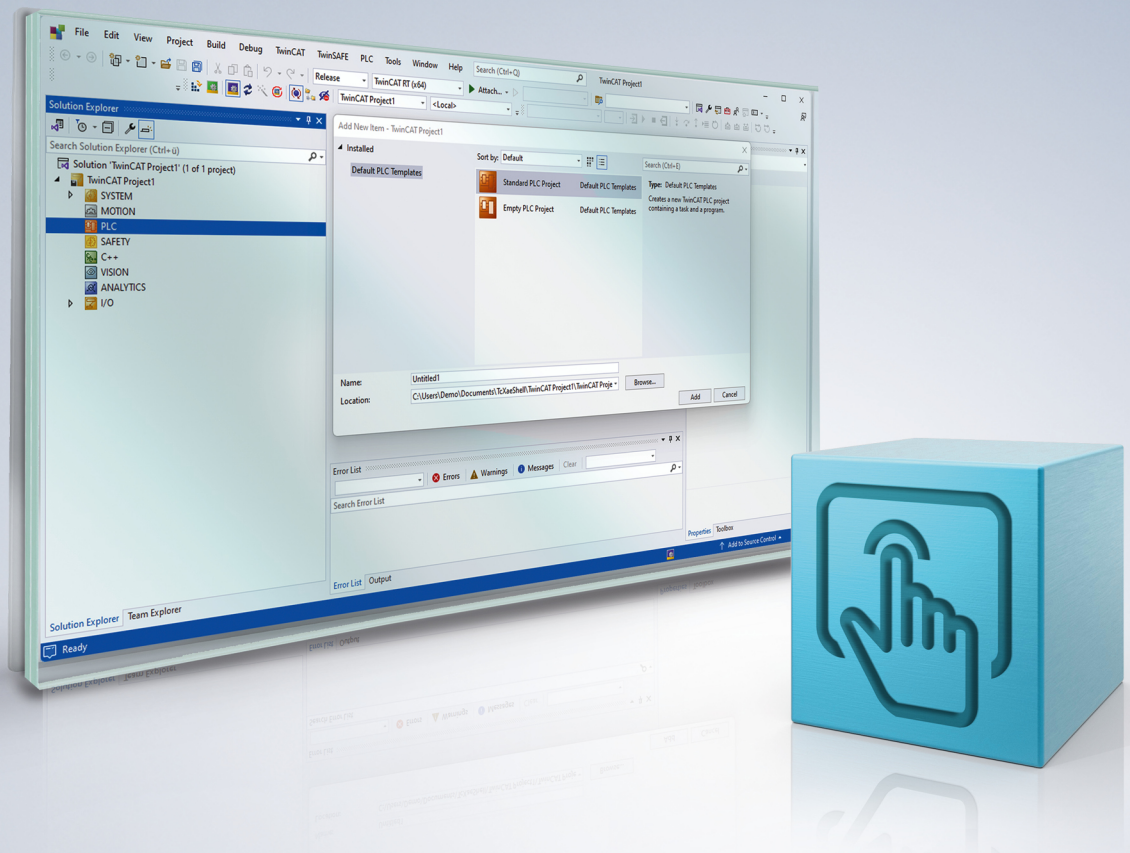
BECKHOFF New Automation Technology

Manual | EN

TwinCAT 3

HMI Process Library



2025-12-18 | Version: 1.0.0

Table of contents

1 Overview 5
1.1 Motivation 5
1.2 Scenarios 8
1.2.1 Ways of mapping PLC variables to the HMI control 8
1.2.2 Faceplate types 12
1.2.3 System architectures 14
1.3 User requirements 15
1.4 System requirements 16
1.5 Architecture 16
1.6 Business model 17
2 Installation 18
3 Quick start 19
3.1 Adding a control to the project 19
3.2 Editing control properties 19
3.3 PLC program interaction scenarios 20
3.3.1 Using the MTP PLC Library 20
3.3.2 Using PLC attribute functionality 21
3.4 Scope configuration 24
3.5 Possible behavior optimizations 25
4 Components and functionality 26
4.1 Terminology 27
4.2 ProcessObject 30
4.2.1 Properties 30
4.2.2 Functions 41
4.2.3 Events 44
4.3 Controls 44
4.3.1 General controls 44
4.3.2 Agitators 46
4.3.3 Blowers and Fans 48
4.3.4 Columns 48
4.3.5 Compressors and Vacuum Pumps 48
4.3.6 Conveyors 49
4.3.7 Drives 50
4.3.8 Filters 50
4.3.9 Heating and Cooling 51
4.3.10 Liquid Pumps 51
4.3.11 Vessels and Tanks 52
4.3.12 Valves 53
4.4 Ready-to-use faceplate elements 55
4.4.1 Operate tab 56
4.4.2 Monitor tab 59
4.4.3 Chart tab 65
4.4.4 Events tab 69

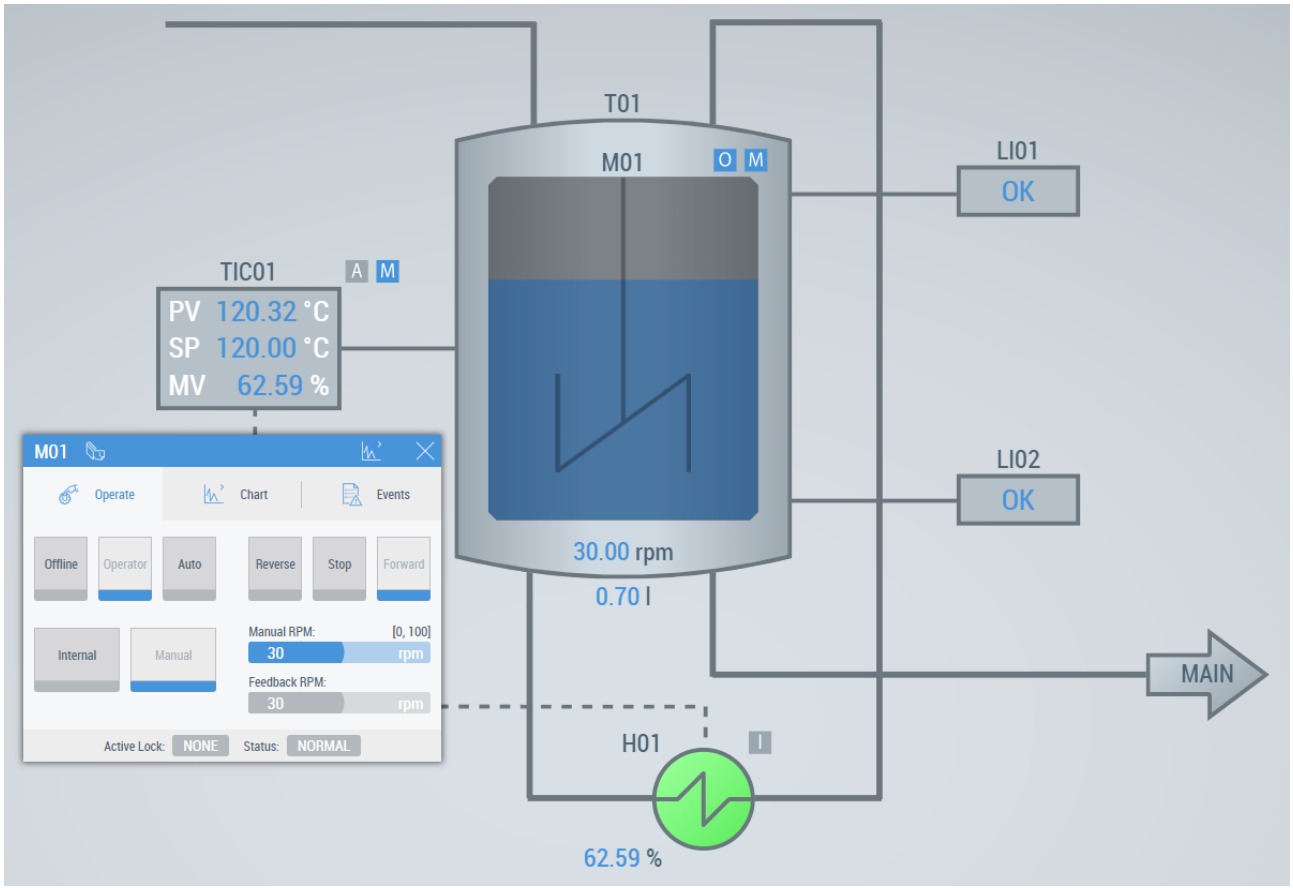
4.4.5 Interlocks tab ... 71
4.4.6 ObjectBrowser view on a Faceplate ... 73
4.4.7 Footer ... 73
4.5 Functions ... 78
4.6 Non-standard engineering units ... 81
4.7 PLC attribute functionality ... 82
4.7.1 Special placeholder description ... 83
4.7.2 Configuration PLC attributes description ... 83
4.7.3 Data PLC attributes description ... 86
5 Control appearance customization ... 88
5.1 UserControl files ... 88
5.2 Creating a new UserControl with SVG based on the existing sample ... 88
5.3 Editing the *.usercontrol.json file ... 89
5.4 Editing the *.usercontrol file ... 89
5.4.1 and <svg> attributes ... 90
5.4.2 <defs> part ... 90
5.4.3 The part with SVG elements ... 90
6 Appendix ... 92
6.1 Glossary ... 92

1 Overview

The TwinCAT HMI Process Library includes additional controls for the TwinCAT HMI (TE/TF2000), designed specifically for the process industry, but it can be used for applications in other areas as well. These controls feature clear graphical symbols that illustrate P&ID elements, complete with all essential information. The controls come with faceplates that are either pre-configured, customizable, or created by the user.

The controls can be linked to the PLC variables/object in various ways: each property can link to a separate PLC variable, or the entire PLC object (e.g. of an FB) can link to the entire control.

The TwinCAT HMI Process Library combines many existing functionalities into ready-to-use controls. For example, TwinCAT Scope charts can be displayed as well as event grids or historize charts.



1.1 Motivation

Ready-to-use controls

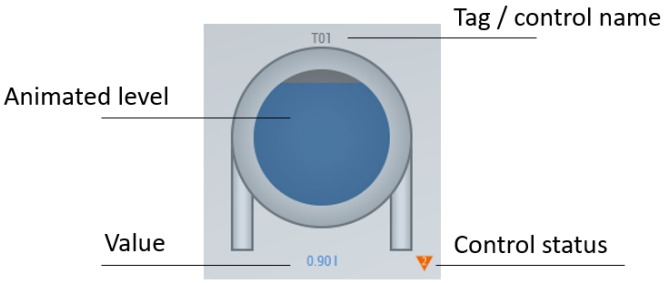
Ready-to-use HMI controls are available with the HMI Process Library. They allow users to save a lot of time when developing their own complex controls. The library includes frequently used controls based on the DIN EN ISO 10628-2 standard.

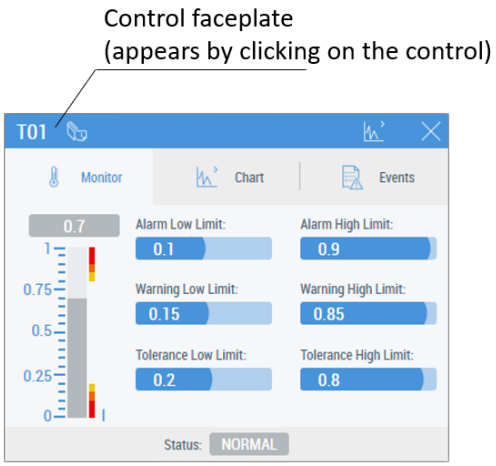
Requests for adding new controls

If any additional controls are required, please send a request to processindustry@beckhoff.com.

1:1 mapping

By mapping a dedicated PLC object, e.g. of an FB, to the control, it can automatically map each piece of information to its designated element. In addition to graphic representation, the control comes with faceplate functionality in form of a pop up.

