EcoStruxure™
Innovation At Every Level

Knowledge Base

EcoStruxure Automation Expert: Virtual Hardware Module for quick testing

Eric Coudurier

Rev 1.0
13 August 2026

Public

Disclaimer

⚠ CAUTION

This technical document is NOT a training or instruction tool or a substitute for hands-on training for the installation, use or maintenance of Schneider Electric equipment. It is provided merely as a demonstration of the installation and configuration of our products, and may ONLY be used as a supplemental reference by a competent service technician who is fully trained in the care, operation and maintenance of the installed electrical equipment. Although reasonable care has been taken to provide accurate and authoritative information, Schneider Electric assumes no responsibility for any consequences arising out of the use of this presentation.

Schneider Electric cautions you that electrical equipment should be installed, serviced or maintained ONLY by qualified electrical maintenance personnel. Remember that all sources of electrical power must be disconnected prior to visual inspection, tests, maintenance or installation of any electrical equipment. Failure to do so can result in serious personal injury, property damage, OR death.

© 2024 Schneider Electric. All rights reserved.

Trademarks

Schneider Electric has made every effort to supply trademark information about company names, products and services mentioned in this manual. Trademarks shown below were derived from various sources.

EcoStruxure Automation Expert is a trademark owned by Schneider Electric.

General Notice:

Some product names used in this manual are used for identification purposes only and may be trademarks of their respective companies.

Public

Contents

1 INTRODUCTION 4
2 SIMULATION HW CAT 5
3 THE FUNCTION BLOCKS 6
3.1 Bus Master 6
3.2 Digital input 7
3.3 Digital Output 8
3.4 Analog Input 9
3.5 Analog Output 10
4 USAGE EXAMPLE 11
4.1 Simple simulation 11
4.2 Advanced simulation 15
5 CONCLUSION 26

Public

1 INTRODUCTION

Objective

Validate the behavior of a function block connected to I/O (inputs/outputs) without physical hardware, avoiding the limitations of direct *forcing* of variables and ad-hoc simulation functions.

1. **Proposed Approach**

Implement virtual input/output modules as HW CATs (Hardware Composite Automation Types). These HW CATs act as a software simulation interface between the function block and the HMI:

- **On the HMI side:** the operator toggles/observes I/O via a dedicated panel.
- **On the logic side:** the HW CAT processes simulated I/O as a physical device would, with no changes to the application code.

Key Benefits

2. **No additional application logic**

All simulation handling (state publishing, Commands, HMI animation) is encapsulated inside the HW CAT. The application remains unchanged—no "simulation mode" branches or FB overloads.

3. **Portable and robust testing**

Eliminates test symbolic links (symlinks) based on instance names—which are fragile (case-sensitive) and project-specific.

- HW CATs expose a stable interface bound to simulated I/O; the test logic becomes reusable across projects.
- Reduces mapping mistakes and risks associated with name-based linking.

4. **More realistic test experience**

Virtual I/O behaves like hardware, which makes it possible to validate:

- in simple simulation, the responsiveness of functional blocks to I/O signals,
- in advanced simulation, the dynamic behavior of the system (feedback, level, flow rate, speed, etc.). The simulation logic can be integrated into the HW CAT.
- the usability of the HMI (status display, controls, alarms).

The content of this document is based on EcoStruxure Automation Expert V25.0 Update 1, however, the archived files contain version V26.0.

Public

2 SIMULATION HW CAT

A master bus Simulation is created, to be able to add the virtual modules to the Hardware Configuration.

A dedicated HW CAT has been developed for each I/O type:

- Discrete Input (digital input).
- Discrete Output (digital output).
- Analog Input.
- Analog Output.

All HW CAT logic have the same structure:

Example for the Digital output simulation:

(1) SIM_BUS bus connection.

Adapter **BusIn** and **BusOut** are used to interconnect the modules.

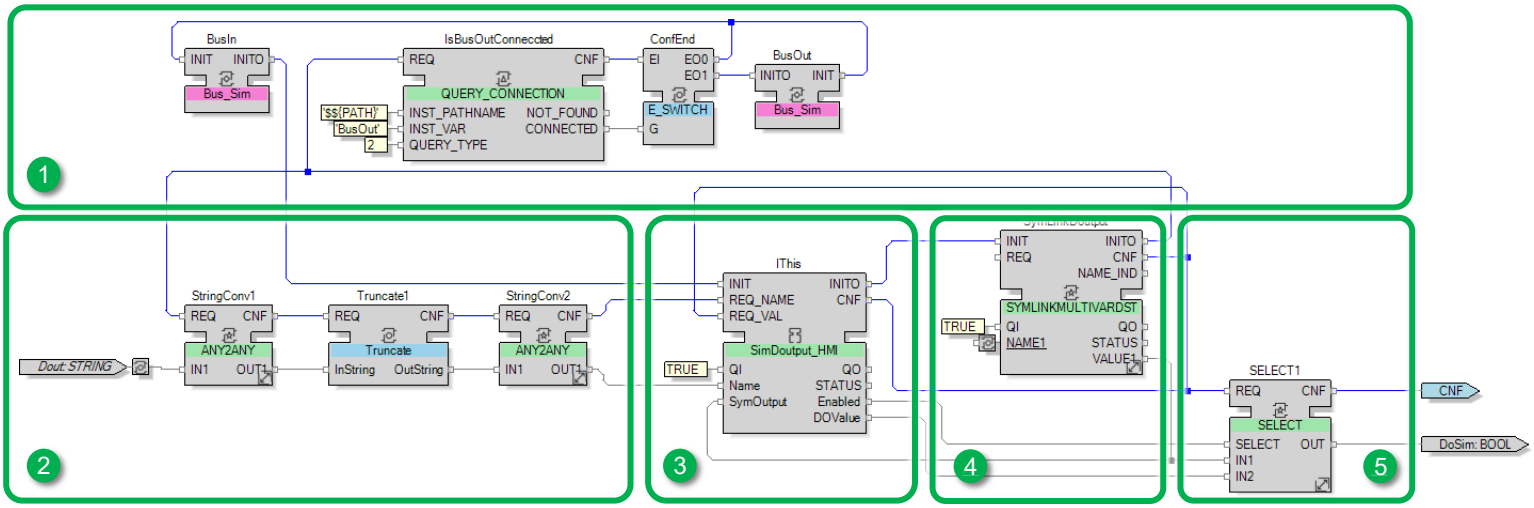
The **Query Connection** mechanism is used to detect whether another HW CAT is connected to the BusOut. If no connection is detected, the HW CAT sends an **INITO event** to the bus master to indicate that configuration is complete.

(2) Name management: Shorten the name eliminating the prefix **RES0** and sufix **Din**, **Dout**...

(3) **HMI** connection.

(4) **SYMLINK** for simulation with forcing option.

(5) Feedback for animation, with the possibility of using the output state in the logic (VTQO00) for advanced simulation, e.g. to create feedback by connecting the output to an input (motor running feedback or valve open or close state).



These HW CATs don't use the process image, they simply create a symbolic link that can be mapped to the logic and use the **HMI** to monitor inputs and control outputs.

