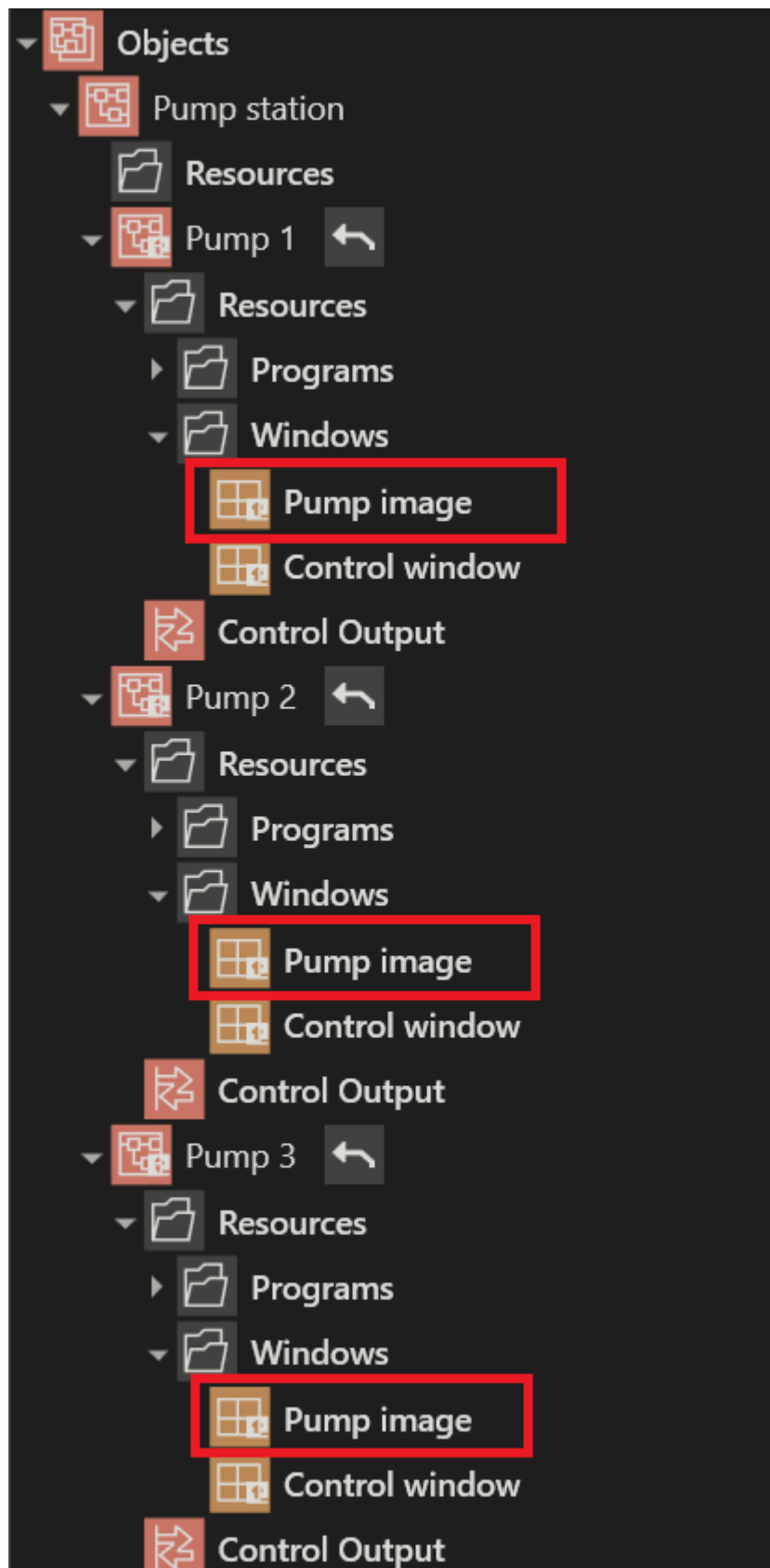


Figure 82 – You will see that in the objects tree this window was also renamed

That is, any change made to the library type is automatically and immediately applied to all its instances. Instances cannot be edited locally — their structure

and properties are fully controlled by the type. This ensures strict data consistency and guarantees that all instances of the same type will be identical.

Now let us make a few more changes. Open the **Pump image** window from the library and make it more compact (close the pump instance window — it cannot be edited):

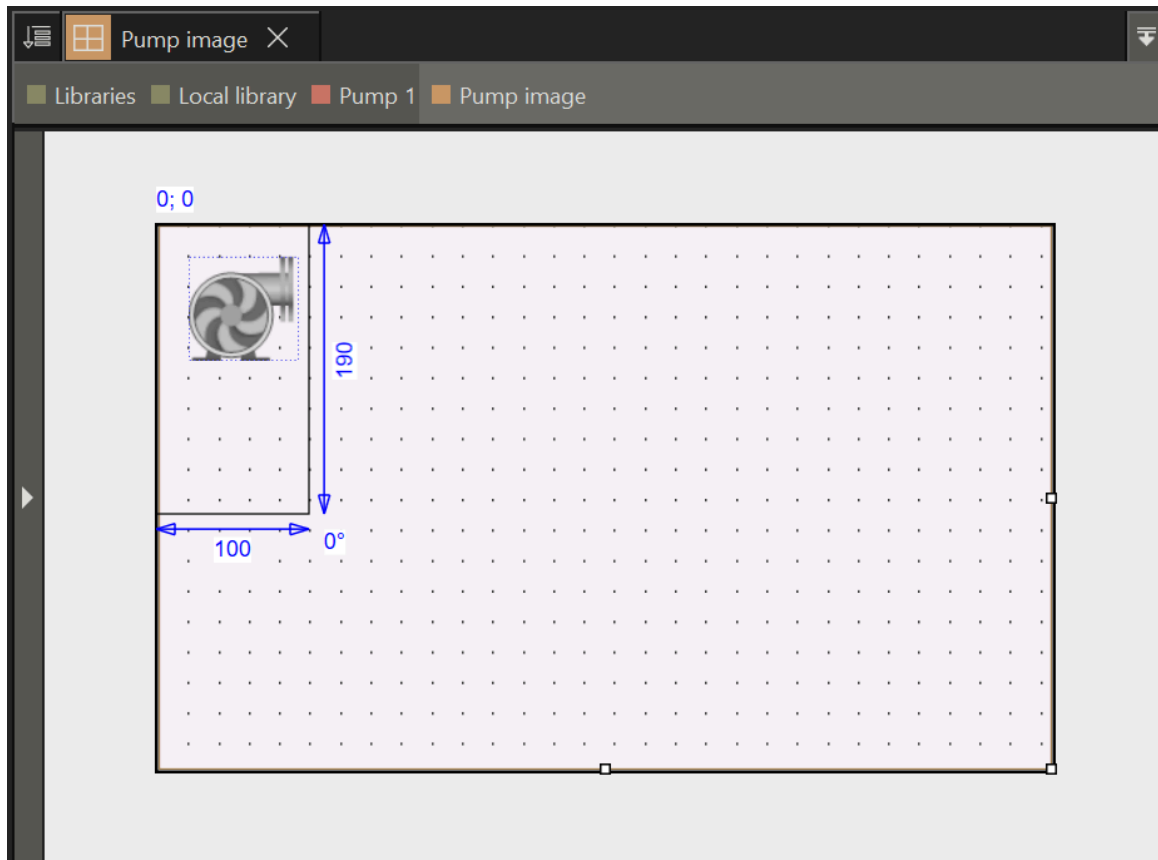


Figure 83

Add a label reflecting the pump state. To do this, right-drag the **Control output** parameter from the library pump (we are now working with the library item), and from the context menu that opens select **Create linked control.Text**:

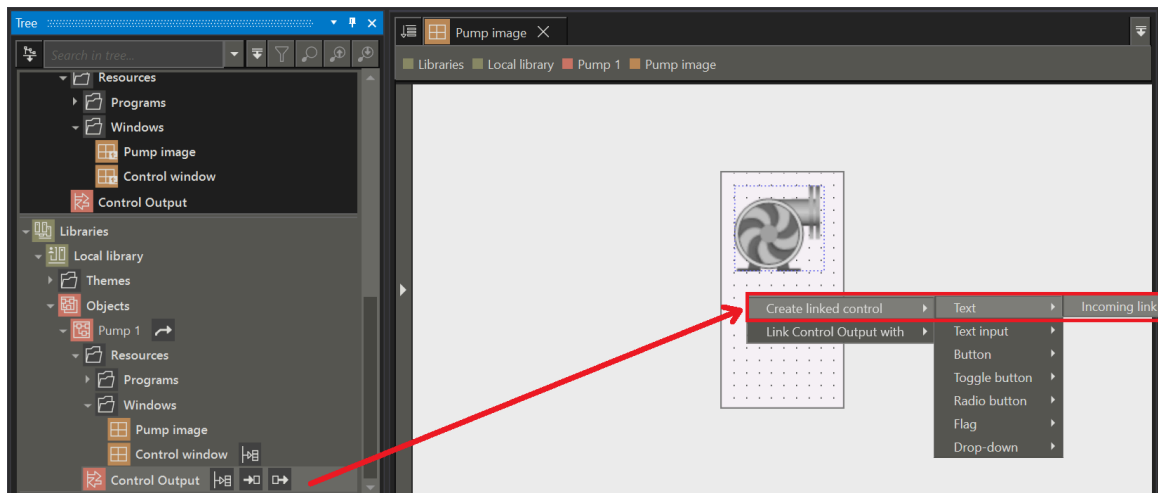


Figure 84

It is very common to display one or more main operating parameters next to an object image. For a valve, this is its position; for a motor with a frequency converter, it is the frequency, and so on. We will simply display the state. Go to the properties panel of the **Text** item, open the converter on the **Text** parameter link. Select point conversion, and for the **False** value specify **Stopped**, and for the **TRUE** value — **Running**.

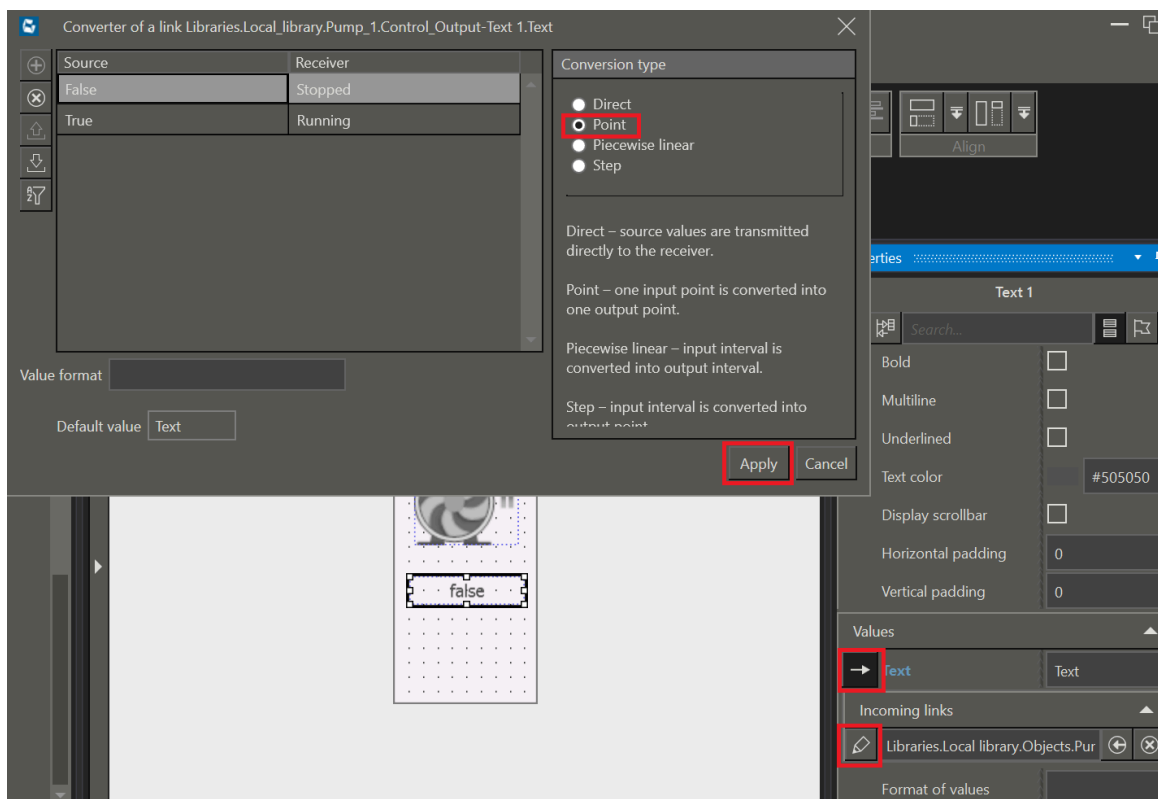


Figure 85

Close the converter by clicking **Apply**.

Reduce the window size to match the image:

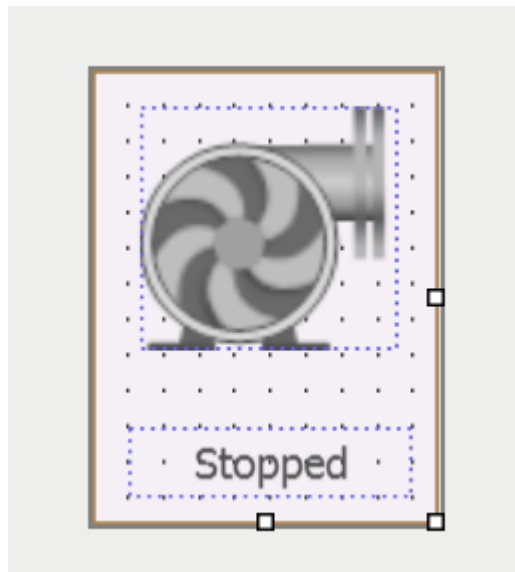


Figure 86 – Reduce the window size to match the image

Add a window in the pump station and name it **Mimic diagram**:

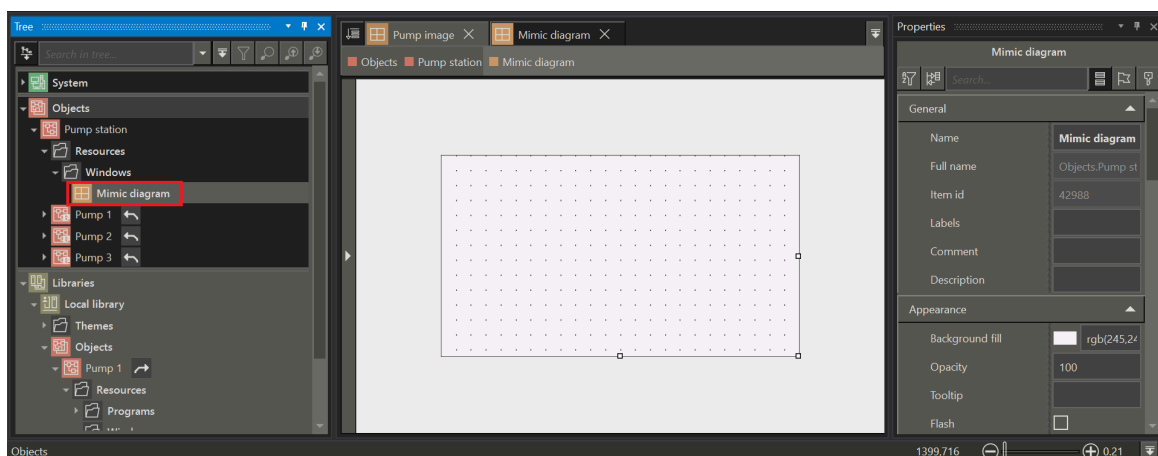


Figure 87 – Add a window in the pump station and name it Mimic diagram

Set **Mimic diagram** as the node startup window using the context menu:

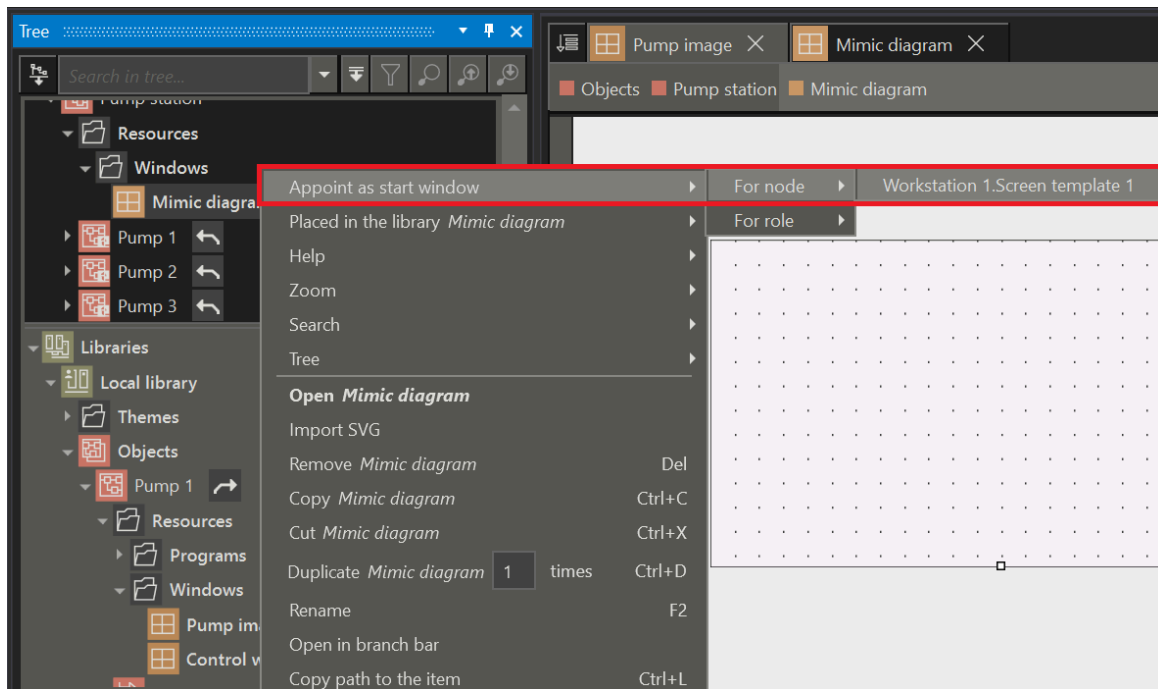


Figure 88 – Set Mimic diagram as the node startup window using the context menu

Now drag all pumps into the window one by one:

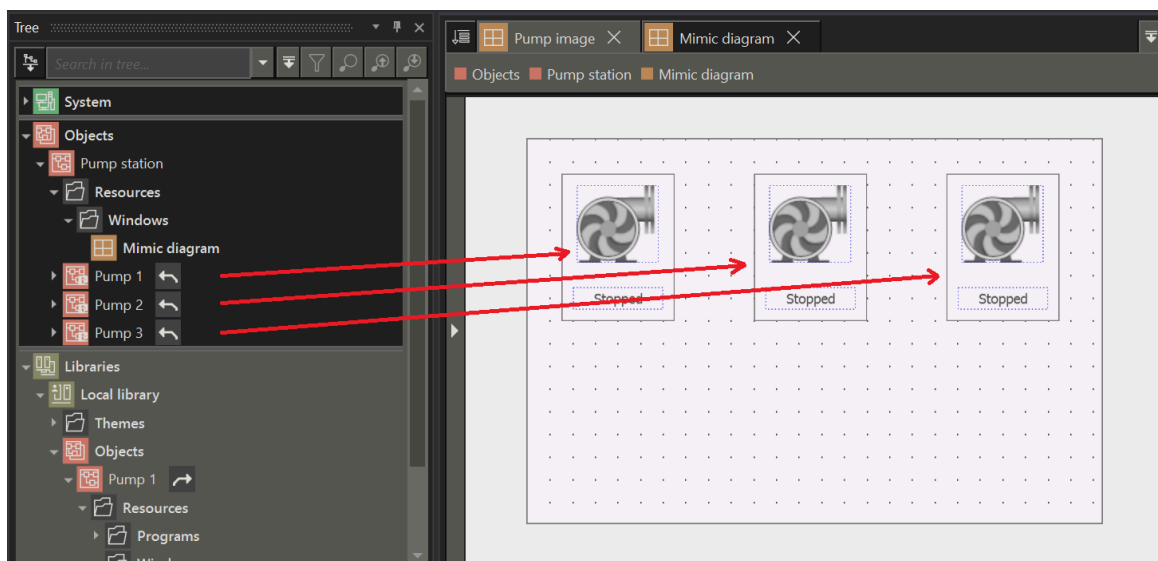


Figure 89 – Now drag all pumps into the window one by one

Link the outputs of the new pumps to the physical outputs of the I/O modules (we connected them in **Lesson 1**). Linking is more convenient when the trees of the linked items are opened in parallel windows. Right-click the **Pump station**, object and from the context menu select **Open in branch panel**:

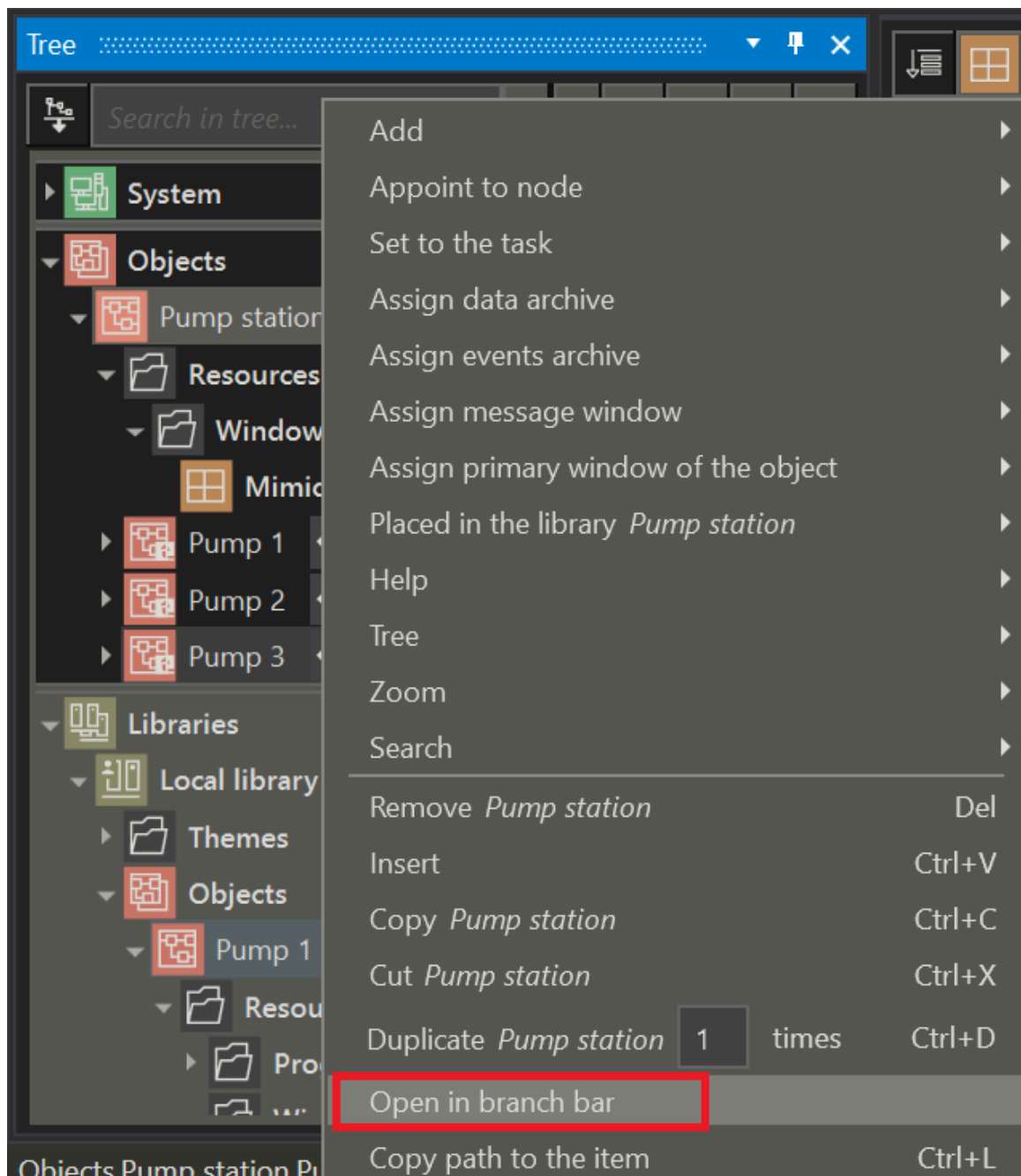


Figure 90

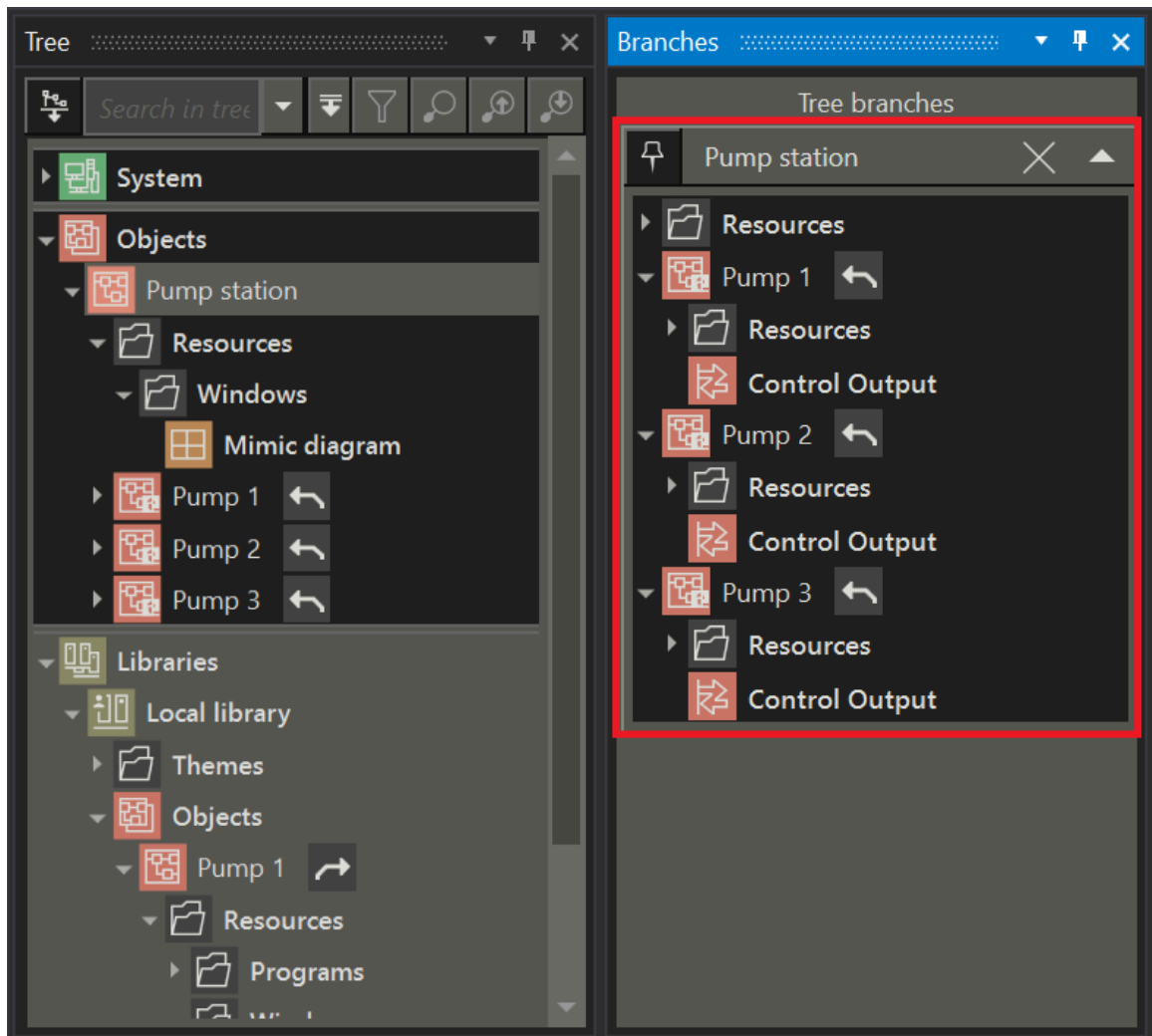


Figure 91

The first pump is already linked. Now link the **Control output** parameter of the second and third pumps to the subsequent discrete outputs of the module:

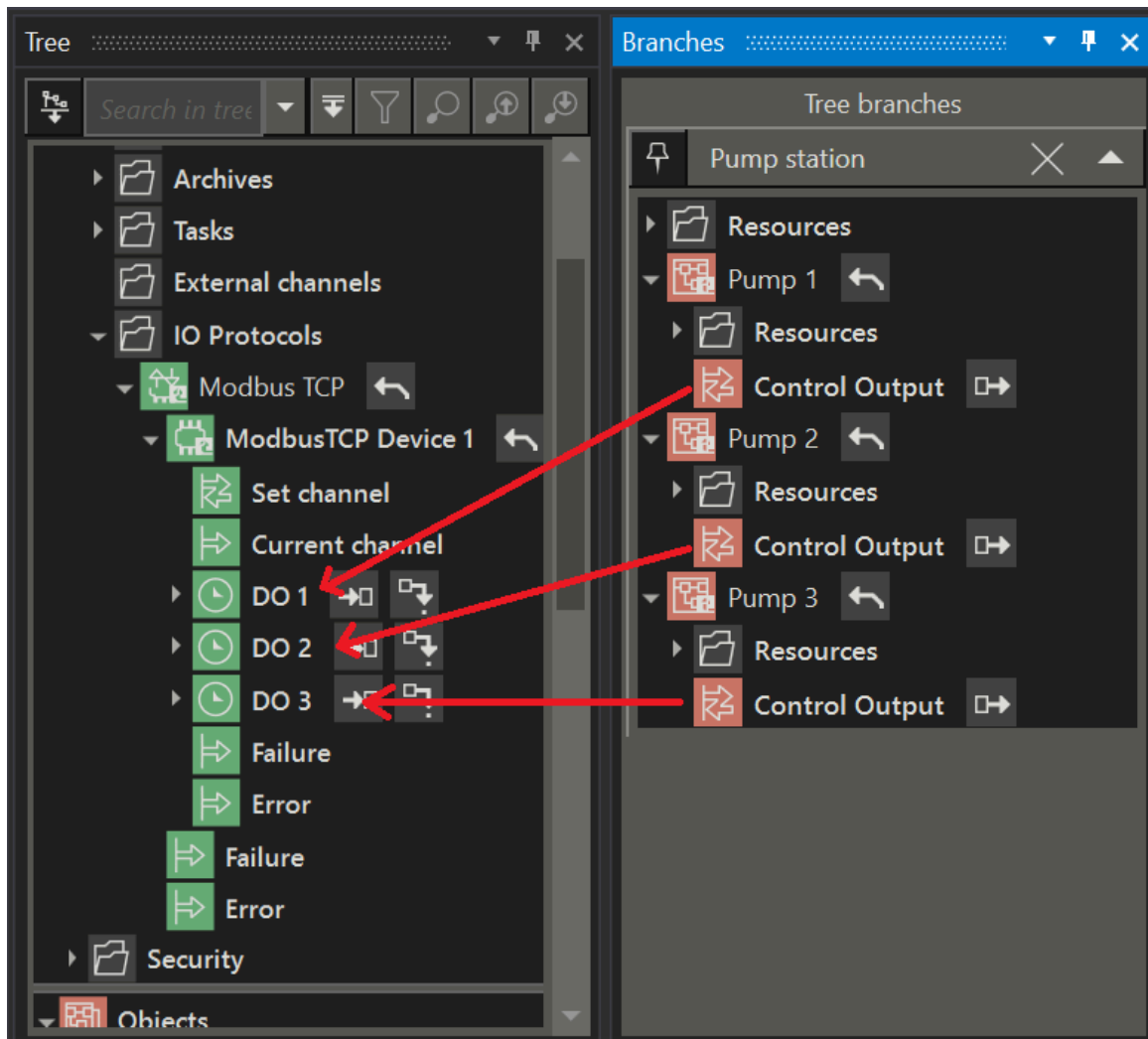


Figure 92

At this point, our pump station is ready!

QS. Lesson 3. Launch

Start **runtime mode**. In the browser you will see the pump station we created with three pumps, each controlled independently of the others. Pumps can be turned on or off in any sequence. You can also close unnecessary windows and then open them again, and so on.

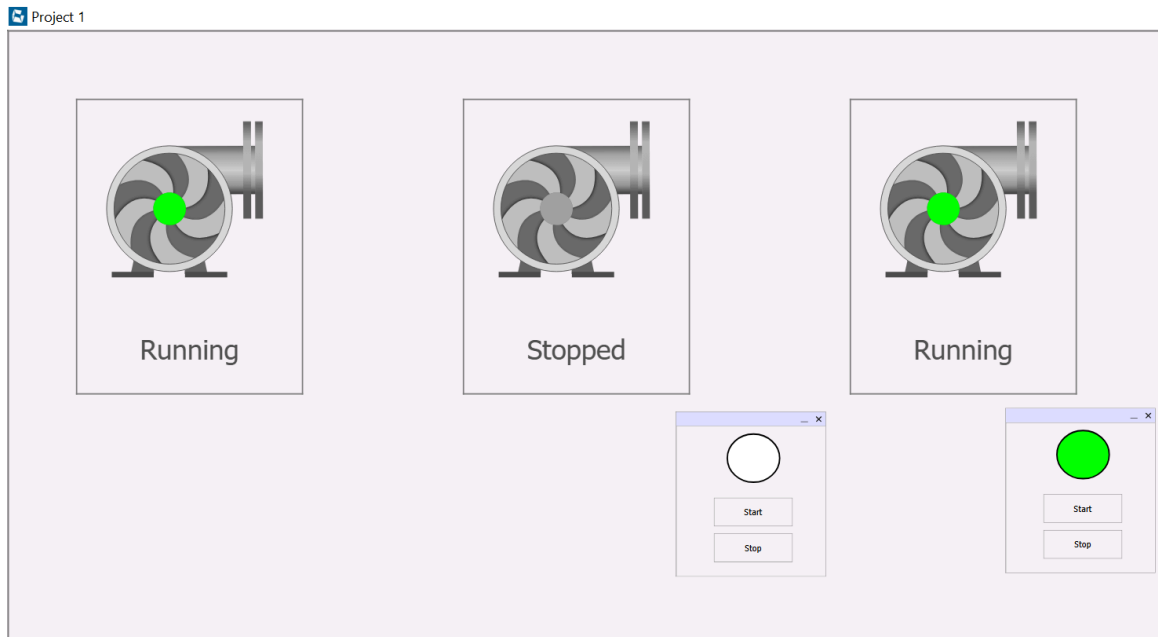


Figure 93

QS. Lesson 3. Library operations

While creating this project, we placed the replicable **Pump** item in the **Local** library. The **Local** library is available only in the project where it was created. If the library item needs to be available for use in other projects as well, it must be created in another library — an existing one or a new one. Right-click the **Libraries** item and select **Add - Library**:

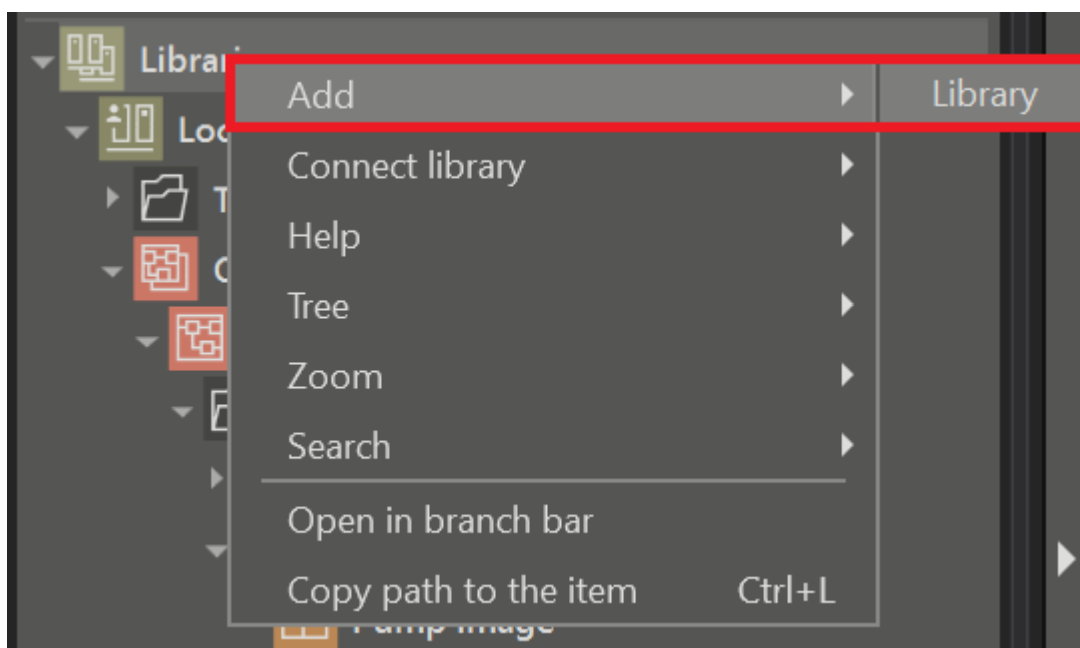


Figure 94

Name this library **My pumps**. Copy the library pump we created from the **Local** library and paste it into the **My pumps** library:

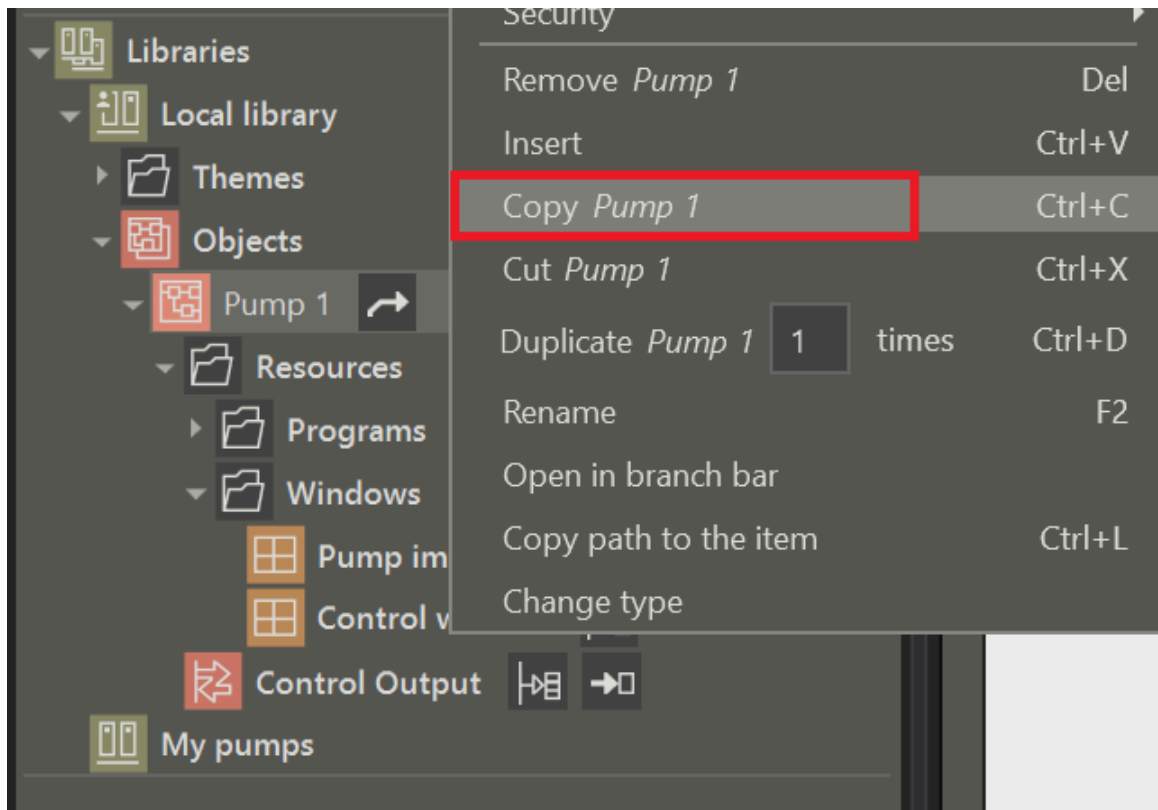


Figure 95

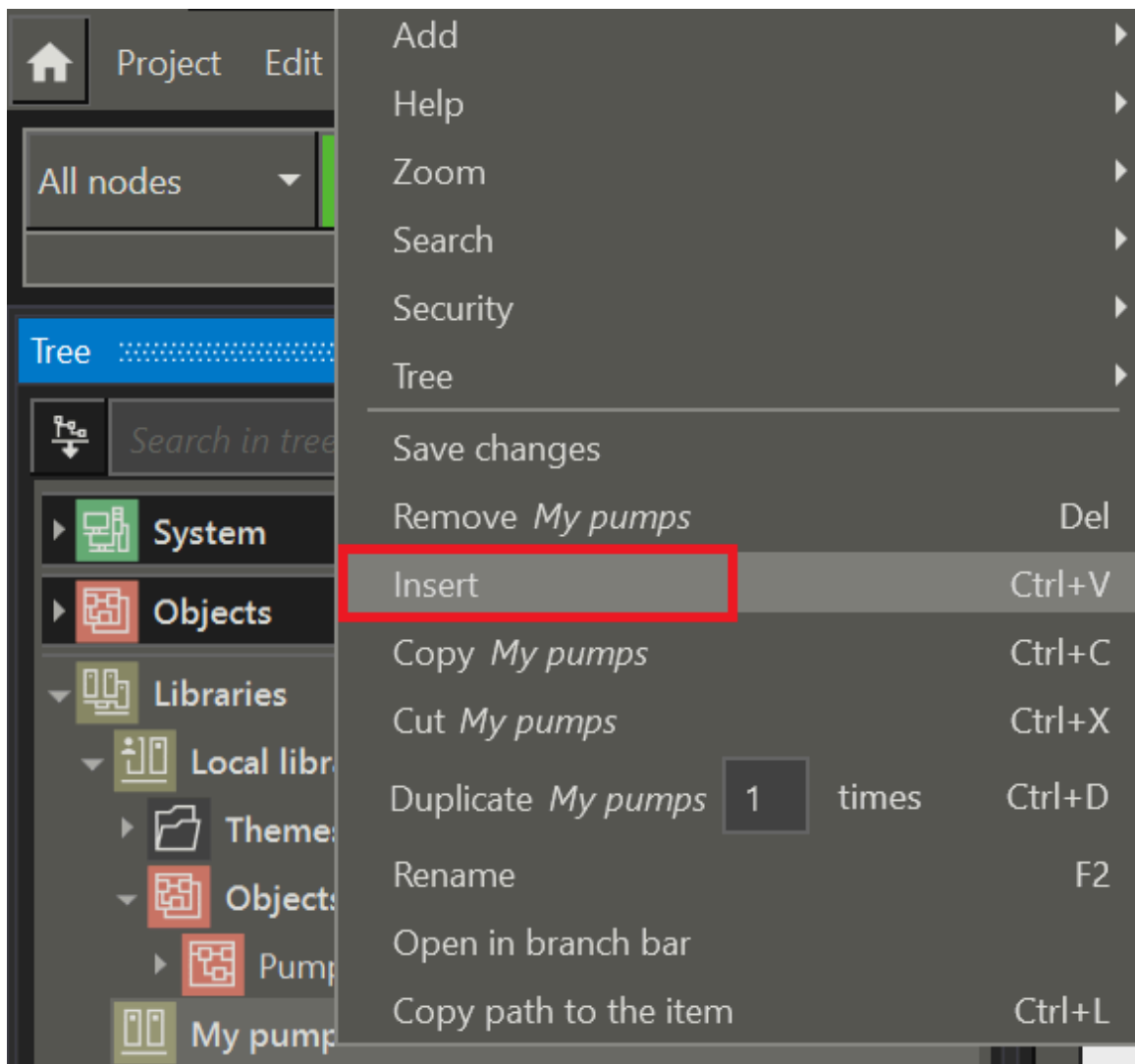


Figure 96

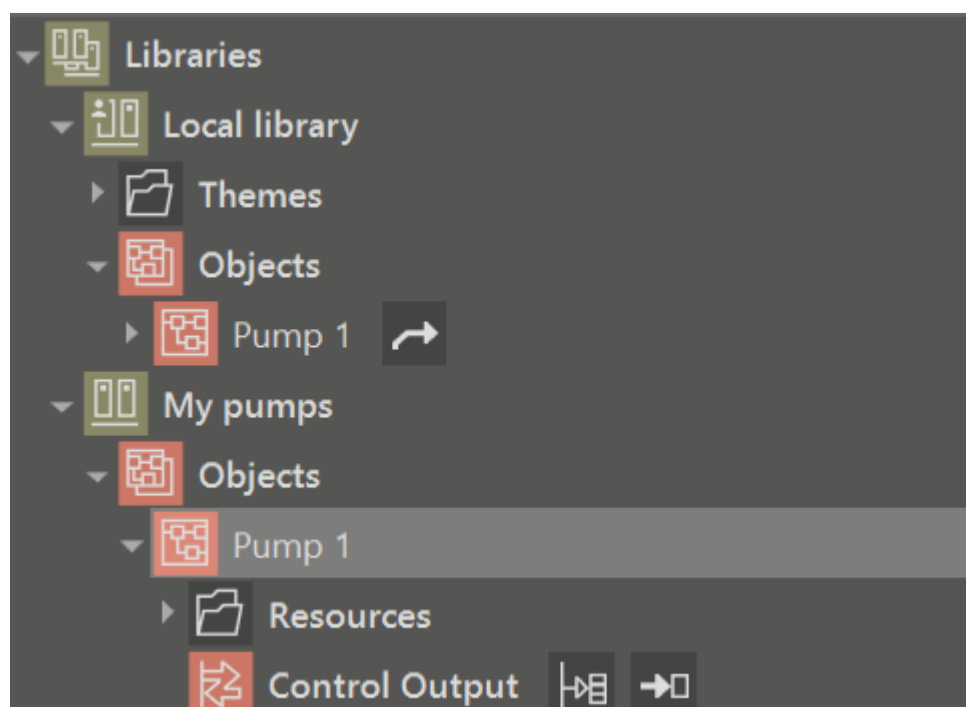


Figure 97

Then right-click the library name and select the context menu item **Save changes**:

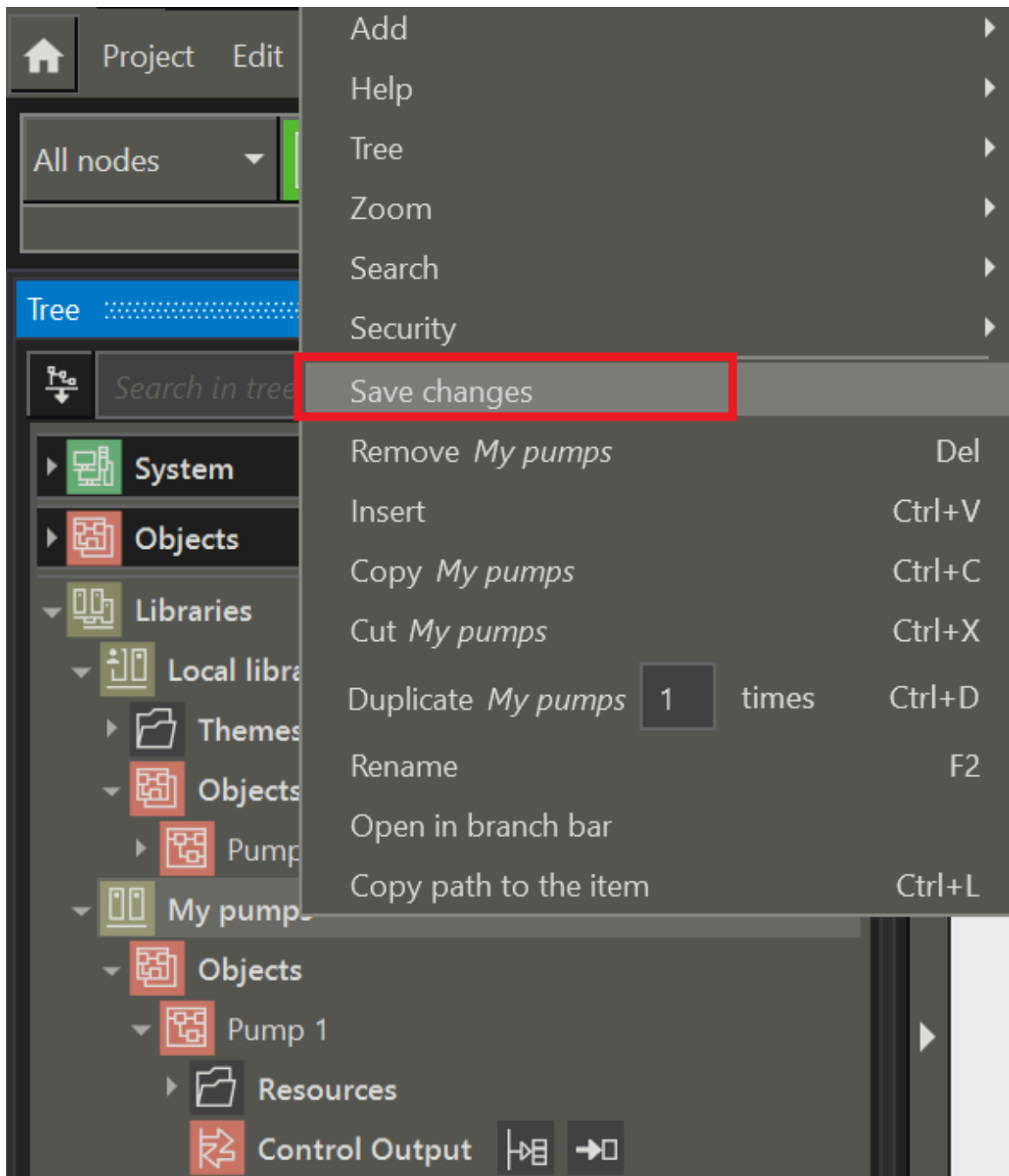


Figure 98 – Then right-click the library name and select the context menu item **Save changes**