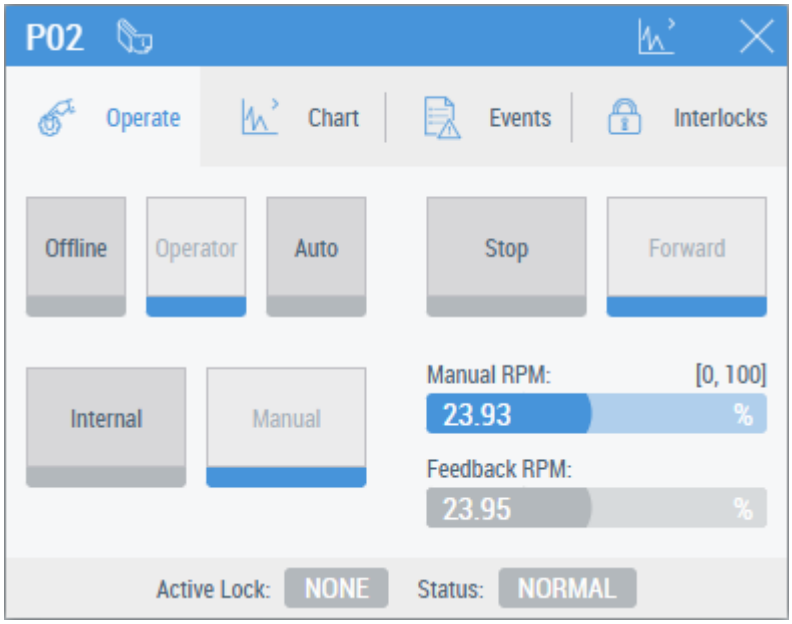


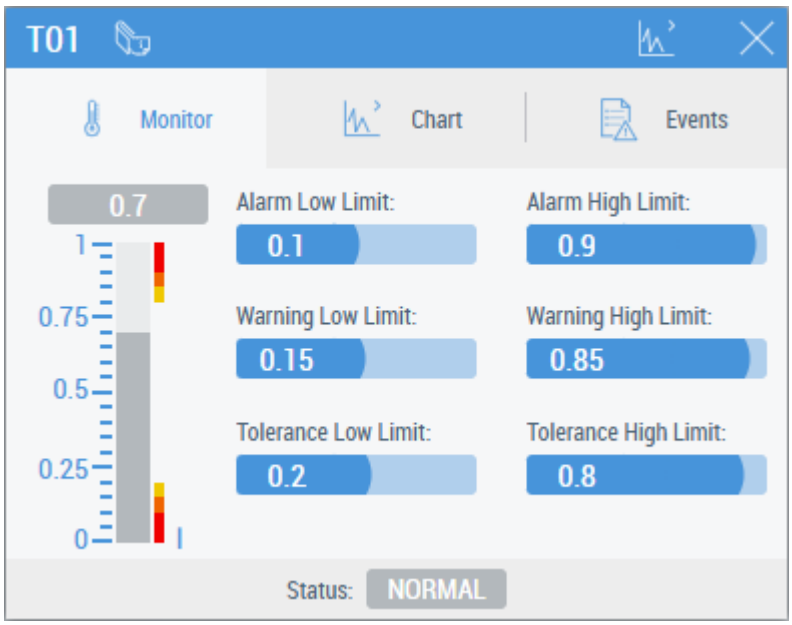


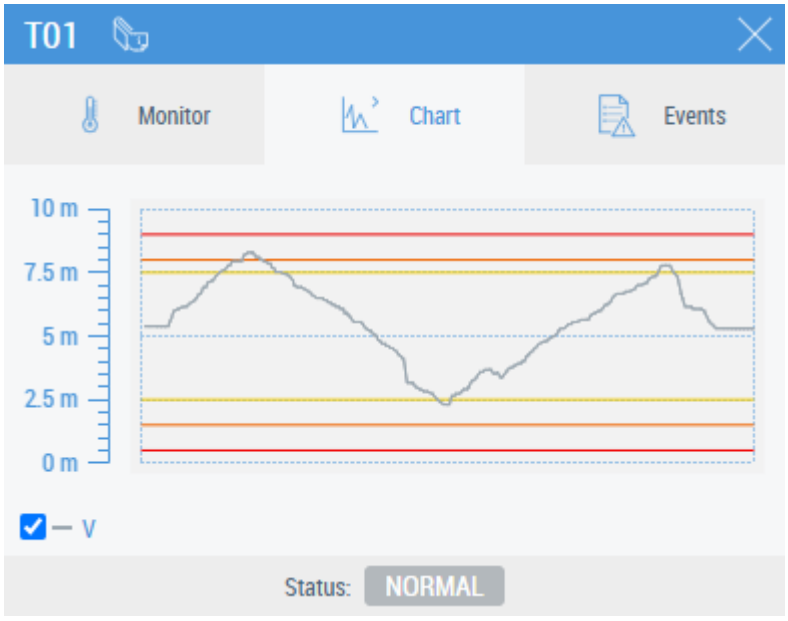
Ready-to-use faceplates

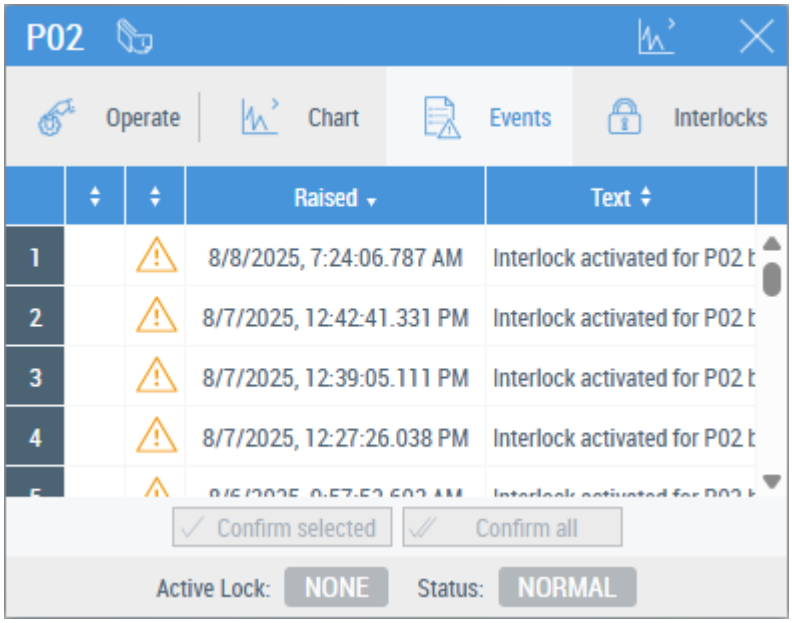
Ready-to-use faceplates provide options for:

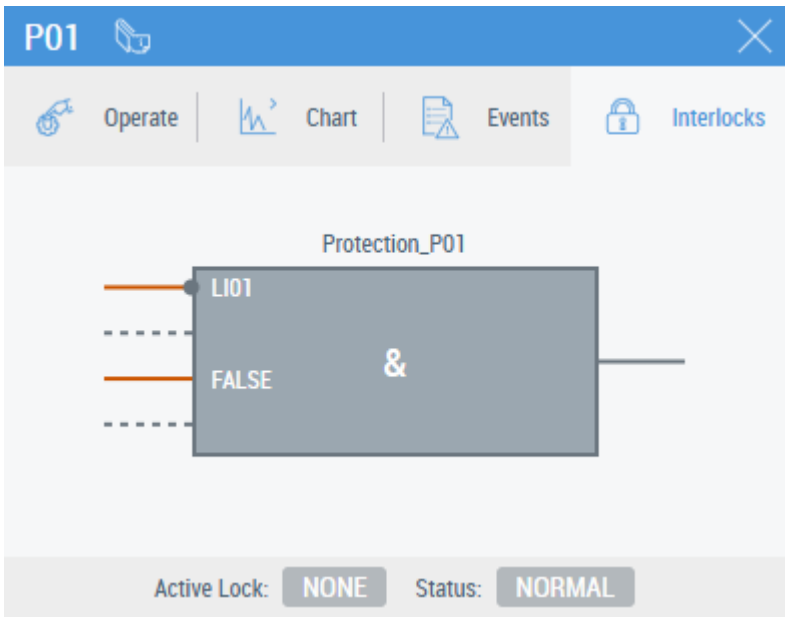
- presenting current data, alarms, and errors
- sending commands and changing setpoints
- configuring settings, error behavior, alarm conditions
- presenting charts for data
- presenting an event log for the control
- displaying the LockView control to clarify the reason for the locking out operation
- showing current status and active lock in the footer




T01
Monitor
Chart
Events
0.7
1
0.75
0.5
0.25
0
Alarm Low Limit:
0.1
Alarm High Limit:
0.9
Warning Low Limit:
0.15
Warning High Limit:
0.85
Tolerance Low Limit:
0.2
Tolerance High Limit:
0.8
Status:
NORMAL


T01
Monitor
Chart
Events
10 m
7.5 m
5 m
2.5 m
0 m
V
Status:
NORMAL


P02
Operate
Chart
Events
Interlocks
Raised
Text
1
8/8/2025, 7:24:06.787 AM
Interlock activated for P02 b
2
8/7/2025, 12:42:41.331 PM
Interlock activated for P02 b
3
8/7/2025, 12:39:05.111 PM
Interlock activated for P02 b
4
8/7/2025, 12:27:26.038 PM
Interlock activated for P02 b
Confirm selected
Confirm all
Active Lock:
NONE
Status:
NORMAL



Customizable functionality

Along with the preconfigured properties of controls and faceplates, there are lots of ways to customize data binding to controls and the content of faceplates.

1.2 Scenarios

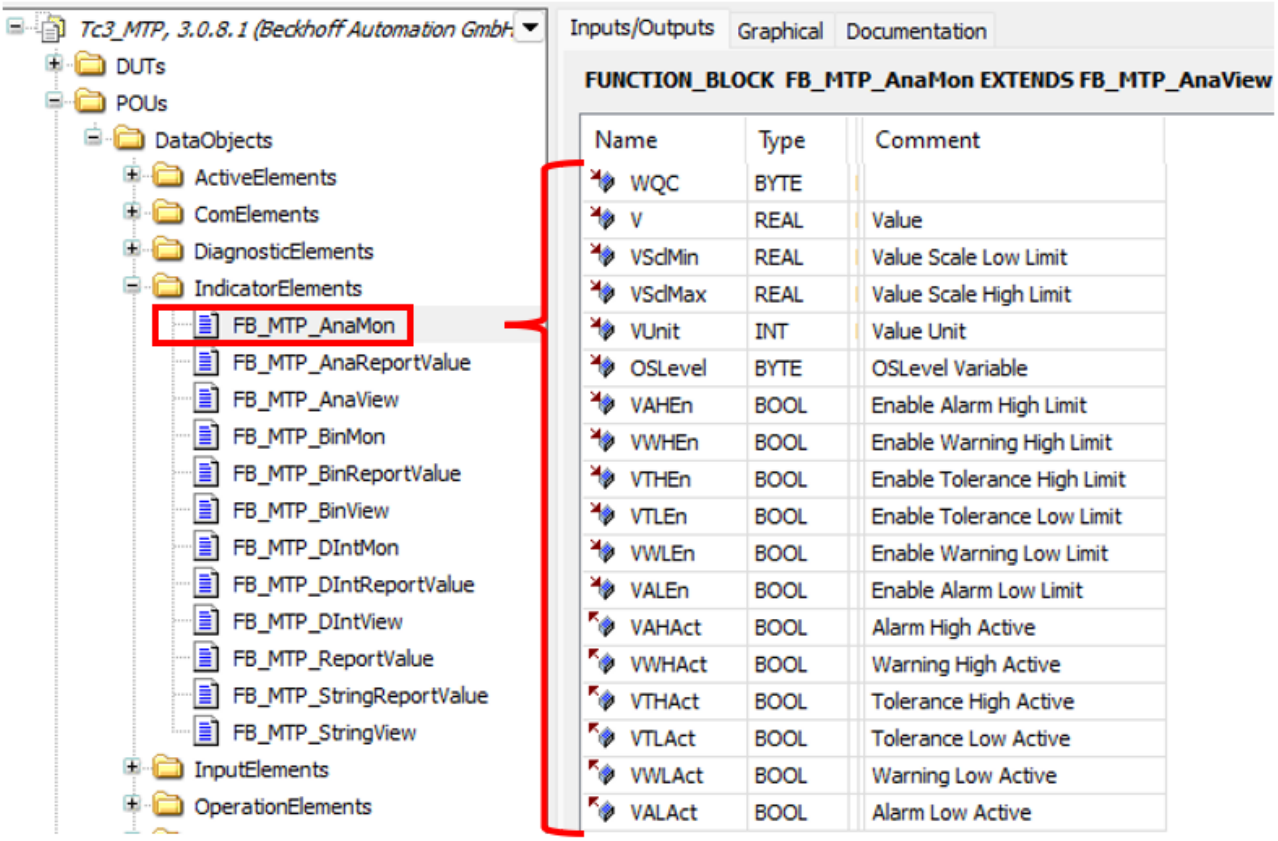
1.2.1 Ways of mapping PLC variables to the HMI control

There are three types of mapping variables for mapping the controls to the PLC: MTP library mapping, custom object (e.g. FB) mapping, and direct variable mapping.

- MTP library mapping is the most convenient and fastest method because a standardized (see MTP Runtime FBs) ready-to-use PLC FB instance is bound to the control via a single **Data Symbol** property.
- Custom FB mapping allows single mapping via the **Data Symbol** property to be implemented as well. It differs from the MTP library mapping in that the user binds the control to an instance of the custom-developed FB which the user has enriched with attribute pragmas. The advantage in comparison to direct variable mapping is that this enrichment only has to be done once in the PLC and not for every instance in the HMI.
- Direct variable mapping means that a control property can be linked to corresponding PLC variable. This method can be used alone or in combination with the previous two. If used in combination, separate control properties can be set or overridden by mapping to a separate PLC variable.