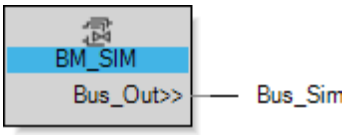
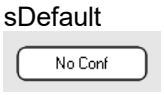
Public

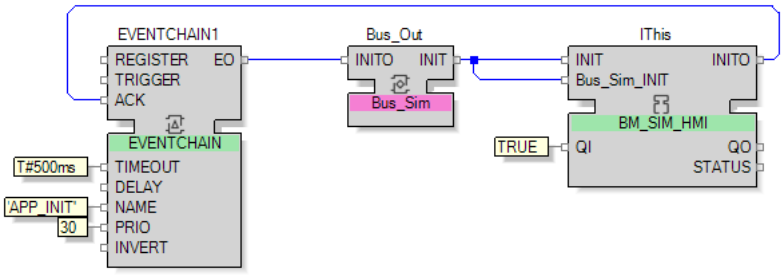
3 THE FUNCTION BLOCKS

Here are the four types of CAT HW used for simulation:

3.1 Bus Master

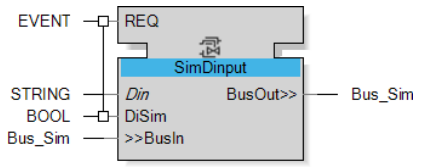
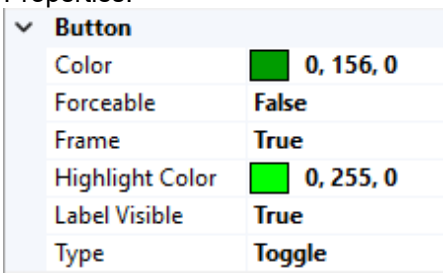
A simulation bus master is created to ensure that the simulation remains completely independent of the actual I/O bus masters. SE libraries are not required, but an initialization is needed with the name **APP_INIT**. A **DPAC_FULLINIT** block can be used to perform this initialization.
A symbol indicates whether the configuration is correct, meaning that all connections between the simulation hardware CATs are properly established.

Type	Symbol
	



Public

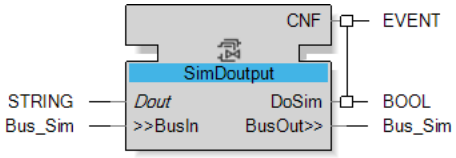
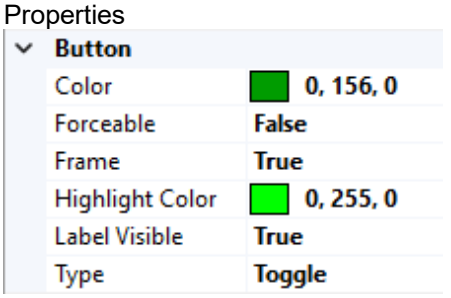
3.2 Digital input

Type	Symbol
	sDefault Forceable = False Forceable = True Properties: 

The button can be used as a **toggle switch** or **momentary switch**, **On** and **Off** colors can be changed.
The force function is used for advanced simulation to substitute the **DiSim** BOOL input.
In forced mode the frame shows the real value of the input set by the simulation logic.

Public

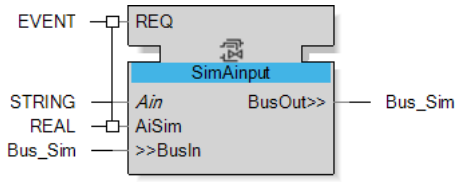
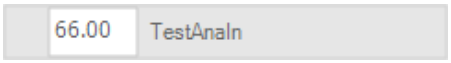
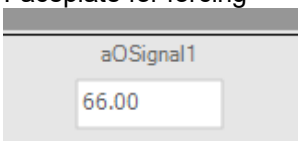
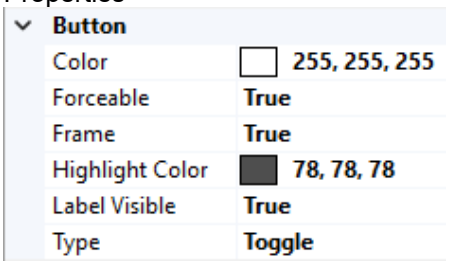
3.3 Digital Output

Type	Symbol
	sDefault Forceable = False Forceable = True Properties 

The output is display as an **On** / **Off** indicator.
For advanced simulation, Output can be forced. In forced mode the frame shows the real value of the output set by the logic.

Public

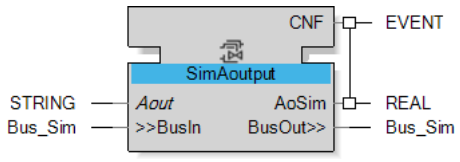
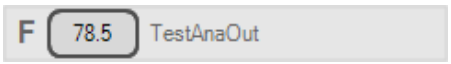
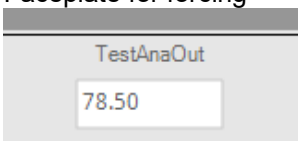
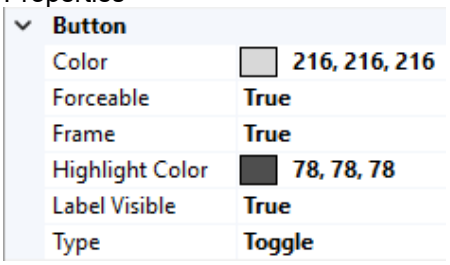
3.4 Analog Input

Type	Symbol
	sDefault Forceable = False  Forceable = True Faceplate for forcing  Properties 

In simple simulation mode (forceable = false), the displayed field allows you to enter the value.
In advanced simulation mode (forceable = true), Clicking the **F** opens the faceplate that allows you to force the value. The current value remains displayed in the rounded rectangle.
To deactivate the **Forced** value, click **F** again. The forced value remains active as long as the faceplate is open. The **F** indicator is displayed in blue while forcing is active.

Public

3.5 Analog Output

Type	Symbol
	sDefault Forceable = False Forceable = True  Faceplate for forcing  Properties 

In simple simulation mode (forceable = false), the value is displayed in the grey rounded rectangle.
In advanced simulation mode (forceable = true), Clicking the **F** opens the faceplate that allows you to force the value. The current value remains displayed in the rounded rectangle.
To deactivate the “Forced” value, click the **F** again.

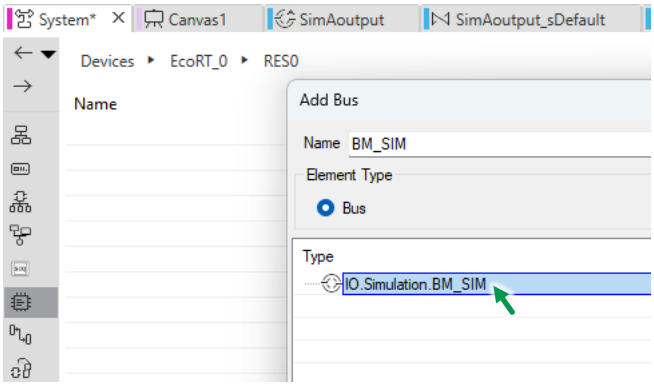
Public

4 USAGE EXAMPLE

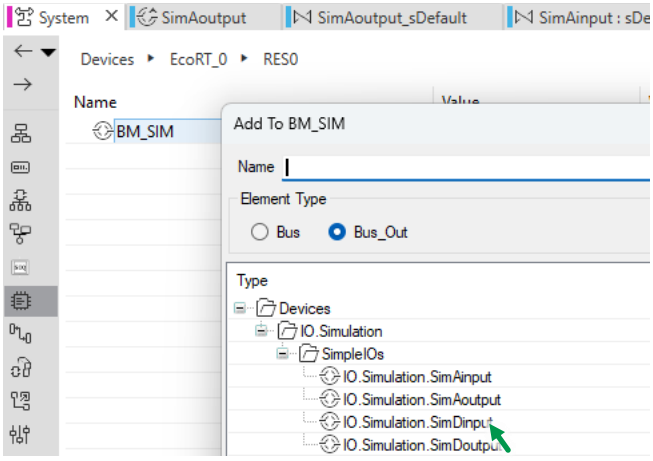
4.1 Simple simulation.

1) The HW CAT can be used directly in the Hardware configuration.
The library must be referenced.

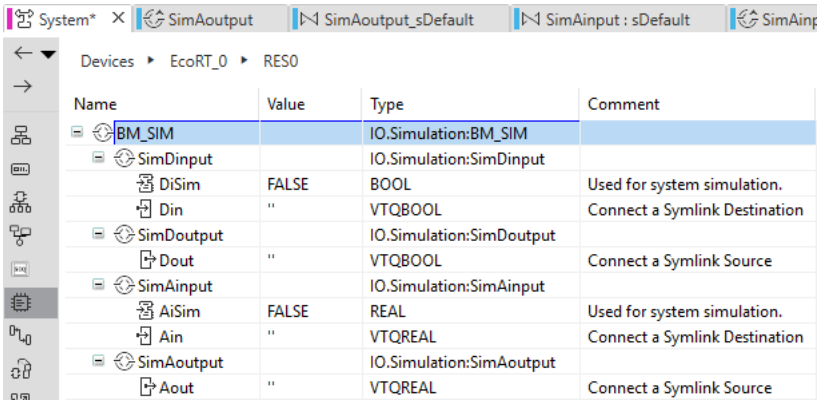
- Add the Bus Master.