After that, the library will be saved to disk and will be available for connection from other projects via the **Connect library** context menu. At the moment it is not available for connection because it is already connected. To demonstrate this, delete the **My pumps** library from the project via the **Delete** context menu:

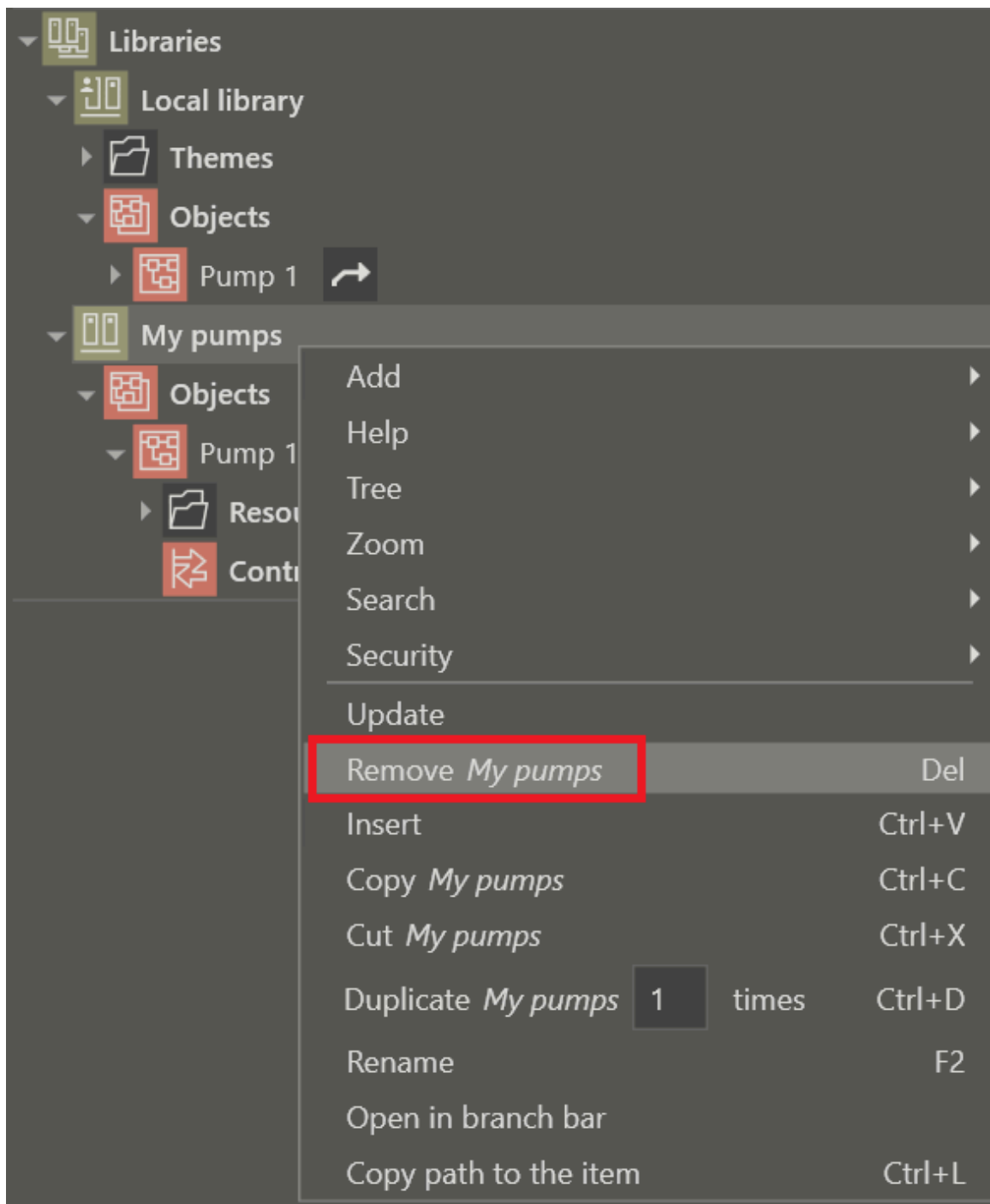


Figure 99

Now connect it to the project: Right-click, select **Connect library**, and from the list of available libraries select **My pumps**:

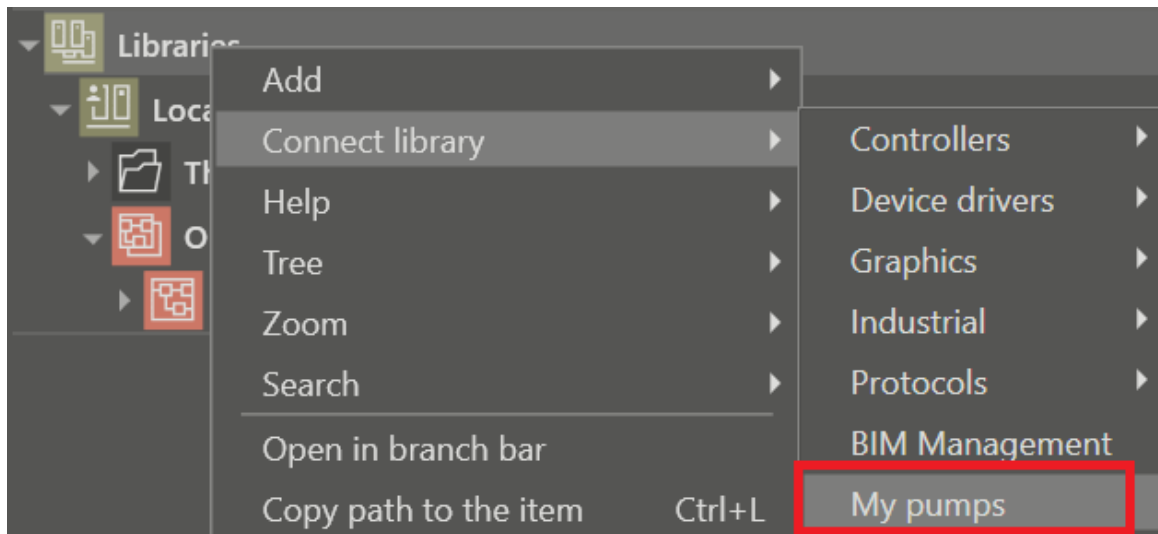


Figure 100

The library has been connected and can now be used.

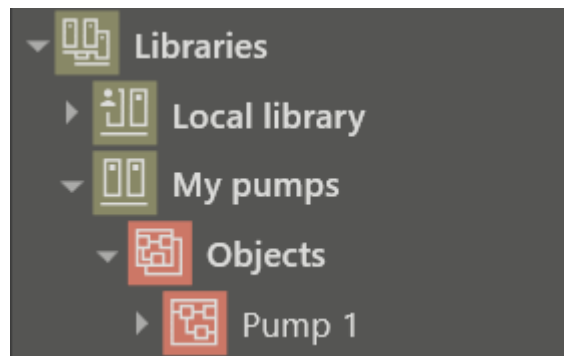


Figure 101