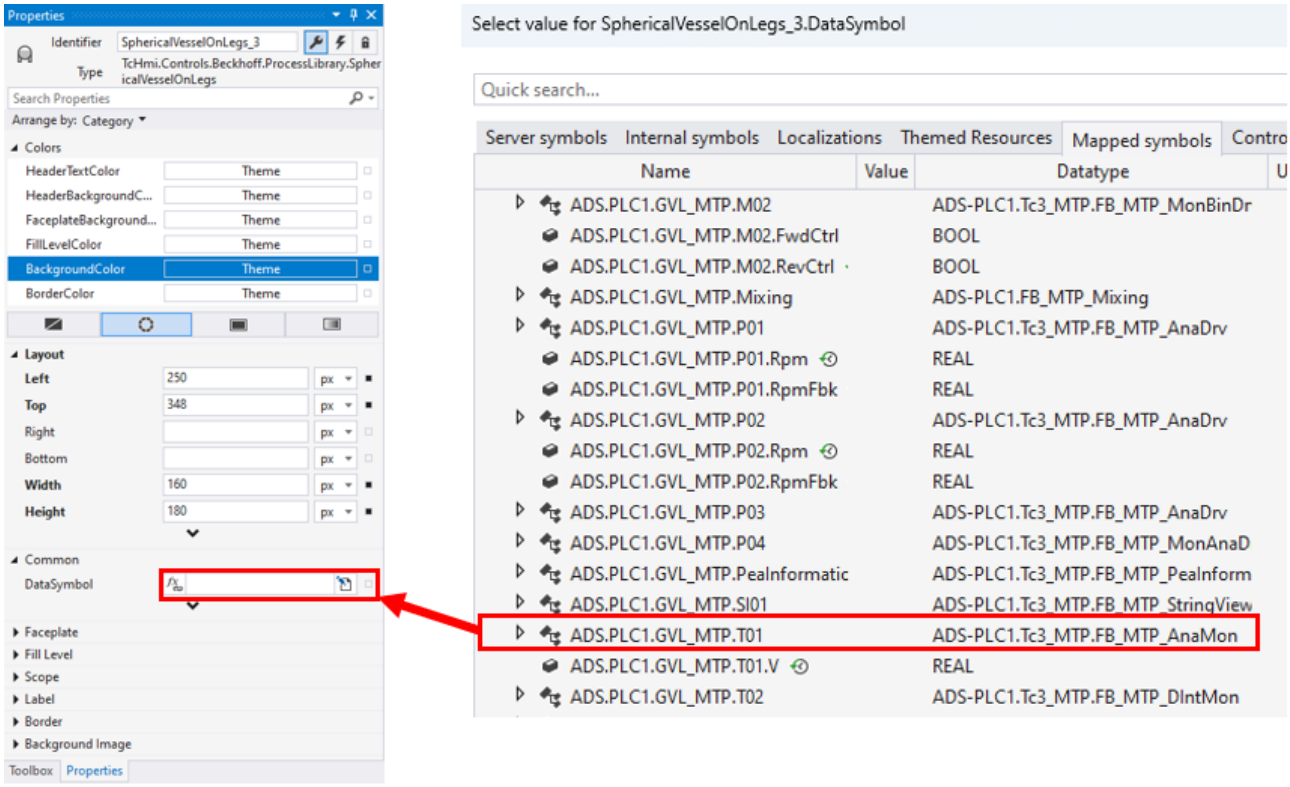
MTP library mapping

The TF8400 | TwinCAT 3 MTP Runtime library includes standardized interfaces. The following shows a sequence which displays how a particular standardized type is bound to the corresponding control.



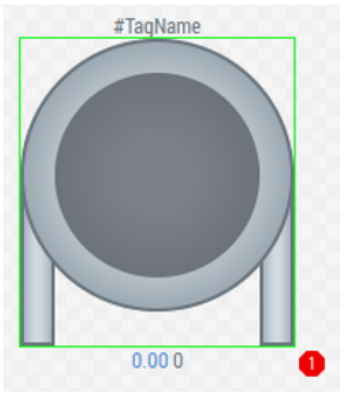
MTP library FBs are instantiated in the PLC and become available for mapping in the HMI project.

```
558     T01: FB_MTP_AnaMon := (
559                     TagName := 'T01',
560                     TagDescription := 'Tank Level',
561                     WQC := 255,
562                     VSclMin := 0,
563                     VSclMax := 1,
564                     VUnit := E_MTP_Unit.Liter,
565                     V := 0,
566                     VAHEn := TRUE,
567                     VAHLim := 1,
568                     VALEn := TRUE,
569                     VALLim := 0,
570                     VWHEn := TRUE,
571                     VWHLim := 0.9,
572                     VWLEn := TRUE,
573                     VWLLim := 0.1,
574                     VTHEn := TRUE,
575                     VTHLim := 0.8,
576                     VTLEn := TRUE,
577                     VTLLim := 0.2
578                     );
```



The result appears immediately after mapping:

a)



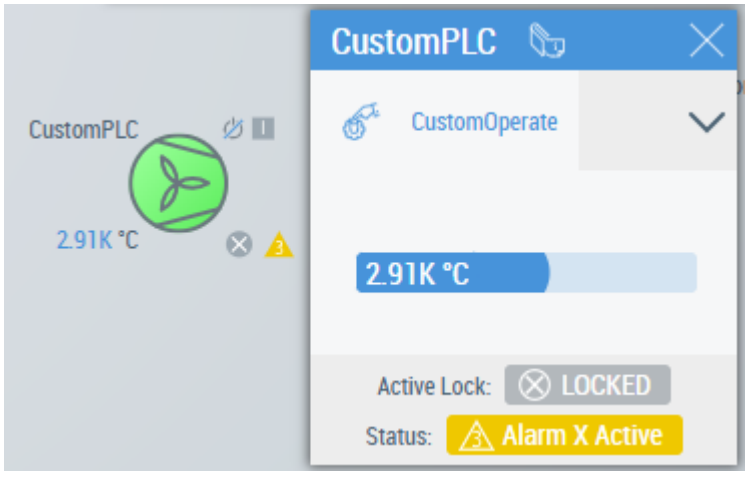
b)

Custom FB mapping

Using the special `'TcHmi.ProcessLibrary'` pragma attribute makes a custom FB available for mapping to the control via the **Data Symbol** property. The pragma attributes must be used to create the FB. Then all its instances pass to the mapped control:

- configuration data (defines some properties and behavior of the control) given in the attributes
- current values from variables marked with the attributes

```
{region 'TcHmiProcessLibrary.FaceplateParameters'}
    {attribute 'TcHmi.ProcessLibrary.Modal' := 'True'}
    {attribute 'TcHmi.ProcessLibrary.Movable' := 'False'}
    {attribute 'TcHmi.ProcessLibrary.RestoreBounds' := 'False'}
    {attribute 'TcHmi.ProcessLibrary.HideWithControl' := 'True'}
    {attribute 'TcHmi.ProcessLibrary.ReshowWithControl' := 'False'}
{endregion}
{region 'TcHmiProcessLibrary.LabelParameters'} [45 lines]
FUNCTION_BLOCK FB_AttrTest2
VAR_INPUT
    {attribute 'TcHmi.ProcessLibrary' := 'LabelText'}
    sName: STRING := 'CustomPLC2';

    {attribute 'TcHmi.ProcessLibrary' := 'LabelValue'}
    {attribute 'TcHmi.ProcessLibrary.Format' := 'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.AnalogCompressionValueFormatter,4'}
    fValue: REAL;

    {attribute 'TcHmi.ProcessLibrary' := 'LabelValueUnit'}
    {attribute 'TcHmi.ProcessLibrary.Format' := 'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.MTPUnitValueFormatter'}
    nUnit: INT := 1001;

    {attribute 'TcHmi.ProcessLibrary' := 'LabelStateMode'}
    nStateMode: INT := 1;

    {attribute 'TcHmi.ProcessLibrary' := 'LabelSourceMode'}
    nSrcMode: INT := 1;

    {attribute 'TcHmi.ProcessLibrary' := 'LabelInterlockStatus'}
    nInterlockStatus: INT := 1;

    {attribute 'TcHmi.ProcessLibrary' := 'LabelAlarmStatus'}
    nAlarmStatus: INT := 1;

    {attribute 'TcHmi.ProcessLibrary' := 'GraphicStatus'}
    nGraphicStatus: INT := 0;

    {attribute 'TcHmi.ProcessLibrary' := 'HeaderText'}
    sHeaderText: STRING := 'New Header';
END_VAR
```

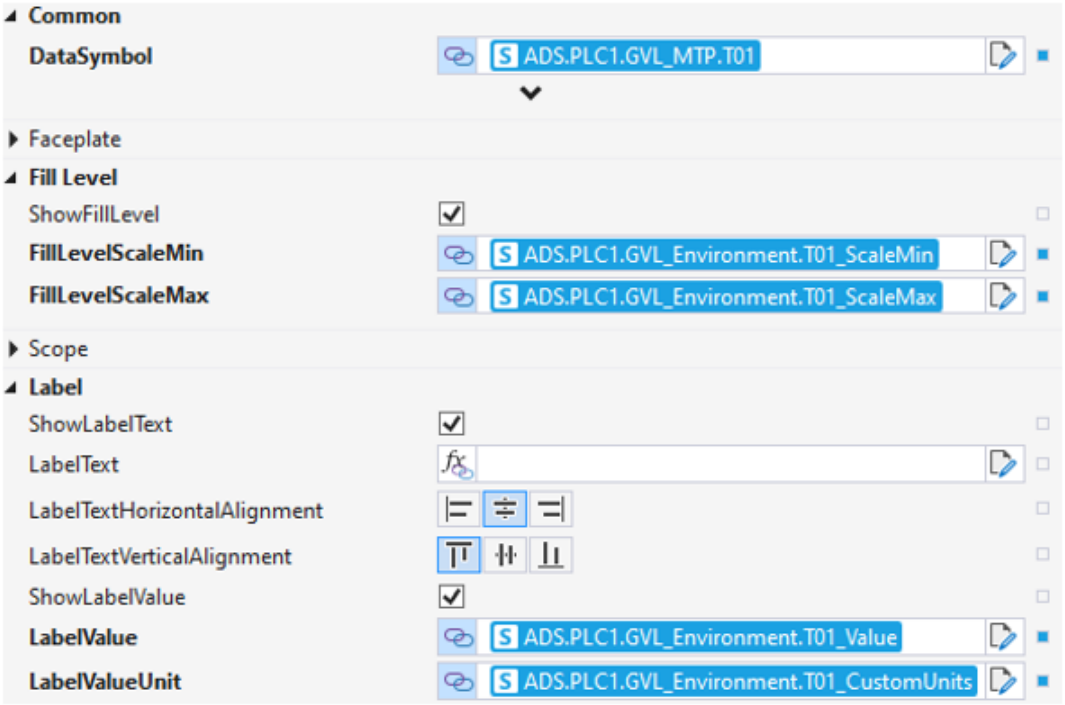


More detailed information on using PLC attributes is given in the PLC attribute functionality [▶ 82] and Using PLC attribute functionality [▶ 21] chapters.

PLC variable direct mapping/overwriting

This is the standard method of variable mapping in TwinCAT HMI: every property of the HMI Process Library control which is available via the **Property Window** can be mapped to a separate variable of a corresponding type.

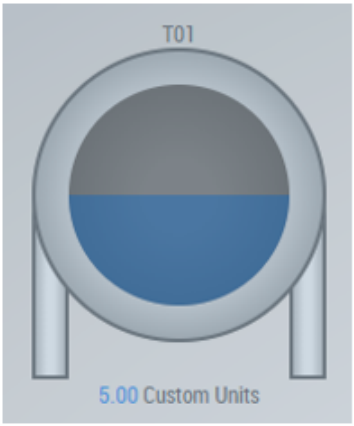
a)



b)

T01_ScaleMax	REAL	10
T01_ScaleMin	REAL	0
T01_Value	REAL	5
T01_CustomUnits	STRING	'Custom Units'

c)



Separate mapping for each control instance

Mapping variables to the control properties (which are needed) has to be done separately for each HMI control instance.

Sometimes, some specific properties of the control which have already been mapped to MTP or custom FB via the **Data Symbol** property need to be overridden; this constitutes a complex scenario. It makes using the controls more flexible.

Mapped data prioritization

Data set via specific properties has priority over the **Data Symbol** property.

1.2.2 Faceplate types

Faceplate types can be distinguished by binding them to the control. There are three options: predefined faceplate, **Tabs** property, and **Faceplate Control** property. The main difference between the **Tabs** and **Faceplate Control** properties is that a TabNavigation control is predefined and can be populated with multiple UserControls in **Tabs** while the **Faceplate Control** property expects a single dedicated UserControl.

The types below are sorted according to their priority (the lowest priority is at the top):

- Predefined faceplates make it easy to create an HMI. They already have all the essential elements for the HMI Process Library controls. This only works with MTP PLC library (TF8400 | TwinCAT MTP Runtime) or custom FBs that implement the exact variable interface.
- **The Tabs** property allows custom tabs to be created for a pre-implemented TabNavigation.
- **The Faceplate Control** property allows the complete faceplate content to be overwritten with customized UserControls. The main difference from using **Tabs** property is that there are no tabs on the faceplate in this case.

Using PLC attributes to set Tab and Faceplate Control properties

Special PLC pragma attributes can be used to set the **Tab** and **Faceplate Control** properties.