- Select the bus master and add the desired Inputs and outputs



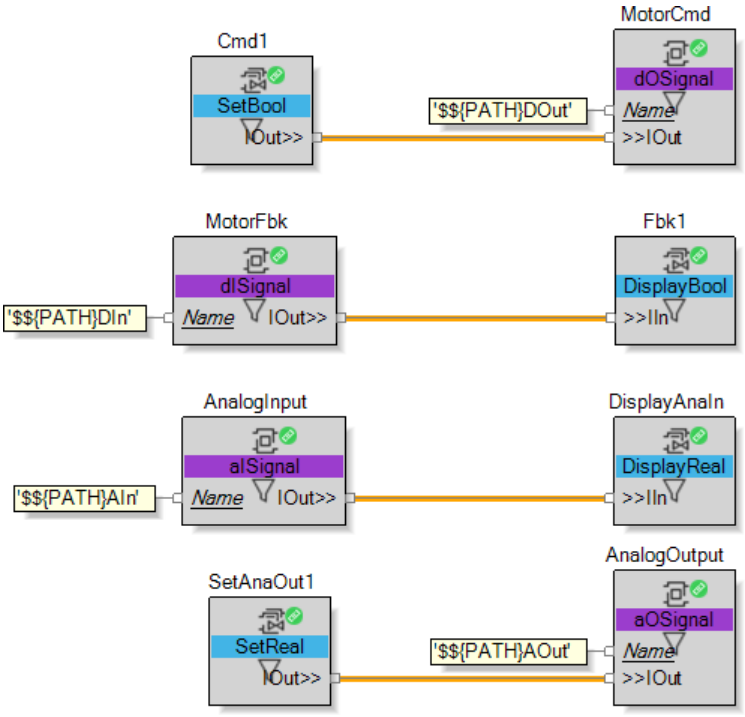
- Expand the list.



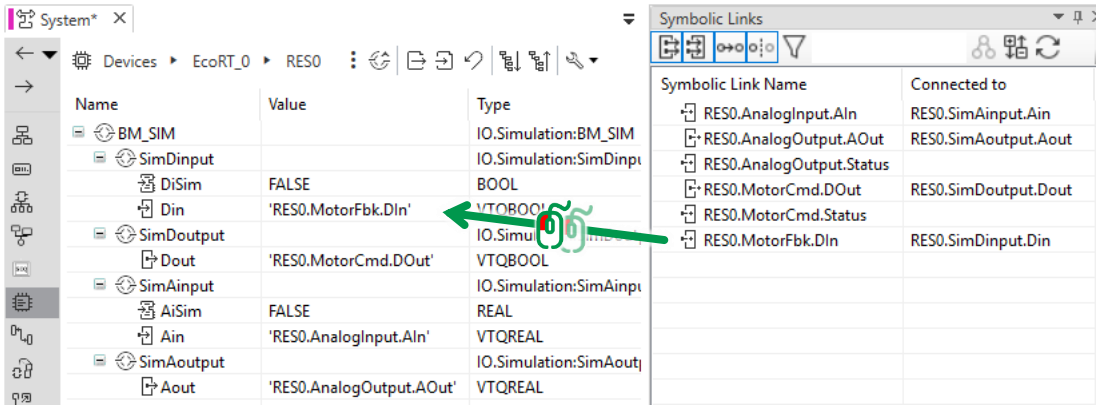
- Mapping can be done on the VTQ values.

Public

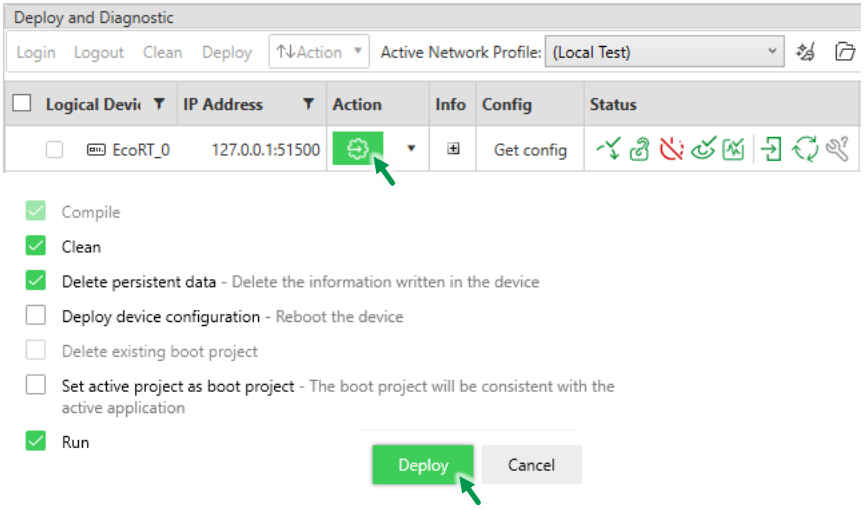
- Reference the **SE.App2Base** library, create a simple logic as shown.
  And map the Fbs to the resourse.



- Map the Signals on the hardware chanel.



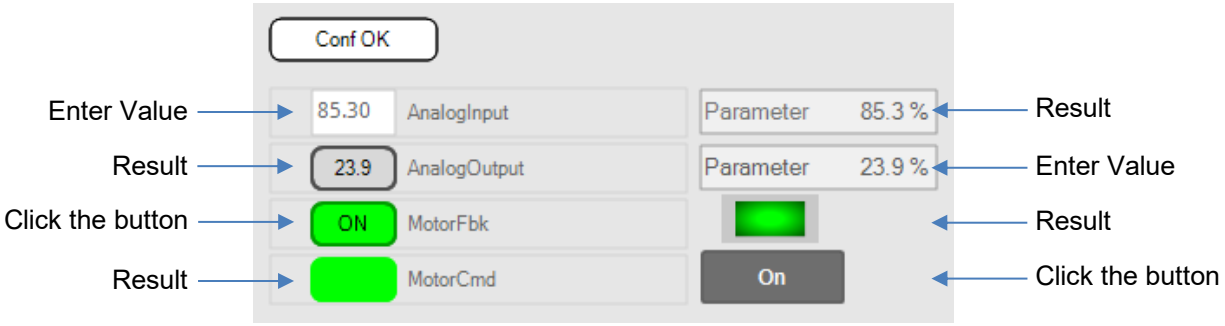
- Build and deploy the solution



Public

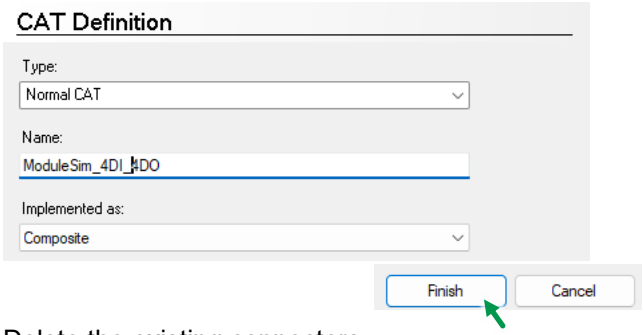
- Create a resolution, add a canvas and instantiate the symbols.

- Start Local HMI.

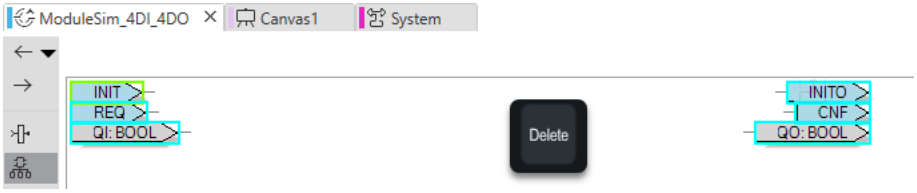


2) Multiple **SimpleIOs** can be grouped to create several channels in a single CAT.