NOTICE

Protection from deleting faceplates created by the user

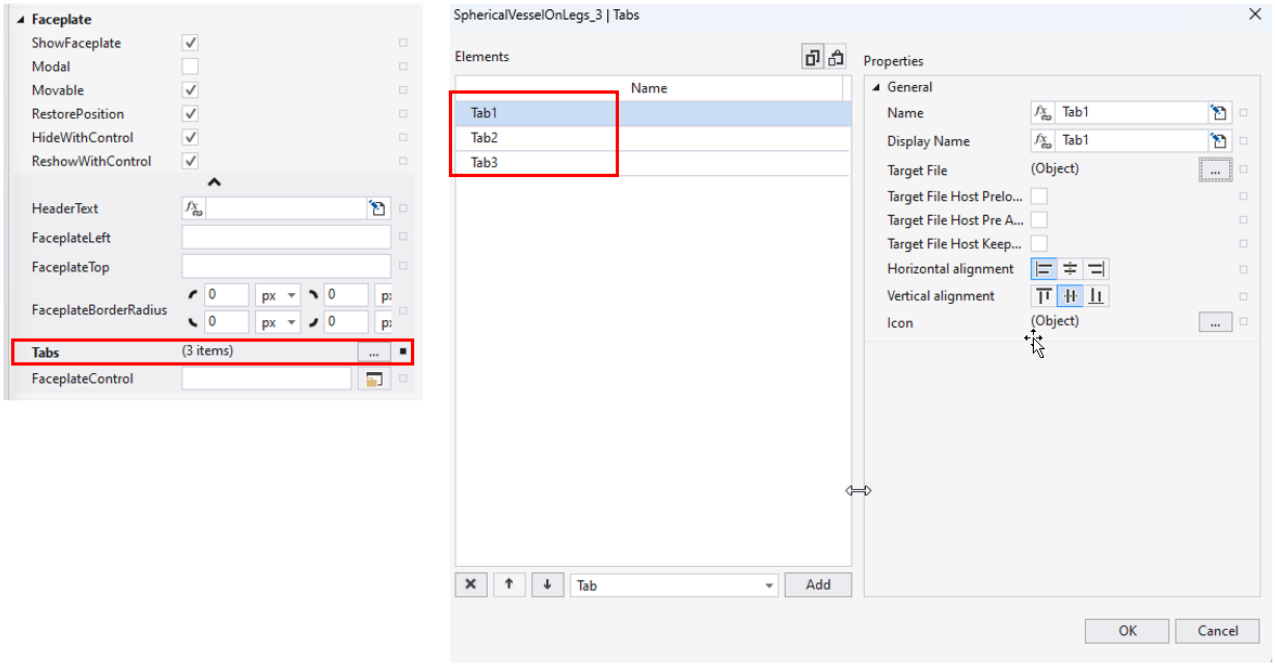
When the HMI Process Library NuGet package is being installed/updated, it places all the ready-to-use faceplate UserControls into the *Faceplates* folder for the TwinCAT HMI project and it is suggested that all the faceplates with the same names should be overwritten in this folder. To avoid overwriting, the own/ adapted faceplate UserControls should **have names that are different** from the names of the faceplates from the HMI Process Library or should be **moved into another folder**.

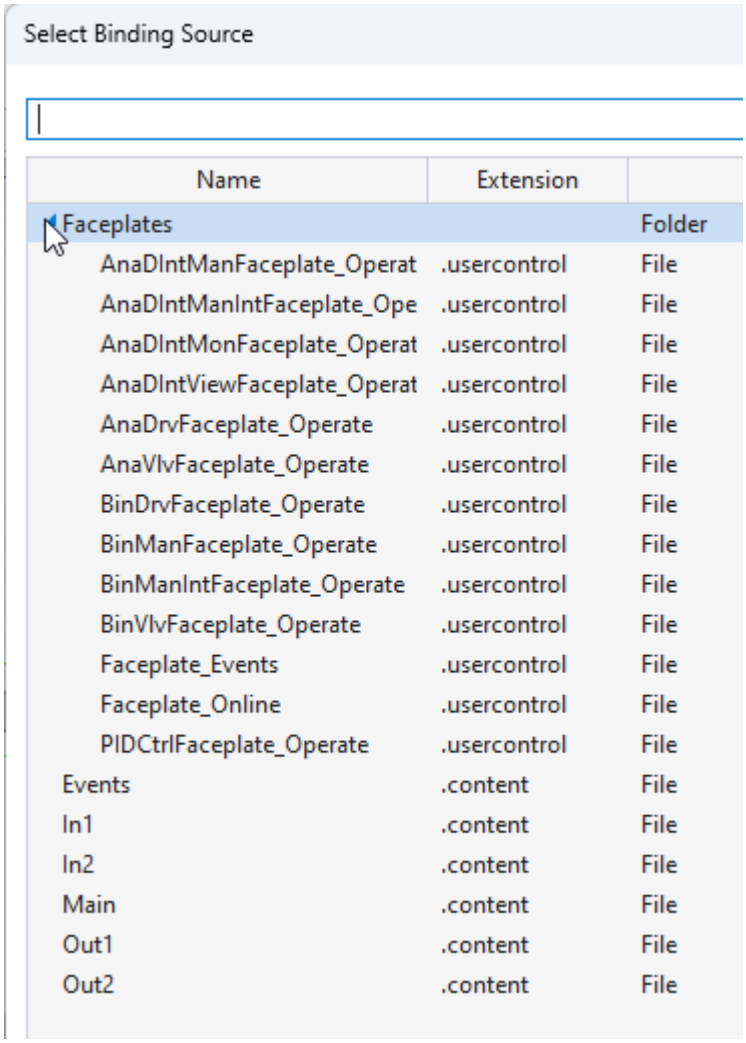
Predefined faceplate

Predefined faceplates are available for all controls and are used automatically if the **Data Symbol** property is linked to the MTP library FB instance. The **Interlocks Tab** property of the control can be used to add an Interlocks tab to the predefined tabs without having to recreate the predefined tabs.

Tabs property

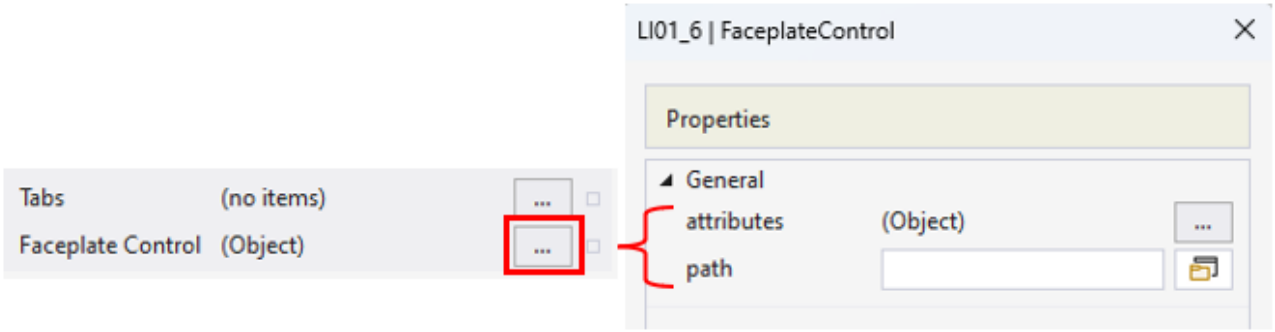
The **Tabs** control property allows faceplates that are available or have been created by the user and content elements to be shown as a faceplate tabs.





Faceplate Control property

The **Faceplate Control** control property allows the path to the desired UserControl to be set, along with its parameters, via **attributes**.



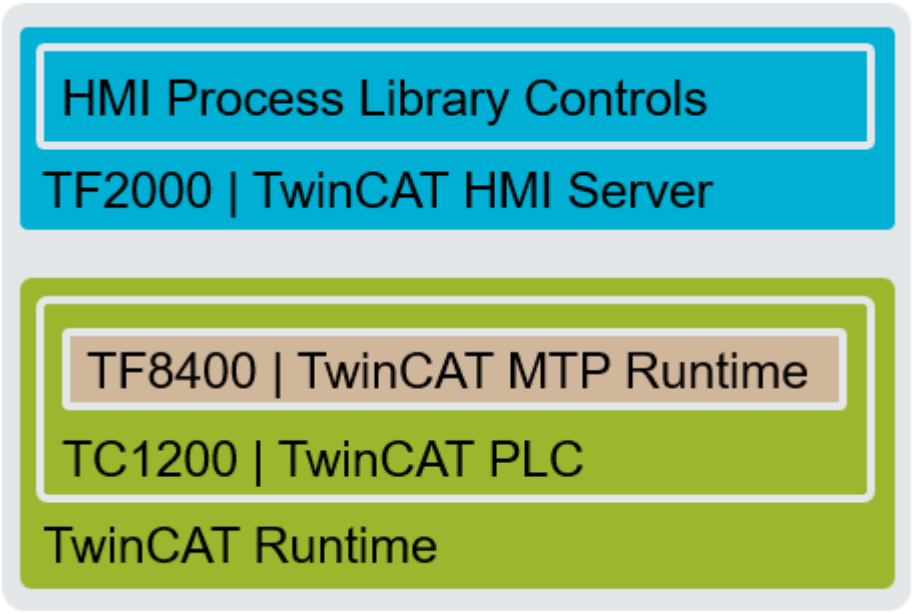
1.2.3 System architectures

Since the HMI Process Library is based on TwinCAT HMI, all scenarios for TwinCAT HMI are accessible for the HMI Process Library as well.

When we take the various options for mapping PLC variables and the flexibility provided by using faceplates into account, there are two main scenarios for using the HMI Process Library from a system architecture point of view: either with or without using TwinCAT MTP PLC Library.

With TwinCAT PLC MTP Library

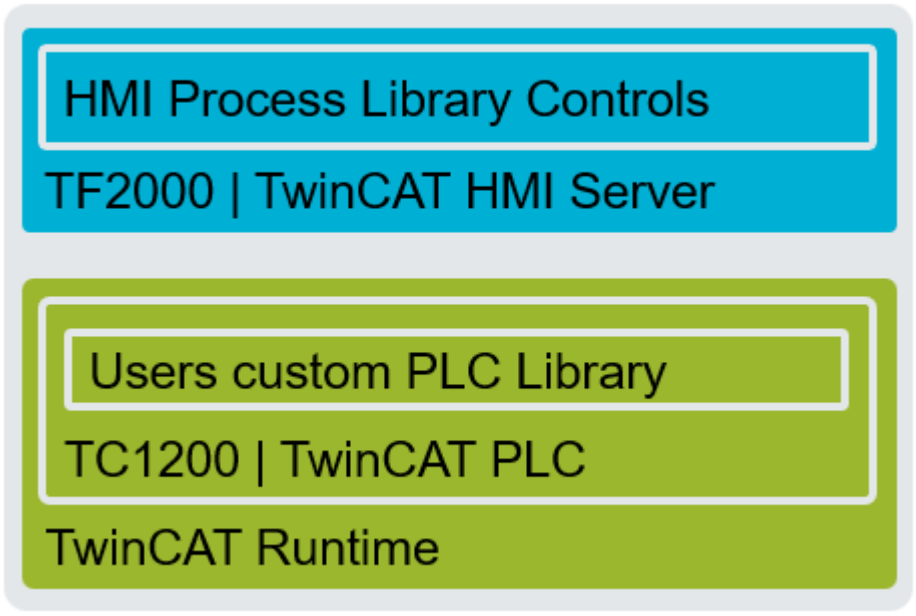
The HMI Process Library provides controls compatible with TF8400 | TwinCAT MTP Runtime FBs that conform to the MTP Specification. This allows the control to be mapped to just one PLC symbol.



This architecture implies that PLC can also be controlled by POL/DCS.

Without TwinCAT PLC MTP Library

Using the HMI Process Library without the PLC MTP Library gives the user the option to utilize ready-to-use controls with the user's own FBs for standard and non-standard technological equipment.



1.3 User requirements

The user of this library requires basic knowledge of the following:

- TwinCAT System (See documentation Basics)
- TwinCAT HMI
- TwinCAT Scope

- TwinCAT EventLogger
- TwinCAT MTP (depends on whether it is used with the library-standard MTP components)
- TwinCAT OPC UA (depends on whether communication with POL/DCS is used)

1.4 System requirements

The HMI Process Library requires TwinCAT HMI version 1.14.

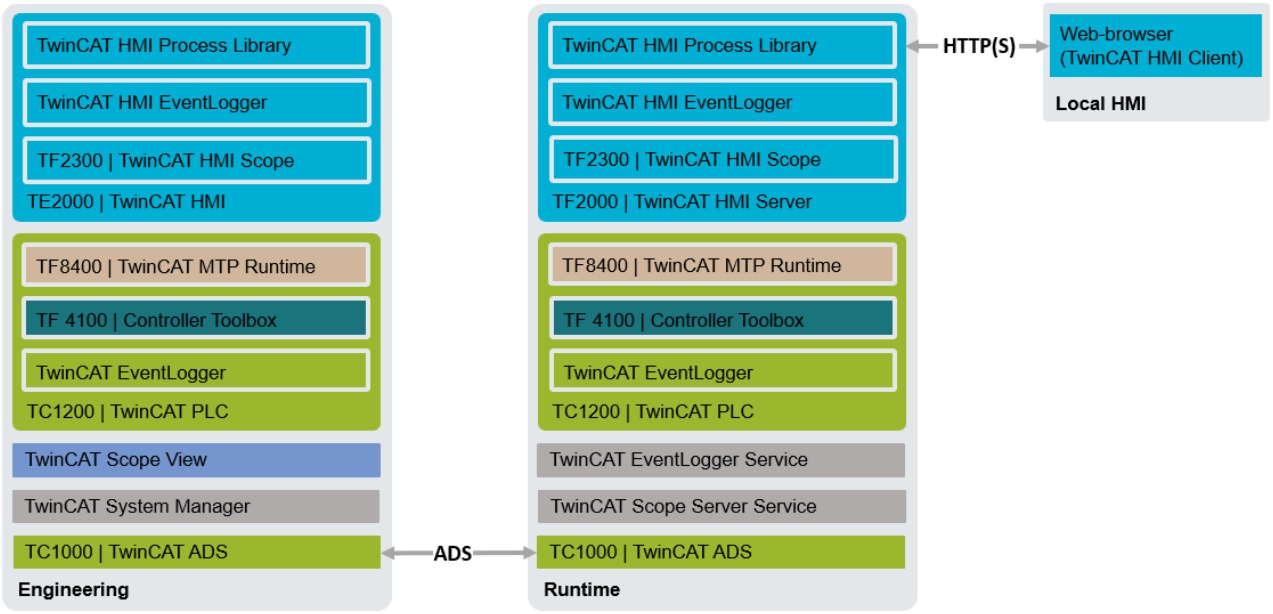
All other requirements are the same as for TwinCAT HMI. Please refer to the documentation for TE2000 TwinCAT 3 HMI Engineering.

Microsoft.TypeScript.MSBuild NuGet package

Updates for this package should be taken from the Beckhoff Offline Repository (not from nuget.org).

1.5 Architecture

In the figure below, the TwinCAT HMI Process Library is shown in relation to the other TwinCAT components. This information is more comprehensive than in the System architectures [▶ 14] chapter. It does not include all TwinCAT components, but it gives an idea of the TwinCAT HMI Process Library's operation and interactions.



The TwinCAT HMI Process Library is an extension of TwinCAT HMI. TwinCAT HMI Engineering and TwinCAT HMI Server must be used respectively to develop and run projects which use the library.

The TwinCAT HMI Process Library gets events from TwinCAT EventLogger, which is why it needs these components:

- TwinCAT HMI EventLogger to show events in HMI and get events from the EventLogger
- (optional) TwinCAT EventLogger library at PLC level to generate and get events from the EventLogger
- TwinCAT EventLogger Service on a Runtime to handle events at OS level

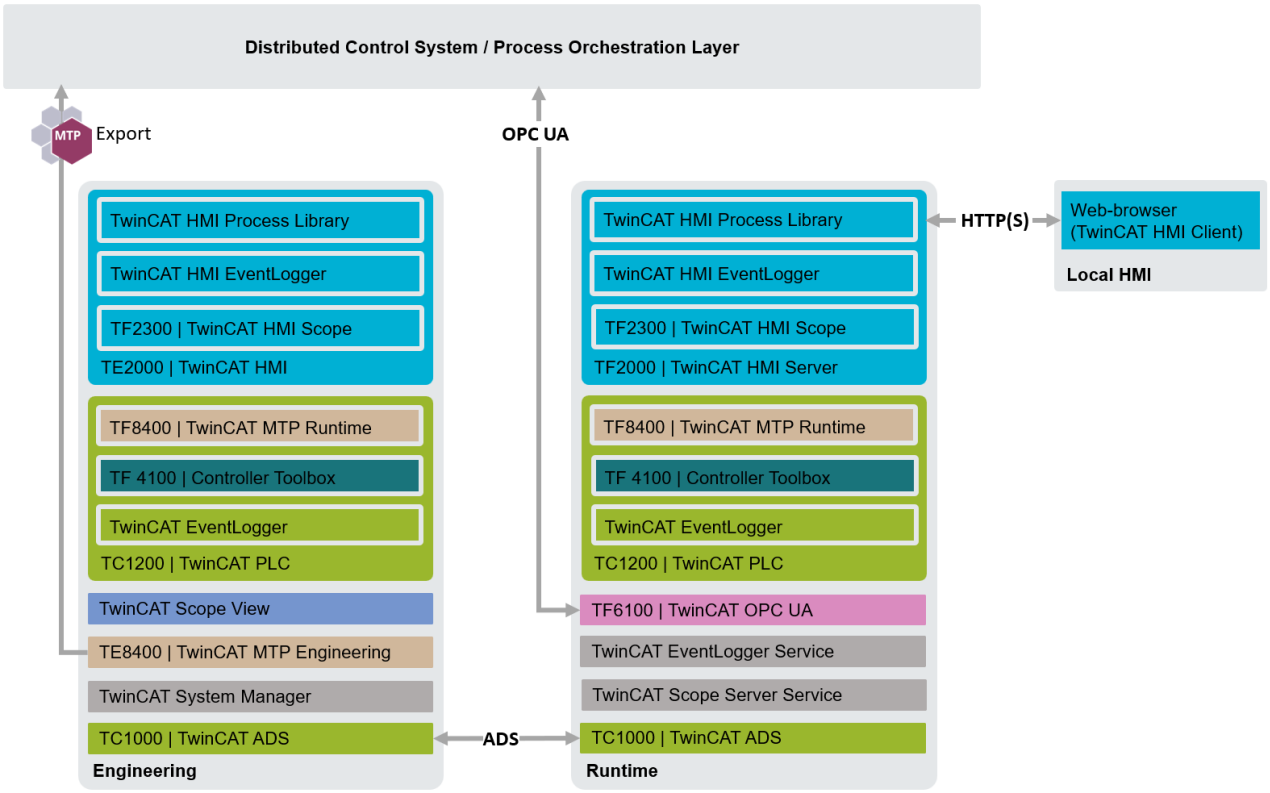
The TwinCAT HMI Process Library can draw charts using the TwinCAT Scope functionality. To do this, it needs the following options:

- TwinCAT HMI Scope
- TwinCAT Scope View on the Engineering side to create and configure charts
- TwinCAT Scope Server Service to handle the Scope data at OS level

TwinCAT MTP Runtime is optional. It provides standardized FBs for technological equipment. If it is used, two more components should be added:

- TwinCAT MTP Runtime itself
- (optional) TwinCAT Controller Toolbox to implement the PID controller that TwinCAT MTP Runtime uses. Other FBs implementing PID functionality can also be used.

The TwinCAT HMI Process Library can be used to create local HMIs for systems which are controlled by POL/DCS. The image below shows the architecture a system like this:



For more information, see the TE8400 | TwinCAT 3 MTP Engineering and TF8400 | TwinCAT 3 MTP Runtime documentation.

1.6 Business model

The TwinCAT HMI Process Library doesn't need a special dedicated license, but it does require a TF2000 license at minimum to run. Other licenses listed in the table below provide additional functionality for solutions with the TwinCAT HMI Process Library.

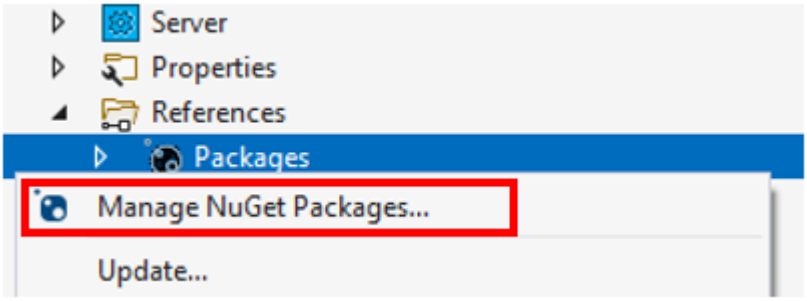
License	Purpose	Target type
TF2000 | TwinCAT 3 HMI Server	Runs HMI Process Library inside TwinCAT HMI	Runtime
TF2300 | TwinCAT 3 HMI Scope	Represents Scope charts inside TwinCAT HMI	Runtime
TC1200 | TwinCAT 3 PLC	Runs PLC program which provides data for TwinCAT HMI	Runtime
TF8400 | TwinCAT 3 MTP Runtime	Implementation of directive compliant MTP interfaces	Runtime
TF4100 | TwinCAT 3 Controller Toolbox	Implementation of PID controller	Runtime
TE8400 | TwinCAT 3 MTP Engineering	Development and configuration of parts of MTP system	Engineering
TF6100 | TwinCAT 3 OPC UA	Provides interaction with POL/DCS via OPC UA	Runtime

2 Installation

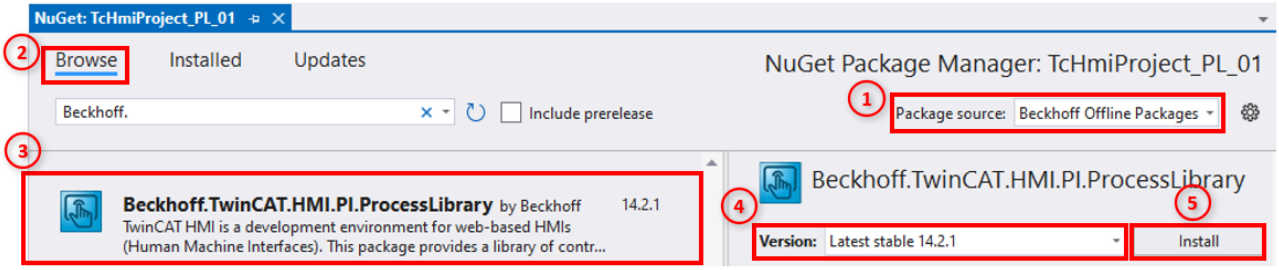
Using the TwinCAT HMI Process Library requires installation of TwinCAT HMI components. See the corresponding instructions in the documentation for TE2000.

The steps for installing a package for the HMI Process Library are shown below.

1. TwinCAT Package Manager should be used for the installation of the HMI Process Library. It's available as a separate package **TwinCAT.HMI.PI.ProcessLibrary** or within the **TE8400 | TwinCAT 3 MTP Engineering** workload.
2. **Select Manage NuGet Packages** in the context menu of the **Packages** node:

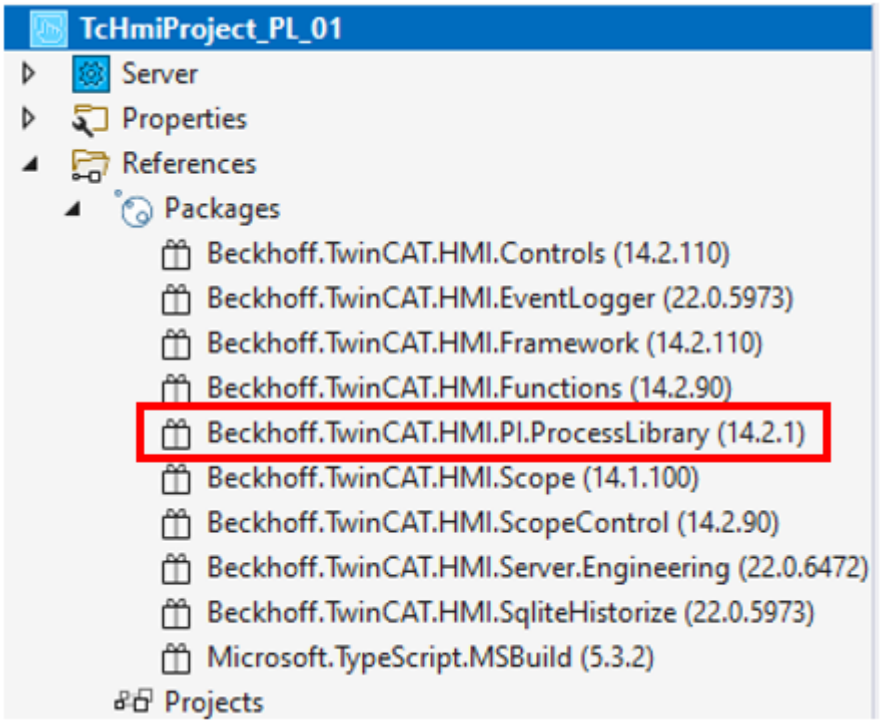


3. Then choose **Package source: Beckhoff Offline Packages** → **Browse** tab → **Beckhoff.TwinCAT.HMI.PI.ProcessLibrary** → Choose **Version** → **Install** button:



⇨ When the HMI Process Library package is installed, other packages that it depends on are also installed.

⇨ The package should appear after the installation:

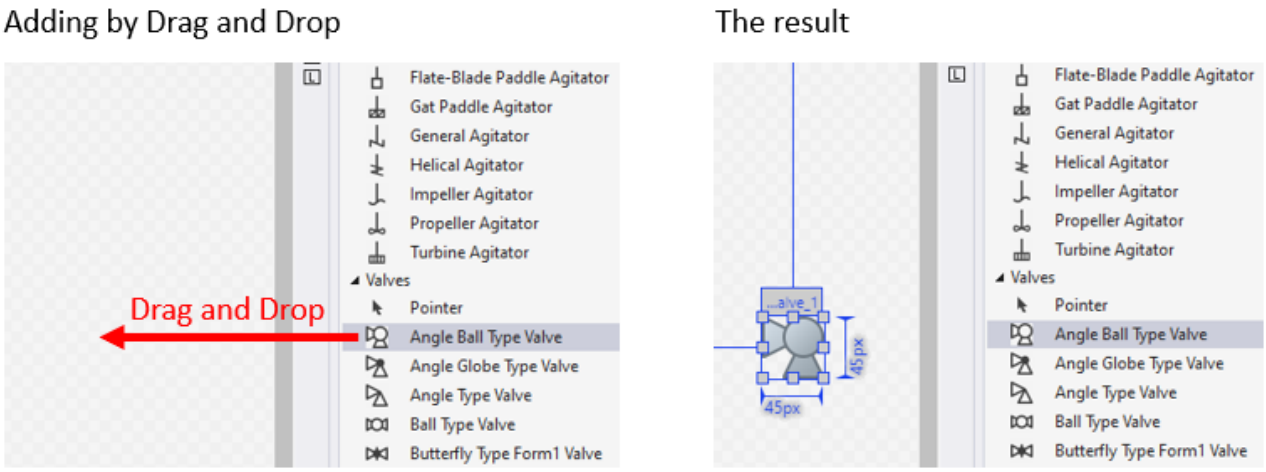


3 Quick start

This quick start chapter provides very basic steps that show you how to add the library controls to the HMI project and map them to PLC data (e.g. MTP and customized FB) using the **Data Symbol** property. Other basic information is also provided.