Create a new HWCAT in your solution

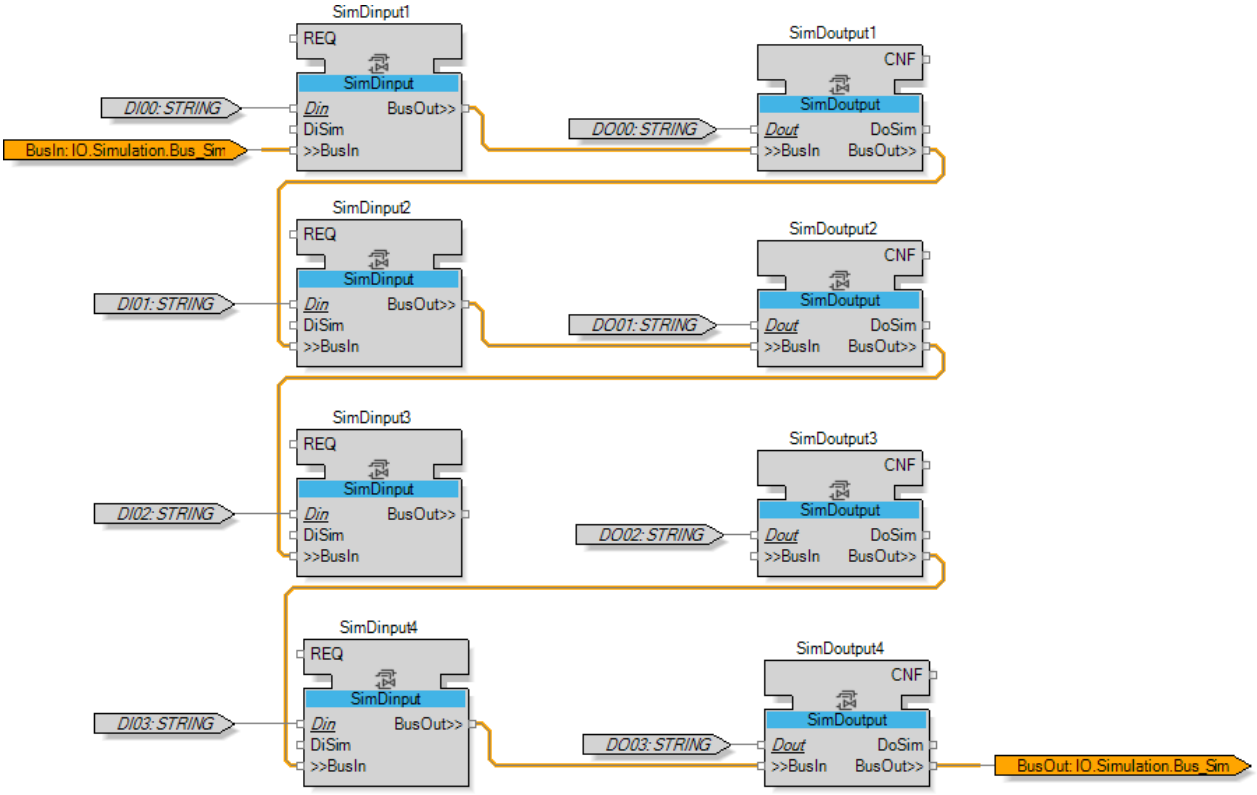


- Delete the existing connectors.



Public

- Drag & Drop the needed IOs. Here 4 Digital Inputs and 4 Digital Outputs, and connect as shown.



- Save and add the module in the configuration.

Name	Value	Type	Comment
BM_SIM		IO.Simulation:BM_SIM	
ModuleSim_4DI_4DO_1		Main:ModuleSim_4DI_4DO	
DI00	''	VTQBOOL	Connect a Symlink Destination
DI01	''	VTQBOOL	Used for system simulation.
DI02	''	VTQBOOL	Connect a Symlink Destination
DI03	''	VTQBOOL	Connect a Symlink Destination
DO00	''	VTQBOOL	Connect a Symlink Source
DO01	''	VTQBOOL	Connect a Symlink Source
DO02	''	VTQBOOL	Connect a Symlink Source
DO03	''	VTQBOOL	Connect a Symlink Source
SimDinput		IO.Simulation:SimDinput	
SimDoutput		IO.Simulation:SimDoutput	
SimAinput		IO.Simulation:SimAinput	
SimAoutput		IO.Simulation:SimAoutput	

The module is ready to use. As in the first example, drag and drop the symbolic links to the channel and create the corresponding symbols on the canvas.

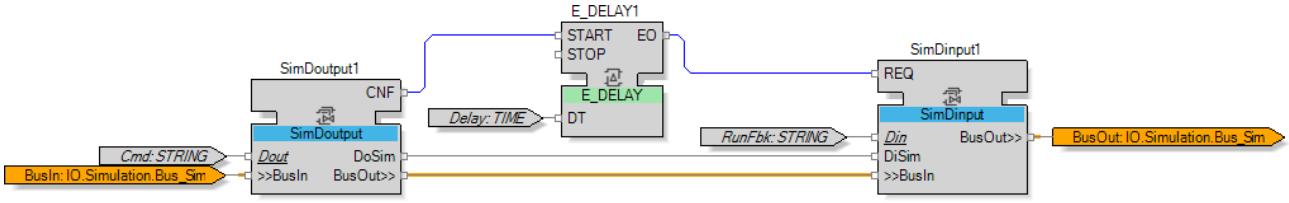
Public

4.2 Advanced simulation

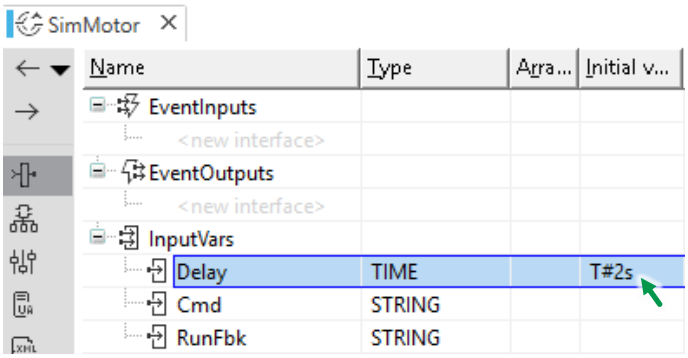
- The HW CAT can include logic to simulate the operational behavior of the system.
  A digital output can trigger an input after a configurable delay to simulate feedback signals, such as a motor running status or valve open/close position feedback.

1) Motor Simulation

- Create an HW CAT name it **SimMotor** and create the following logic.



- In the interface editor add an initial value for the delay.

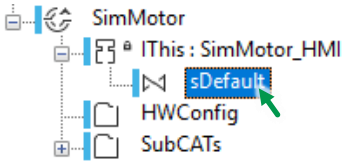


- Save the HW CAT, add it to the hardware configuration, rename it **MyMotorSim** and map it with the **Signals** symbolic links you created earlier (MotorCmd.Dout and MotorFbk.Din).

Name	Value	Type	Comment
BM_SIM		IO.Simulation:BM_SIM	
MyMotorSim		Main:SimMotor	
Delay	T#2s	TIME	Delay Time
Cmd	'RES0.MotorCmd.DOut'	VTQBOOL	Connect a Symlink Source
RunFbk	'RES0.MotorFbk.DIn'	VTQBOOL	Connect a Symlink Destination

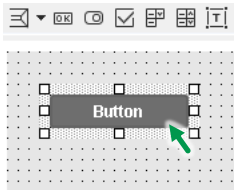
In the Hardware Configuration, **Delay** is a configurable parameter with the predefined initial value.