3.1 Adding a control to the project

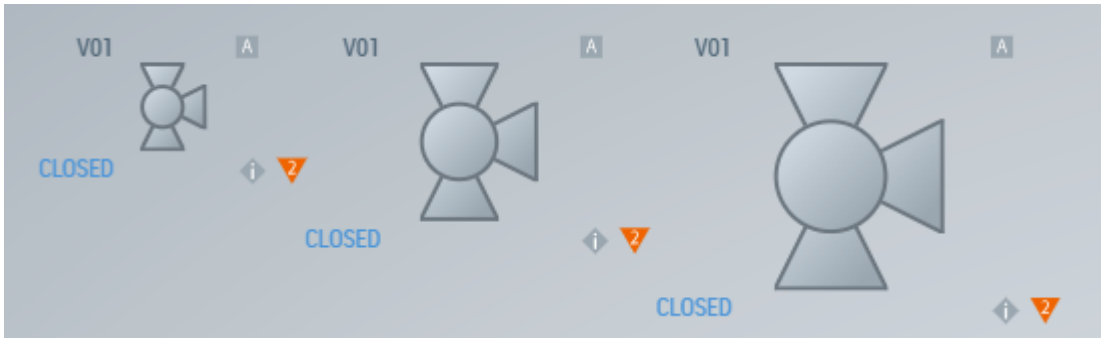
Adding a control from the HMI Process Library to the project is done in the same way (drag and drop from the **Toolbox** panel) as for other TwinCAT HMI controls:



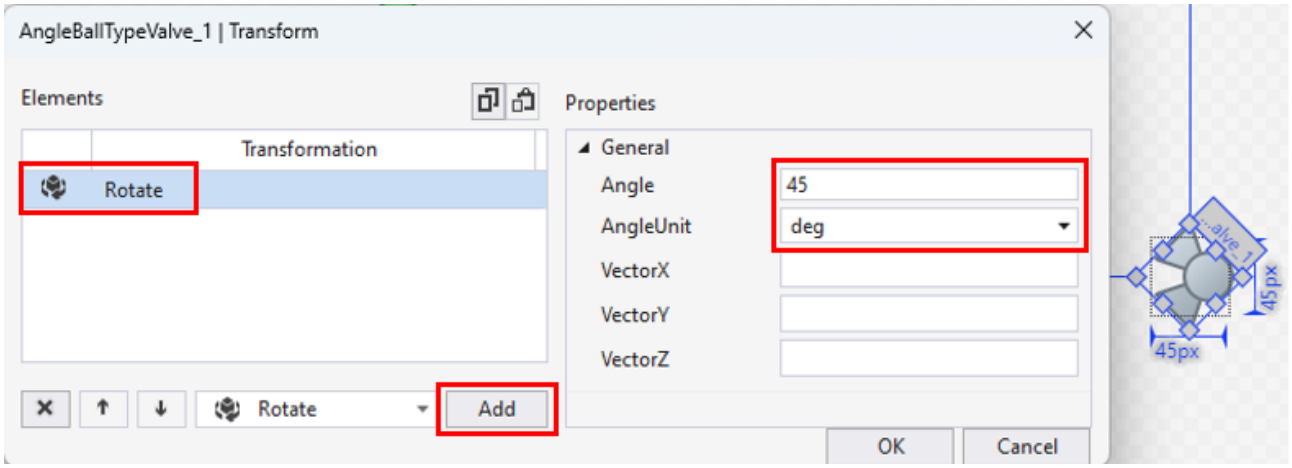
3.2 Editing control properties

The properties, events, and permissions are available for the HMI Process Library controls in a **Properties** panel. The description of properties is in the Components and functionality [▶ 26] chapter.

The position and size of the control can be changed using graphical interface or the relevant properties.



Rotation can be applied to the control using the **Transform** property:



3.3 PLC program interaction scenarios

The main property which is used to set behavior of the control is the **Data Symbol**. Binding the FB to the **Data Symbol** property with an appropriate set of variables may be sufficient to completely specify the behavior of the control. However, other properties can also be set at the same time to customize the behavior. Other properties take priority over similar variables of the **Data Symbol** property.

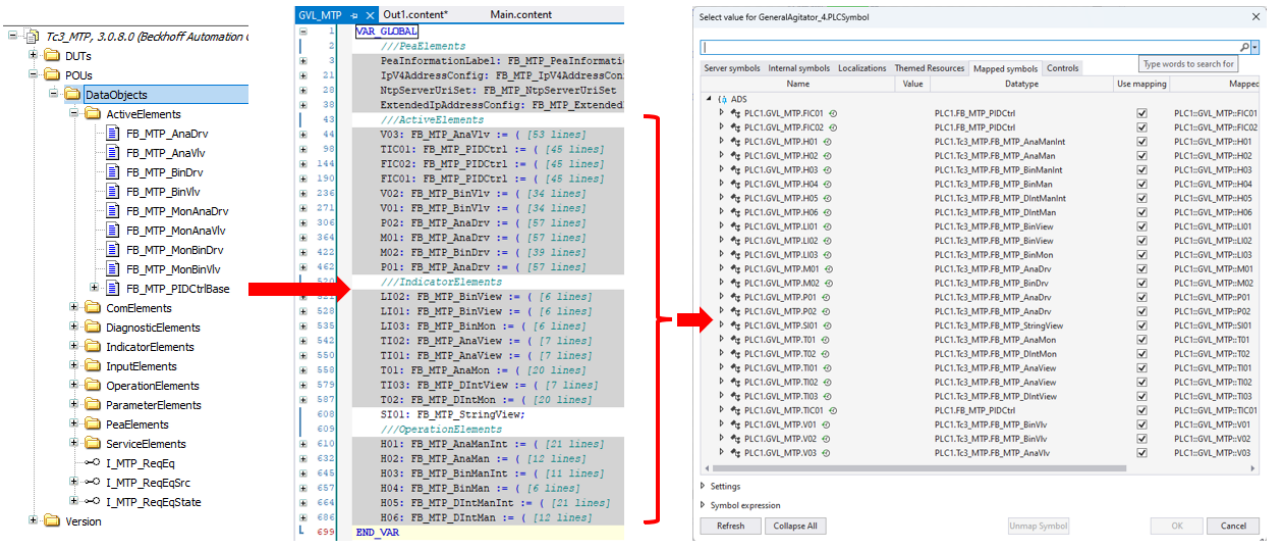
There are two ways to make a PLC FB bindable to the **Data Symbol** property:

- using PLC attribute functionality
- using the MTP PLC library

3.3.1 Using the MTP PLC Library

TwinCAT MTP PLC Library (*Tc3_MTP*) comes with TF8400 | TwinCAT 3 MTP Runtime. It provides a number of ready-to-use FBs corresponding to the MTP Specification. Simply declare an instance of FB for a certain DataAssembly in a PLC program and initialize its variables in line with the HMI Process Library control properties. After this and successfully compiling the PLC project, this FB will be available for linking to the **Data Symbol** property:

Ready-to-use FBs | Instances in PLC program | Available to link to Data Symbol property in HMI Engineering



TwinCAT MTP PLC Library implements structures of variables according to the MTP Specification and also provides services which are appropriate for this standard. This means that the FBs include algorithms for controlling the equipment. This creates a complete system, from the equipment control level to the HMI.

3.3.2 Using PLC attribute functionality

Customized PLC FBs can be created for binding to the **Data Symbol** property of the TwinCAT HMI Process Library control element. Using PLC attribute functionality in a customized FB gives access to the control properties. A description of potential attributes is given in the PLC attribute functionality [▶ 82] chapter.

3.3.2.1 Using configuration PLC attributes

The common form for using configuration PLC attributes is:

```
{attribute 'TcHmi.ProcessLibrary.[Attribute Name]' := '[Value/Values separated by comma]'}
```

Where to use configuration attributes

Configuration attributes should be used in the declaration. Depending on a type, before FUNCTION_BLOCK keyword or within the VAR_ area right before a declaration of the variable which is to be linked to an element of the control.

Pragma can be used to group the attributes, {region}…{endregion}.

Examples of using configuration PLC attributes

The TcHmi.ProcessLibrary.FaceplateControl attribute with the region pragma:

```
{region 'TcHmiProcessLibrary.FaceplateControl'}
    {attribute 'TcHmi.ProcessLibrary.FaceplateControl.TargetFile':= 'CustomFaceplates/
fbAttrTest_Operate.usercontrol'}
    {attribute 'TcHmi.ProcessLibrary.FaceplateControl.Parameter' := 'DataSymbol:= THISEXP^'}
{endregion}
```

Configuring a faceplate with the region pragma:

```
{region 'TcHmiProcessLibrary.FaceplateParameters'}
    {attribute 'TcHmi.ProcessLibrary. ShowFaceplate' := 'True'}
    {attribute 'TcHmi.ProcessLibrary.Modal' := 'False'}
    {attribute 'TcHmi.ProcessLibrary.Movable' := 'True'}
    {attribute 'TcHmi.ProcessLibrary.RestoreBounds' := 'False'}
    {attribute 'TcHmi.ProcessLibrary.HideWithControl' := 'True'}
    {attribute 'TcHmi.ProcessLibrary.ReshowWithControl' := 'False'}
{endregion}
```

Configuring a faceplate tab:

```
{attribute 'TcHmi.ProcessLibrary.Tab1.Name' := 'CustomOperate'}
{attribute 'TcHmi.ProcessLibrary.Tab1.TargetFile' := 'CustomFaceplates/
fbAttrTest_Operate.usercontrol,true,true,false'}
{attribute 'TcHmi.ProcessLibrary.Tab1.Parameter' := 'DataSymbol:= THISEXP^'}
{attribute 'TcHmi.ProcessLibrary.Tab1.Alignment' := 'Center,Center'}
{attribute 'TcHmi.ProcessLibrary.Tab1.Icon' := 'PLPATH^/Images/Operate.svg,32,32,px,px'}
```

Configuring a faceplate tab with the ready-to-use **Faceplate_Chart.usercontrol**:

```
{attribute 'TcHmi.ProcessLibrary.Tab2.Name' := 'Chart'}
{attribute 'TcHmi.ProcessLibrary.Tab2.TargetFile' := 'Faceplates/
Faceplate_Chart.usercontrol,true,true,false'}
{attribute 'TcHmi.ProcessLibrary.Tab2.Parameter' := 'DataSymbolPath1:=
THIS^.fValue,SclMin:=0,SclMax:=10000,Unit:=STAG^THIS^::nUnitETAG^'}
{attribute 'TcHmi.ProcessLibrary.Tab2.Alignment' := 'Center,Center'}
{attribute 'TcHmi.ProcessLibrary.Tab2.Icon' := 'PLPATH^/Images/Chart.svg,32,32,px,px'}
```

Making a variable for chart historization

The ‘Chart’ attribute should be added before the declaration of the variable (within VAR_ area). A chart will be shown for this to historize it:

```
{attribute 'TcHmi.ProcessLibrary.Chart'}
fValue: REAL;
```

Configuring a faceplate tab using ready-to-use faceplate:

```
{attribute 'TcHmi.ProcessLibrary.Tab3.Name' := 'CustomOperate2'}
{attribute 'TcHmi.ProcessLibrary.Tab3.TargetFile' := 'Faceplates/
BinVlvFaceplate_Operate.usercontrol,true,true,false'}
```

```
{attribute 'TcHmi.ProcessLibrary.Tab3.Parameter' := 'DataSymbol:=STAG^ PLC1.GVL_MTP.V01ETAG^'}
{attribute 'TcHmi.ProcessLibrary.Tab3.Alignment' := 'Center,Center'}
{attribute 'TcHmi.ProcessLibrary.Tab3.Icon' := 'Imports/Images/EX-logo.svg,32,32,px,px'}
```

Configuring label appearance:

```
{region 'TcHmiProcessLibrary.LabelParameters'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextHorizontalAlignment' := 'Right'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextVerticalAlignment' := 'Bottom'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.Left' := '-10'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.LeftUnit' := 'px'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.Top' := '-50'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.TopUnit' := '%'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.Right' := '-10'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.RightUnit' := 'px'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.Bottom' := '-10'}
    {attribute 'TcHmi.ProcessLibrary.LabelTextPadding.BottomUnit' := 'px'}
{endregion}
```

3.3.2.2 Data PLC attributes

Variables inside the FB should be declared with the data PLC attribute "`TcHmi.ProcessLibrary`" in order to be linked to the control properties automatically. The full template is:

```
{attribute 'TcHmi.ProcessLibrary' := '[Attribute Name]'}
```

Data attribute positioning

Data attributes should be used within **VAR_** area right before the declaration of the variable that is to be linked to an element of the control.

Examples of how to use data PLC attributes

Linking an `sName` variable to the `LabelText` property:

```
{attribute 'TcHmi.ProcessLibrary' := 'LabelText'}
sName: STRING := 'CustomPLC';
```

Linking an `nStateMode` variable to the `LabelStateMode` property:

```
{attribute 'TcHmi.ProcessLibrary' := 'LabelStateMode'}
nStateMode: INT := 1;
```

Using configuration PLC attributes for functions with data PLC attributes

Formatting functions can be applied to the properties via declaration in a PLC code. The attribute pragma "`TcHmi.ProcessLibrary.Format`" should be used:

```
{attribute 'TcHmi.ProcessLibrary.Format' := '[FormattingFunctionName]'}
```

FormattingFunctionName is the name of a formatting function listed in the Functions [▶ 78] chapter (category: Formatting). The attribute with the function should follow the data PLC attribute.

Below are some examples of how variables can be declared with the attribute pragmas:

```
{attribute 'TcHmi.ProcessLibrary' := 'LabelValue'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.AnalogCompressionValueFormatter,4'}
fValue: REAL;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValueUnit'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.MTPUnitValueFormatter'}
nUnit: INT := 1001;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValue'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.BinaryValueFormatter,EMERG,NORM'}
_V: BOOL;
```

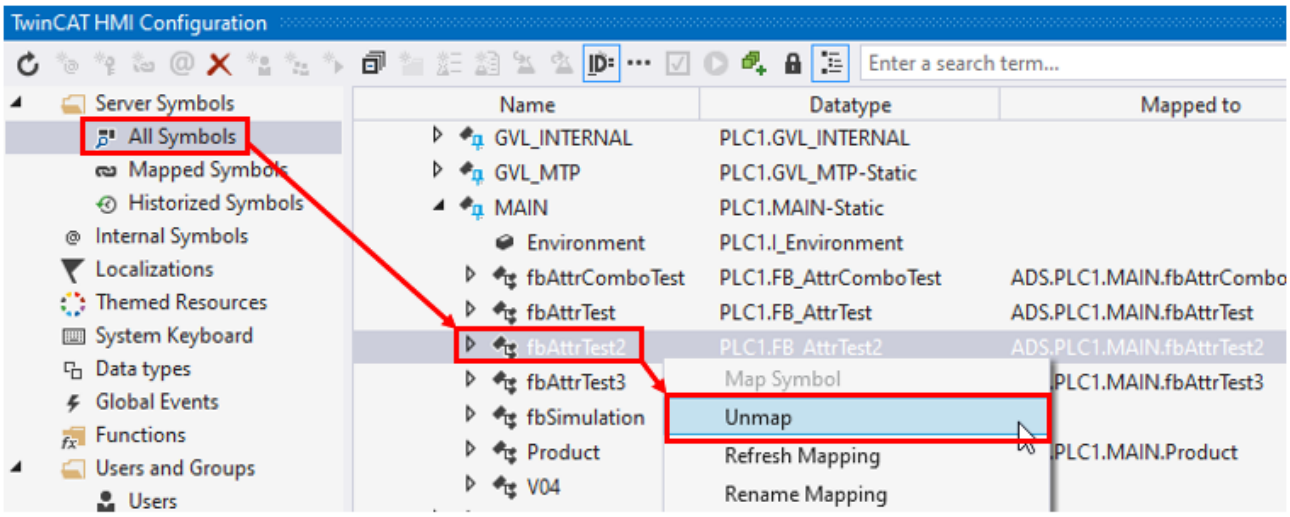
Using PLC attributes when a control needs to show several values

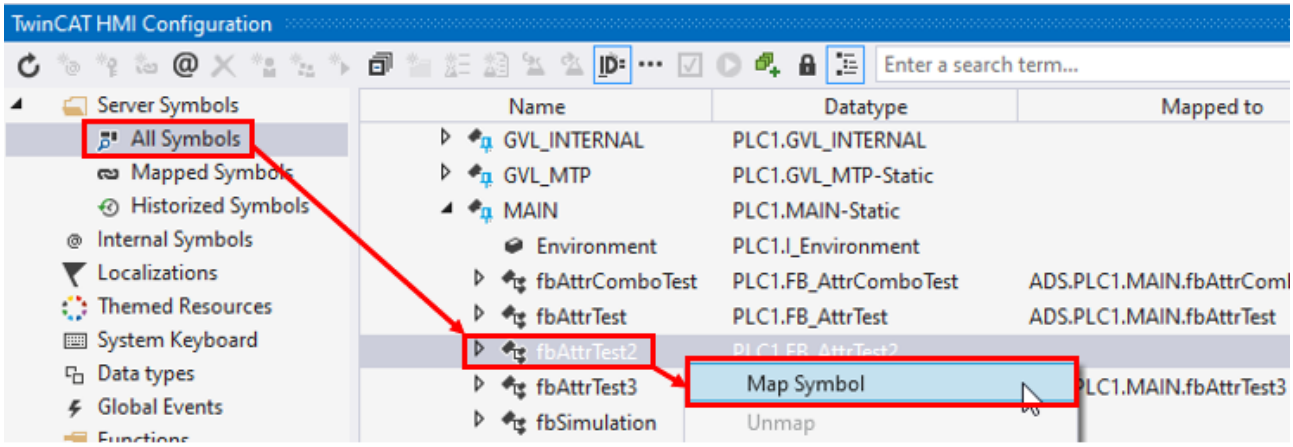
Sometimes several values need to be shown for a control element, e.g. for PID control. To implement that, the LabelValueAdd1 and LabelValueAdd2 attributes can be used along with LabelValueUnitAdd1 and LabelValueUnitAdd2 to show units. Here are examples of how to use the attributes:

```
{attribute 'TcHmi.ProcessLibrary' := 'LabelValue'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.AnalogCompressionValueFormatter,4'}
_SP: REAL;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValueAdd1'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.AnalogCompressionValueFormatter,4'}
_PV: REAL;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValueAdd2'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.AnalogCompressionValueFormatter,4'}
_MV: REAL;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValueUnit'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.MTPUnitValueFormatter'}
nSPUnit: INT;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValueUnitAdd1'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.MTPUnitValueFormatter'}
nPVUnit: INT := 1010;

{attribute 'TcHmi.ProcessLibrary' := 'LabelValueUnitAdd2'}
{attribute 'TcHmi.ProcessLibrary.Format' :=
'TcHmi.Functions.Beckhoff.PI.ProcessLibrary.MTPUnitValueFormatter'}
nMVUnit: INT := 1342;
```

3.3.2.3 Applying the PLC attribute functionality step-by-step

1. Add/edit attributes in the PLC program.
2. Build PLC program and load it into the PLC runtime.
3. **Important**: If instances of the FB with added/edited attributes are already mapped to the TwinCAT HMI, execute Unmap /Map procedure for them in the **TwinCAT HMI Configuration** window:

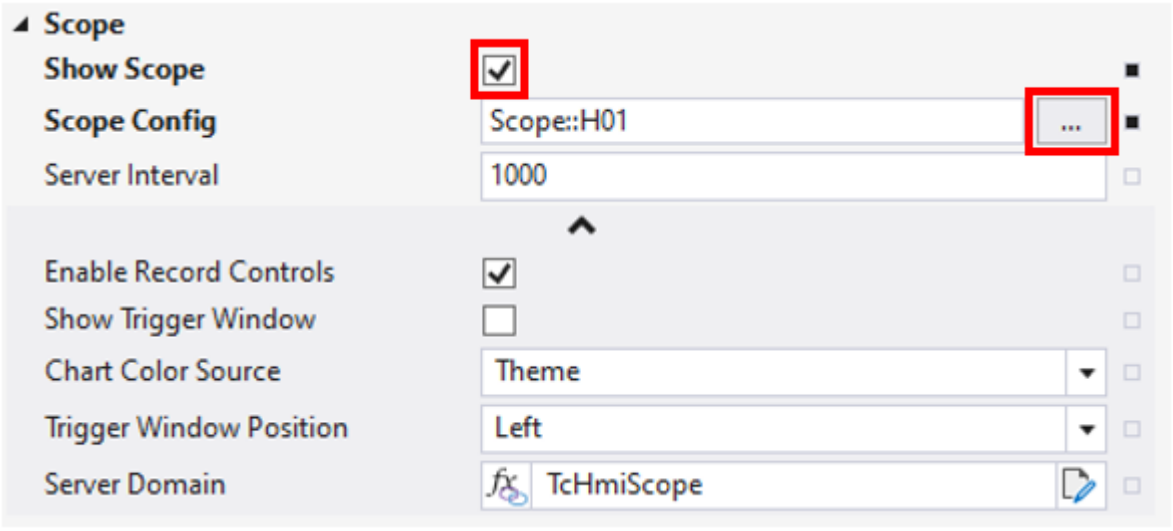




4. Reload the HMI page in the web browser.

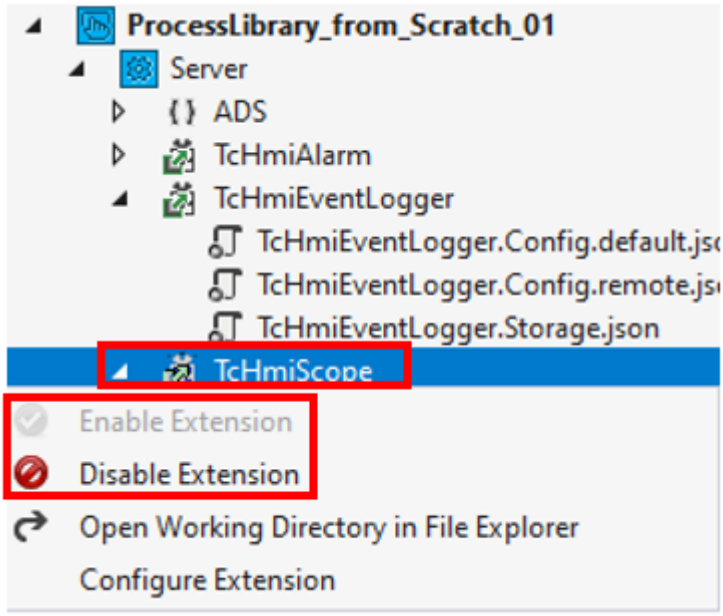
3.4 Scope configuration

To show Scope charts, the Scope project should be created. **Beckhoff.TwinCAT.HMI.Scope** NuGet package should be installed. It's necessary to enable **Show Scope** property and choose a configuration from the Scope project at **Scope Config** property:



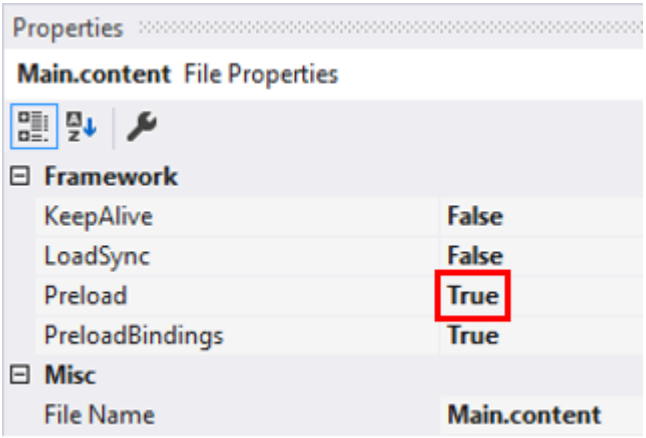
Disable-Enable TcHmiScope extension

Sometimes it's needed to Disable and then Enable **TcHmiScope** extension after changing **Scope Config** property to see the Scope chart in HMI.



3.5 Possible behavior optimizations

To make switching between HMI pages smoother and faster, it's recommended to enable **Preload** setting for content files (**.content*) of the project:

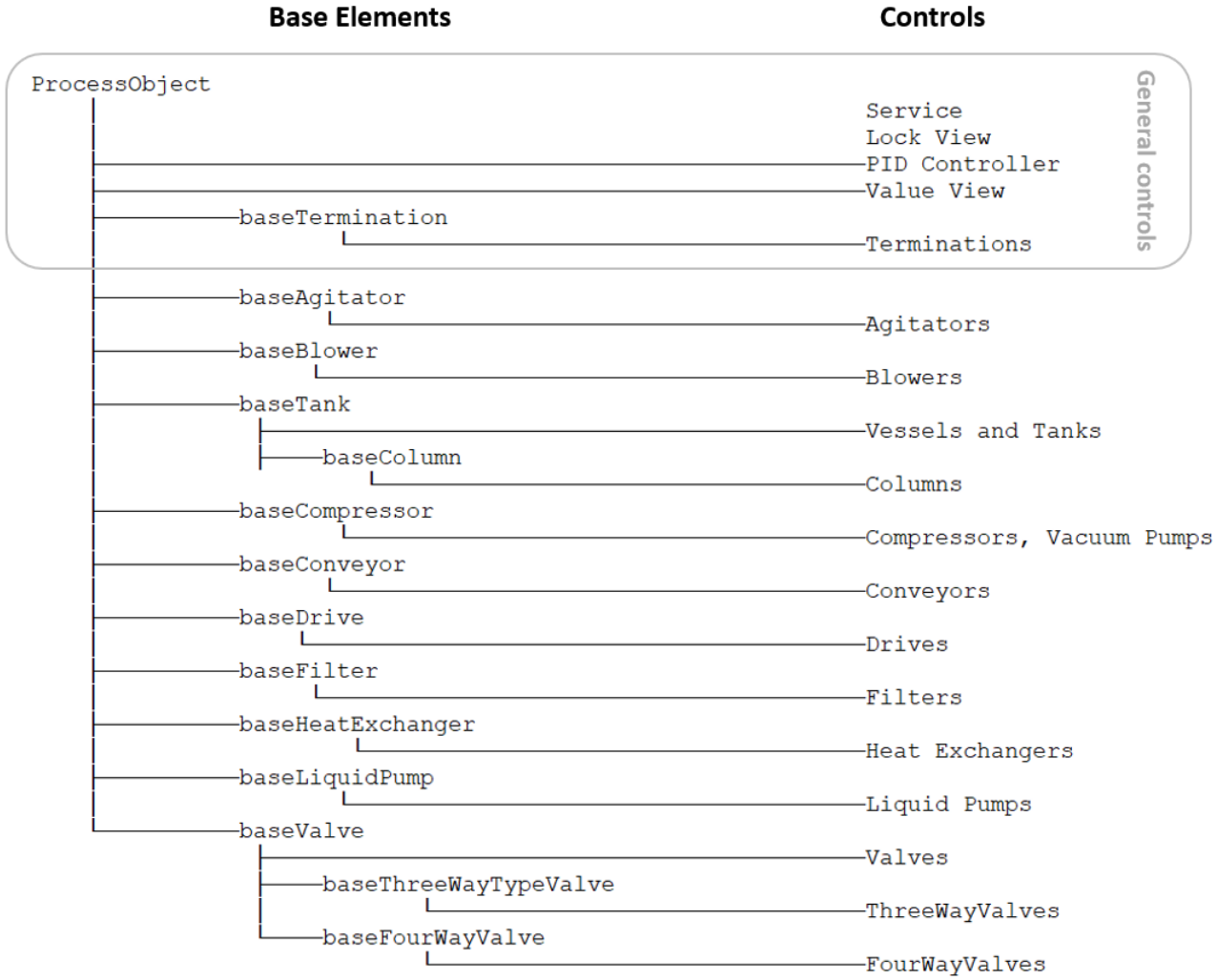


As a side effect, enabling **Preload** leads to increasing memory consumption on the client side. See TwinCAT HMI (TE2000) documentation for details.