- Open the **sDefault** Symbol to create a **faceplate** to control the motor



Public

- Add a **Button**



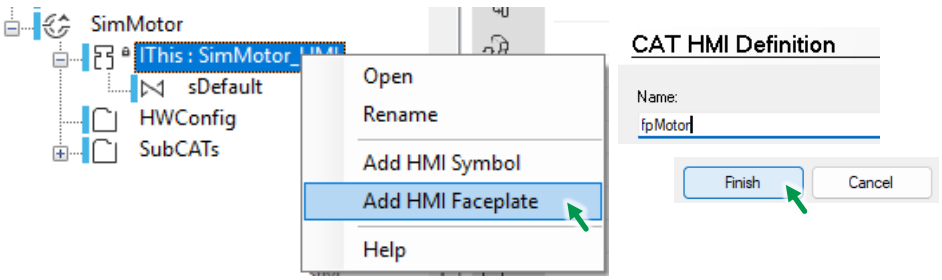
- Select the **Button**, Open the **Source Code** and add the following lines.
  The instance name of the hardware CAT created in the hardware configuration will show up on the button.

SimMotor_sDefault

```
/*
 * Created by EcoStruxure Automation Expert.
 * User:
 * Date: 7/27/2026
 * Time: 10:44 AM
 *
 */

using System;
using NxtControl.GuiFramework;
namespace HMI.Main.Symbols.SimMotor
{
    /// <summary>
    /// Description of sDefault.
    /// </summary>
    public partial class sDefault : NxtControl.GuiFramework.HMISymbol
    {
        string instname;
        public sDefault()
        {
            //
            // The InitializeComponent() call is required for Windows Forms designer support.
            //
        }
        protected override void OnInitializationCompleted()
        {
            base.OnInitializationCompleted();
            instname = FullInstancePath;
            string[] parts = instname.Split('-');
            string lastPart = parts[parts.Length - 1];
            drawnButton1.Text = lastPart;

        }
    }
}
```

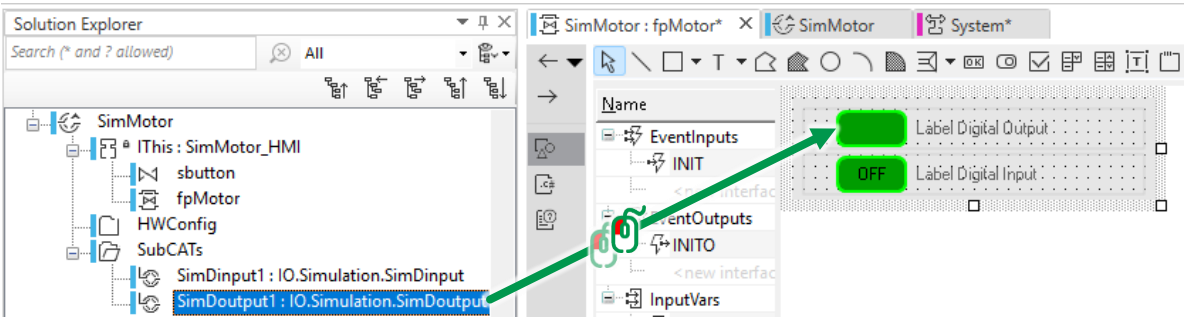
Save, then select **Verify** to see if there are any errors.

- Create the face plate.

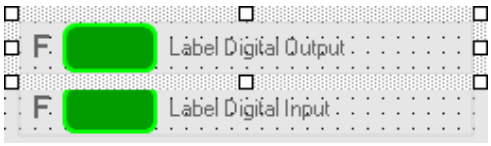


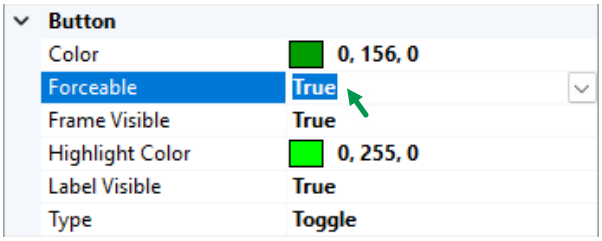
Public

- In the face plate drag and drop the 2 SubCATs **SimDinput1** and **SimDoutput1**.



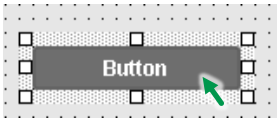
- In the properties list for each symbols select **Forceable = True**

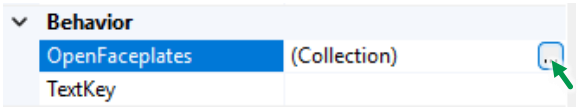




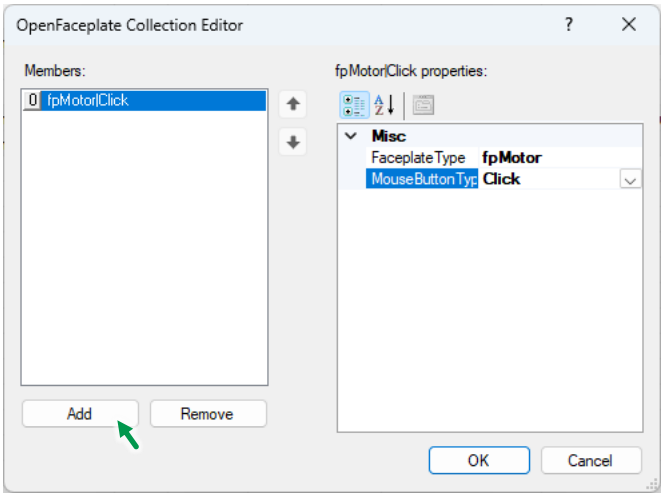
A **F** is shown on the symbol when **Forceable** is set to **True**.

- Save the **faceplate** and reopen the **sDefault**.
- Select the button, in the **Properties** / **Behavior** / **Open** the F**aceplates** Collection.





- In the editor select **Add**, then **FaceplateType: fpMotor** and **Mousebuttontype: Click**.



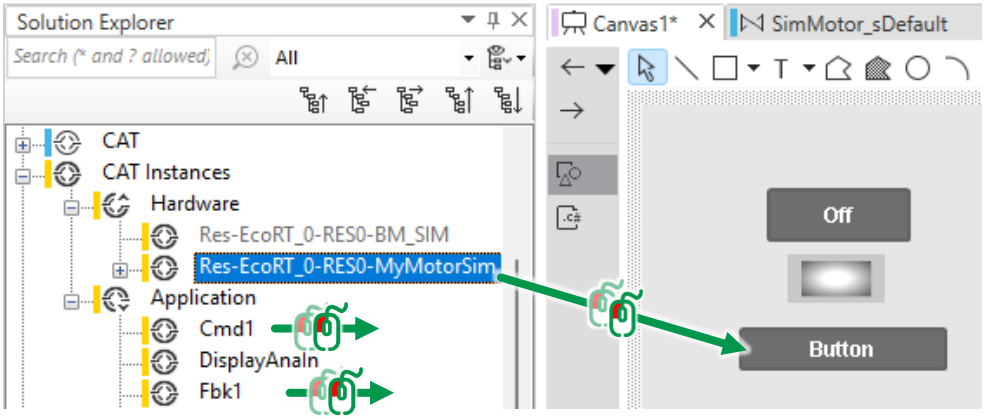
Click **OK** button to save the changes.

Public

- To display the instance name in the faceplate header, open the **Code Source** editor again and add the following lines.

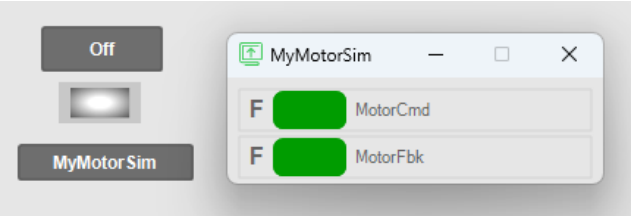
```
/*
 * Created by EcoStruxure Automation Expert.
 * User:
 * Date: 7/27/2026
 * Time: 11:25 AM
 *
 */

using System;
using NxtControl.GuiFramework;

namespace HMI.Main.Symbols.SimMotor
{
    /// <summary>
    /// Description of sbutton.
    /// </summary>
    public partial class sDefault : NxtControl.GuiFramework.HMISymbol
    {
        Main.Faceplates.SimMotor.fpMotor _fp;
        string instname;
        public sDefault()
        {
            //
            // The InitializeComponent() call is required for Windows Forms designer support.
            //
            InitializeComponent();
        }
        protected override void OnInitializationCompleted()
        {
            base.OnInitializationCompleted();
            instname = FullInstancePath;
            string[] parts = instname.Split('-');
            string lastPart = parts[parts.Length - 1];
            drawnButton1.Text = lastPart;
        }

        protected override void OnInitializeFaceplate(HMIFaceplate hmiFaceplate)
        {
            _fp = hmiFaceplate as Main.Faceplates.SimMotor.fpMotor;
            if (_fp != null)
            {
                _fp.Title = drawnButton1.Text; ;
            }
            base.OnInitializeFaceplate(hmiFaceplate);
        }
    }
}
```

- Open the Canvas, and add the symbols



Public

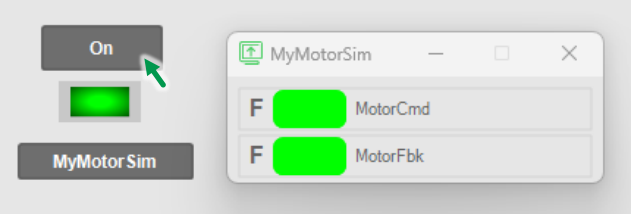
- Deploy the solution and start the HMI.



The **Button** and **Faceplate** names display the instance name used in the hardware configuration.

- Test the simulation.

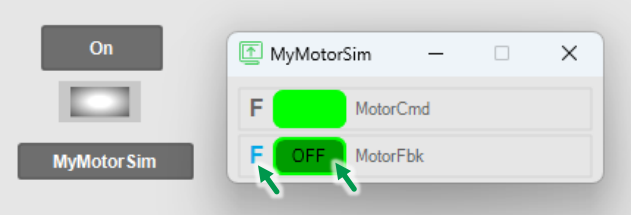
  Click on **OFF** button.



The **MotorCmd** indicator light comes on, after 2 seconds **MotorFbk** and the **Fbk1** indicator lights come on.

- Test the Forcing.

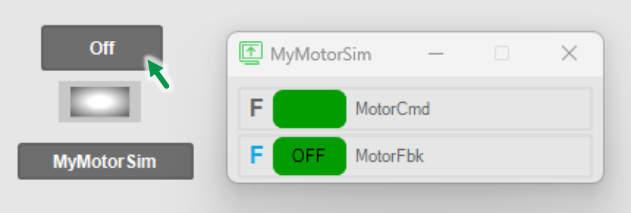
  Click on the **F** of the **MotorFbk** to force the input feedback to **OFF**.



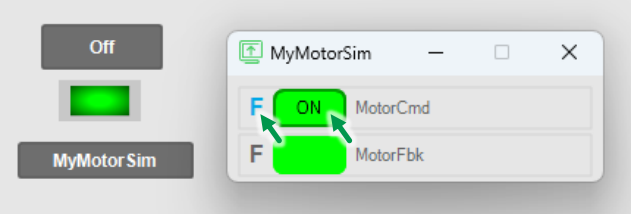
F Lights in blue to indicate forcing. **MotorFbk** turns dark green when forced to OFF.

The rounded rectangle shows the actual value of the input.

Set the **On** switch to **Off**. After 2 seconds the rounded rectangle turns to dark green.



- In the same way you can test forcing on the command.



After 2 seconds the Feedback lights turn to light green.

Forcing is used to verify anomalies and the corresponding alarms.

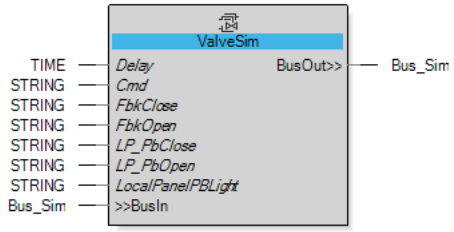
Public

2) Valve simulation

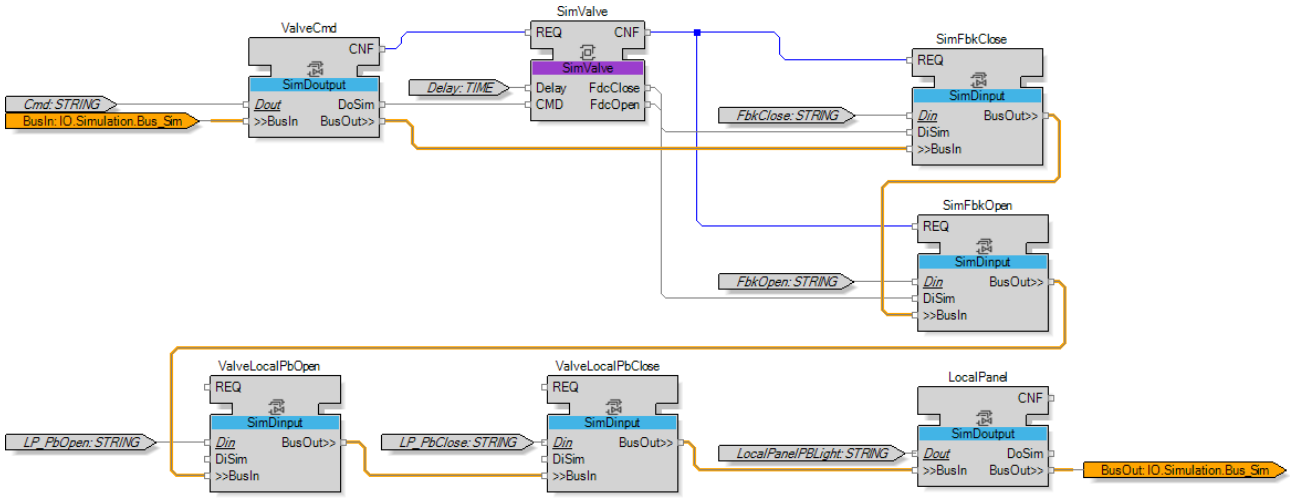
In the attached solution, a simulation valve is available. It allows you to test the following functions:

- Opening/closing in manual or programmed mode; the open and close limit switches are controlled by the simulation.
- The local control panel: the indicator light turns on when activated and allows you to control the valve using the open/close pushbuttons.

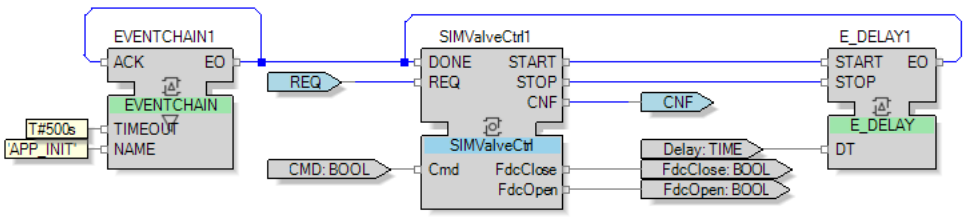
- **ValveSim** Type:



- Logic:



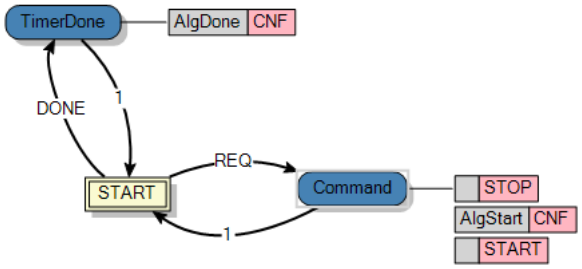
- Composite **SimValve** simulates the Open and Close feedback.
  A delay time of 2 seconds allows you to see the valve travel.



Public

- Basic **SIMValveCtrl**

Name	Type	With
EventInputs		
DONE		
REQ		Cmd
<new interface>		
EventOutputs		
START		
STOP		
CNF		FdcOpen, FdcClose
<new interface>		
InputVars		
Cmd	BOOL	
<new variable>		
OutputVars		
FdcClose	BOOL	
FdcOpen	BOOL	



- Algorithm

AlgStart | AlgDone

Name	Type	Arra...
VAR_TEMP		
<new variable>		

```
1  ALGORITHM AlgStart IN ST:
2  (*Add your comment(as per IEC 61131 - 3) here
3  Normally executed algorithm
4  *)
5  (* var_declarations *)
6  (* end_var_declarations *)
7
8  // Open Command
9  IF Cmd AND NOT FdcOpen THEN
10     FdcClose := FALSE;
11 END_IF;
12 // Close Command
13 IF NOT Cmd AND NOT FdcClose THEN
14     FdcOpen := FALSE;
15 END_IF;
16 END_ALGORITHM
```

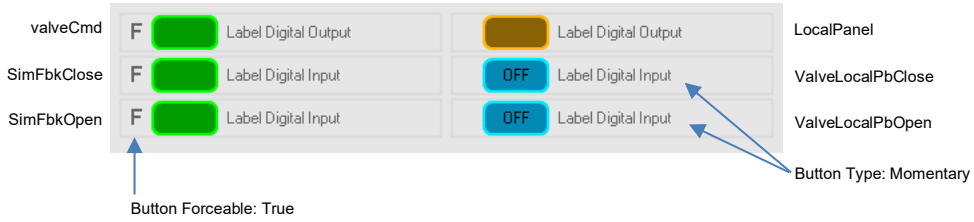
AlgStart | AlgDone

Name	Type	Arra...
VAR_TEMP		
<new variable>		

```
1  ALGORITHM AlgDone IN ST:
2  (*Add your comment(as per IEC 61131 - 3) here
3
4  *)
5  (* var_declarations *)
6  (* end_var_declarations *)
7  ;
8  FdcClose := NOT(Cmd);
9  FdcOpen := Cmd;
10 END_ALGORITHM
```

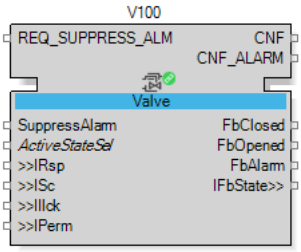
- HMI Graphic Symbol:

- Faceplate:

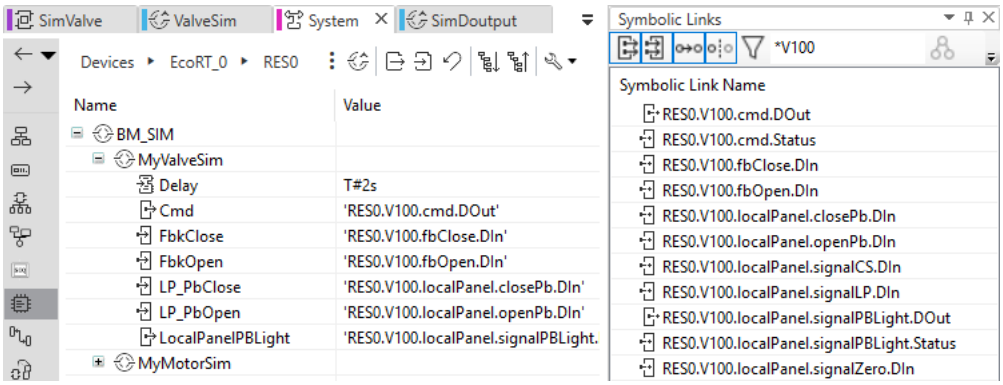


Public

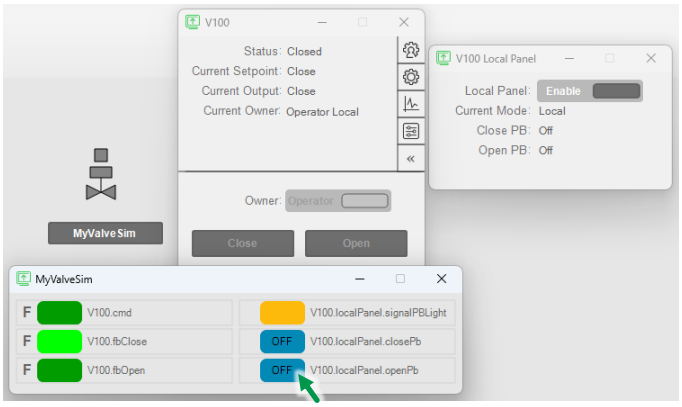
- Example of use with a SE.App2.CommonProcess **Valve**:



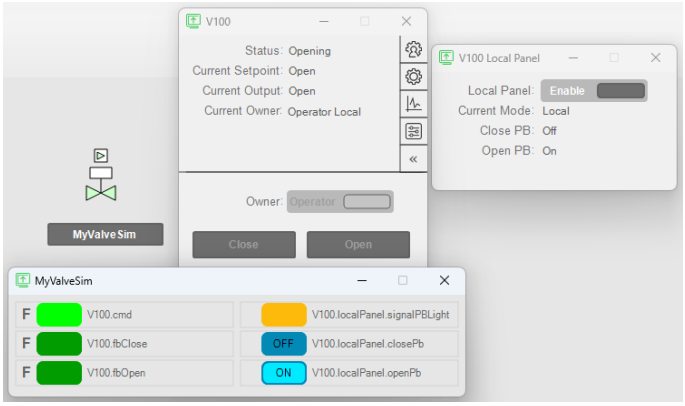
- Hardware Configuration:



- The valve is set in **Local panel** mode. Click on the **openPb** button.



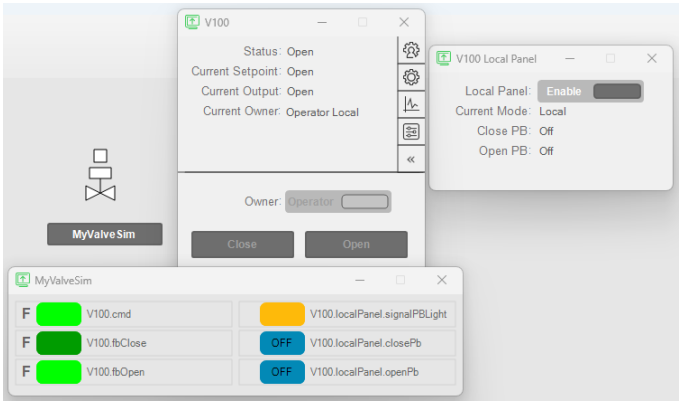
- **V100.cmd** indicator light turns **ON**, and **V100.fbClose** indicator light turns **OFF**.



The valve is moving.

Public

- After 2 seconds, **V100.fbOpen** is reached, the valve is displayed as open.

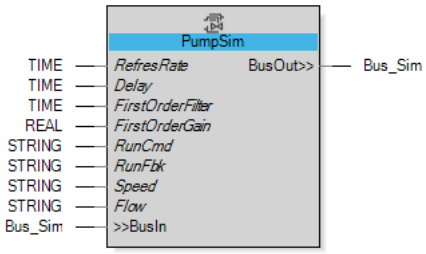


3) Pump simulation.
A MotorVs is used the following function are simulated:

- Run Command with running feedback.
- Speed setpoint with the flowmeter feedback.

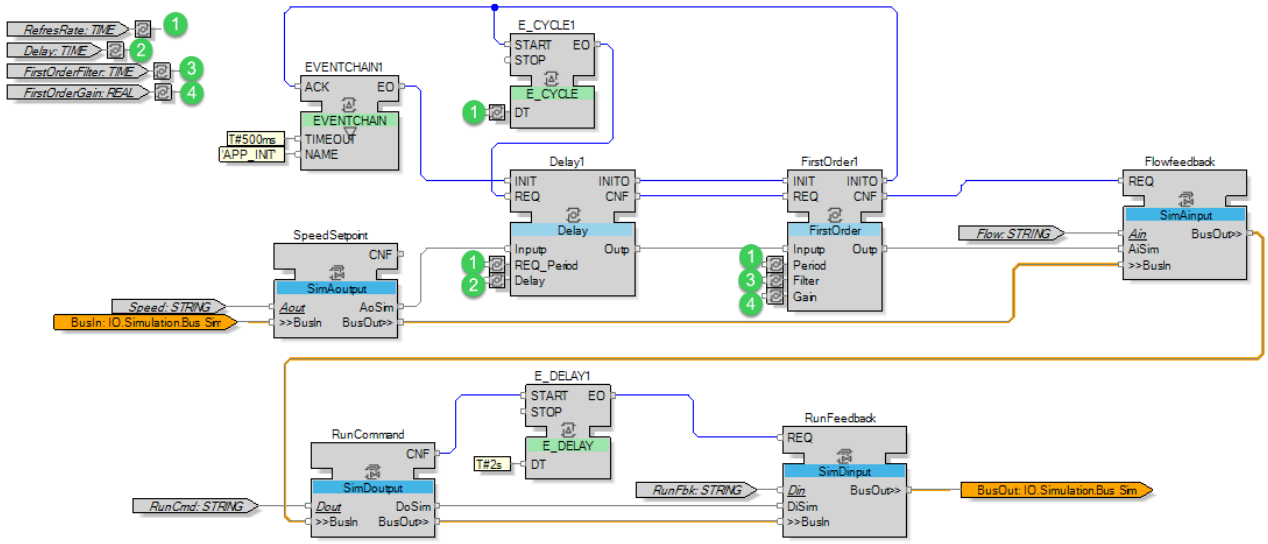
The creation of dedicated simulation symbols is not systematically required. If simulation of the operational process alone meets the project requirements, animations can be implemented using the native objects and animation capabilities provided by the chosen library.

- PumpSim Hardware CAT:

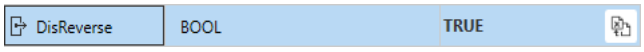


Public

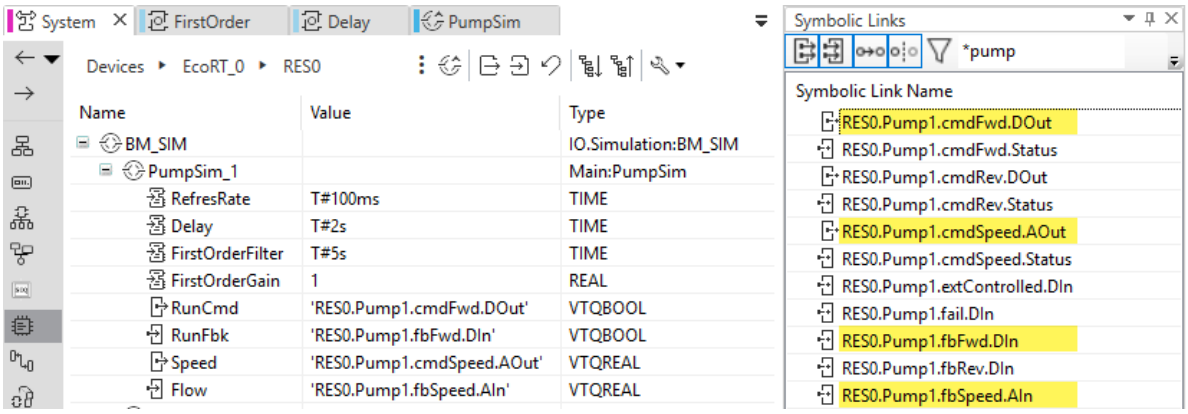
- Code:
  - The pump simulation is implemented using the motor analog output. A pure delay of 2 seconds and a first-order filter with a 5-second time constant and unity gain are applied to this signal. The filter output is then connected to the analog input, which simulates the pump flow rate.
  - The filters require cyclic execution, which is provided by an E_CYCLE function block configured with a default cycle period of 100 ms.
  - The delay, filter time constant, gain, and cycle period are fully configurable to meet the desired pump simulation behavior.



  - At the bottom of the logic diagram is the pump control; a 2-second delay is applied to the feedback cycle.
  - The **MotorVS** is configured to operate in forward mode only.



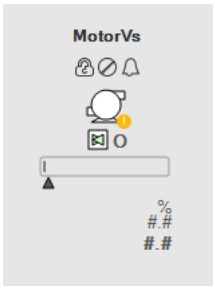
- When the **MotorVS** is instantiated and the simulation HW CAT added to the configuration, the mapping can be done:



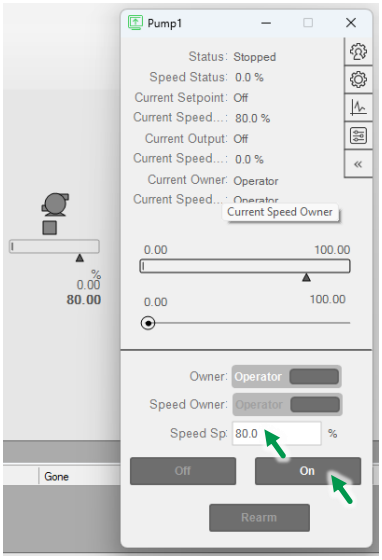
Public

The parameters are available at the top of the list.

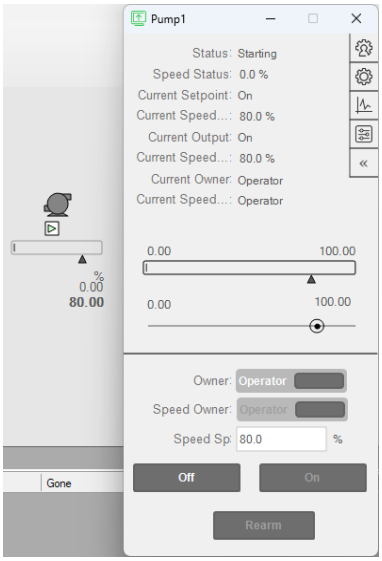
- In the canvas, simply add the **MotorVS** and use the Pump Symbol representation.



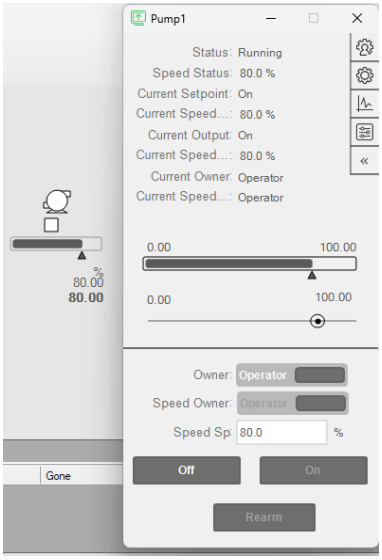
- Start the HMI:



Open the faceplate, select the speed, and turn On the Pump.



The pump is starting, waiting for the running feedback.



Once the pump starts, the flow rate gradually increases until it reaches the setpoint.

Public

5 CONCLUSION

The use of virtual modules (HW CATs) provides a simple and robust solution for testing the behavior of a system without relying on physical hardware. Based on a dedicated simulation bus and four types of virtual I/O modules (digital input, digital output, analog input, and analog output), this approach integrates seamlessly into the application. No modifications to the application code are required, and fragile symbolic links based on instance names are eliminated in favor of a stable and reusable interface that can be deployed consistently across projects.

The examples presented—simple motor simulation with forcing capabilities, valve simulation with travel-time delay, and pump simulation using a first-order filter—demonstrate the ability of this method to support both basic functional testing and more realistic dynamic behaviors, including status feedback, flow-rate, and speed simulation. In addition, the associated HMI enables validation of operator-screen ergonomics (displays, commands, and alarms) in parallel with software testing activities.

In summary, this virtual simulation approach reduces the risk of I/O mapping errors, accelerates testing and validation campaigns, and improves the portability and maintainability of EcoStruxure Automation Expert solutions, while providing a testing experience that closely reflects the behavior of real hardware.

Public