4 Components and functionality

The TwinCAT HMI Process library consists of HMI controls and functions, for example, formatting functions, which help to present values in a more convenient format.

Most HMI controls have a hierarchical inheritance structure (only Service and Lock View controls are separate):



The ProcessObject element inheritance

The ProcessObject element is a parent of most of other controls. That means that these controls inherit ProcessObject properties, functions, and events, but some of them may be unavailable in inheriting controls.

The information below is organized so that parent element properties, functions, and events are described first in the ProcessObject [▶ 30] chapter. Furthermore, these members of the ProcessObject can be applied to the inheriting controls and are not considered separately for each inheriting control (Controls [▶ 44] chapter). If a member of the parent is changed for the inheriting control, then relevant information is provided. For each control having its special properties, functions and events are given relevant descriptions of the ones.

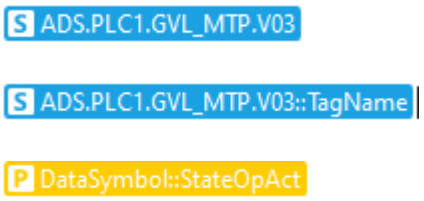
Using ProcessObject and Base Elements as controls

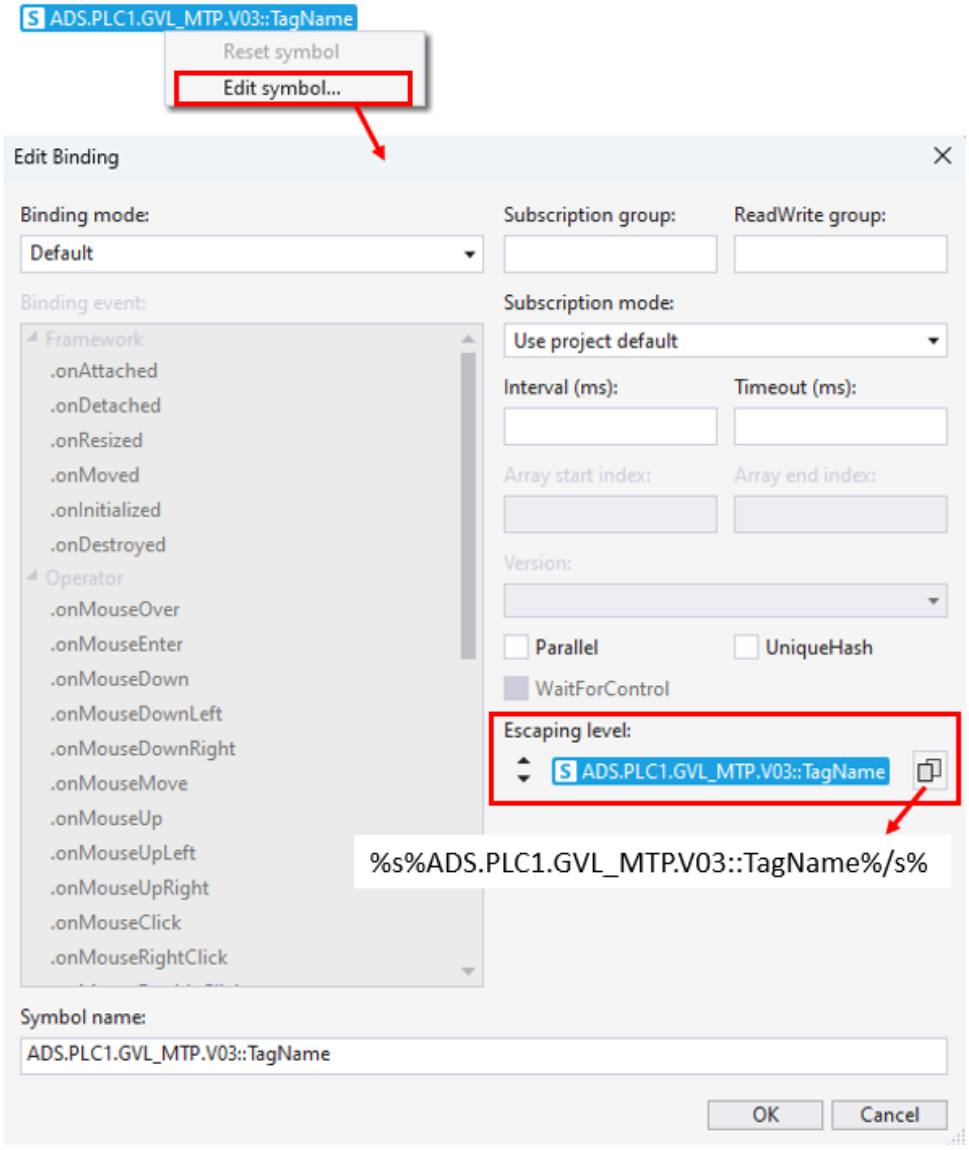
The ProcessObject can be instantiated as a control but does not have any graphical representation by default. It will receive its faceplate based on the **Data Symbol** property.

The Base Elements cannot be used as controls.

4.1 Terminology

TwinCAT HMI symbols can be represented in different forms. Some explanations, which are given in the table below, clarify the use of the terms “Symbol”, “SymbolExpression” and “SymbolPath” within this document:

Term	Explanation
Symbol	In TwinCAT HMI, a Symbol is an addressable data point exposed by the HMI server. This is generally a PLC variable exported via the TMC, but it can also be a dynamic or internal Symbol. Visually, symbols can be shown: • in graphical form:  • as a SymbolExpression (see below)

Term	Explanation
SymbolExpression	A representation of Symbol as a string wrapped in special tags (%s%, %pp%, etc., depending on the type of Symbol). For example: • `%s%ADS.PLC1.GVL_MTP.V03%/s%` • `%s%ADS.PLC1.GVL_MTP.V03::TagName%/s%` • `%pp%DataSymbol::StateOpAct%/pp%` The SymbolExpression is a text form of the graphical representation which can be found in the context menu of the Symbol:  In some cases, SymbolExpression must be used instead of the graphical representation.

SymbolPath	The Symbol is represented as a path (without wrapping in special tags), for example:
	`ADS.PLC1.GVL_MTP.V03`
	`ADS.PLC1.GVL_MTP.V03::TagName`
	Sometimes the SymbolPath must be used to address the Symbol.

4.2 ProcessObject

As the ProcessObject element is derived from the `TcHmi.Controls.System.TcHmiControl` class, all inherited properties can be viewed in the documentation for this class. All properties, functions, and events which are specific to the ProcessObject and are available for users are described below.

4.2.1 Properties

Properties of the controls are available in a **Properties Window**.

Prioritization of properties

By default, data from the **Data Symbol** property is used to represent the control and its behavior, but if other properties are set for the control, they are prioritized below the relevant data of the **Data Symbol** property.

4.2.1.1 Category: Common

The following table shows common category properties.

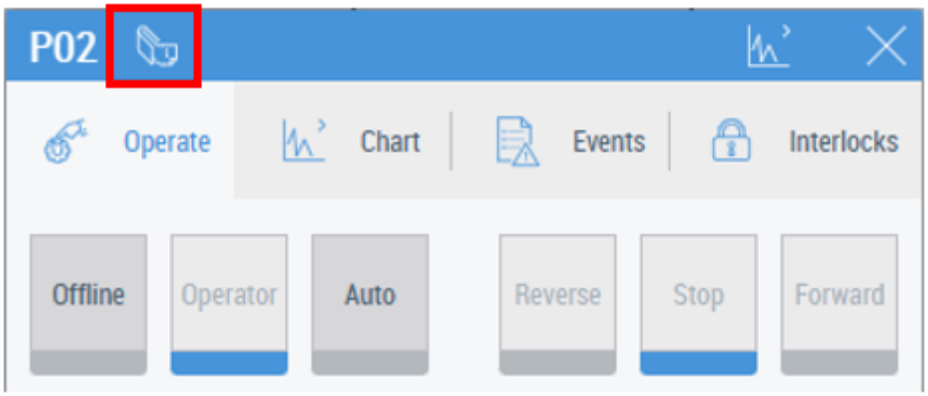
Property	Description
Data Symbol	A PLC FB (the MTP Specification compatible or user custom) to be bound to the control.
Data	Data to use for object. This property is not used if the **Data Symbol** property is configured. It allows a structure of properties which corresponds the structure of standard MTP DataObjects to be configured and these properties can then be set or bound separately.
Graphic Status	Variable that defines the graphics state, i.e., default, active, warning, error. If the behavior of the control from the **Graphic Status** is different from the ProcessObject, see the control's description for the appropriate values. If the behavior is the same as it is for ProcessObject, the **Graphic Status** property isn't mentioned for the control in its description.
Graphic Control	The path to the custom graphical representation (a UserControl with possible parameters that can be evaluated) of the control instead of the one taken from the **Data Symbol** property.
Custom Units	Variable that defines a correspondence between integer values and names of non-standard engineering units. If it contains the match for the number from the standard, it is used and prioritized over the standard one.

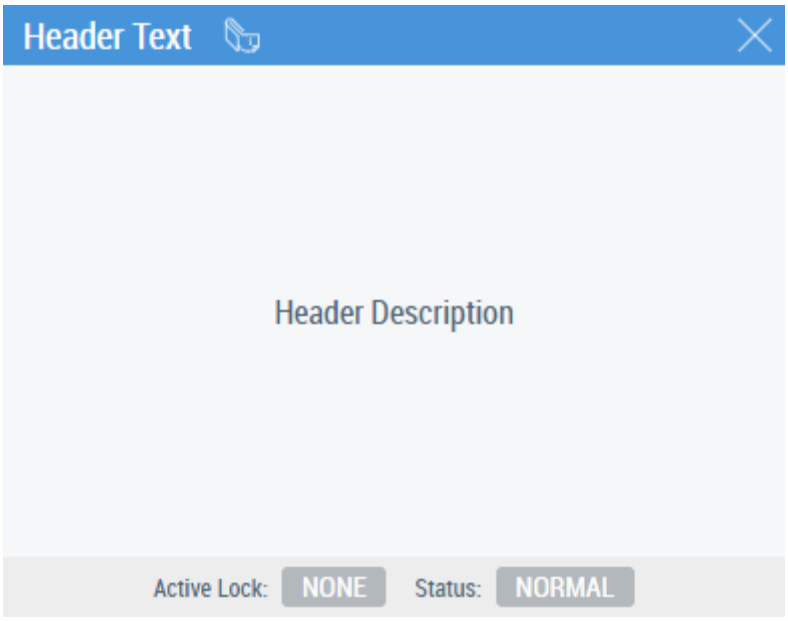
In addition to **Graphic Status** the behavior for ProcessObject is shown with a drive example:

Graphic Status value	1	2	3	Any other
Meaning	Active	Warning	Error	Default
Appearance				

4.2.1.2 Category: Faceplate

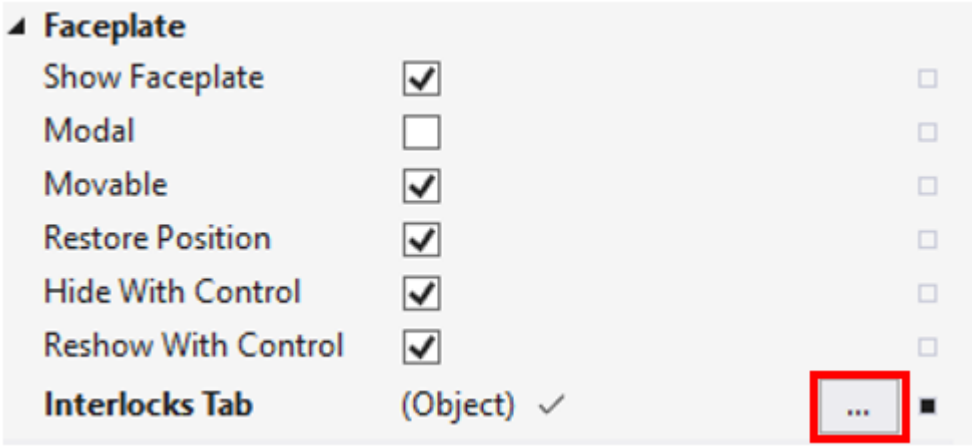
The following table shows the faceplate category properties

Property	Description
Show Faceplate	Enable/disable faceplate (1 – enable; 0 – disable).
Modal	Defines whether the popup overlay is modal. A modal popup darkens the background and is not moveable. Clicking on the darkened background closes the modal pop-up.
Movable	Defines whether the pop-up overlay is movable.
Restore Position	Defines whether the last bounds after moving/resizing the pop-up will be restored when opened the next time.
Hide With Control	Hide faceplate automatically if the control is hidden (i.e., when switching on another view). **Preload** or at least **KeepAlive** *.content page properties have to be set to TRUE in order for the faceplate not to be hidden if **Hide With Control** is FALSE.
Reshow With Control	Shows faceplate automatically if the control is shown again and the faceplate was shown before hiding the control. **Preload** or at least **KeepAlive** *.content page properties have to be set to TRUE so that the Faceplate will appear again if **Reshow With Control** is TRUE.
Interlocks Tab	Path to an Interlock UserControl to present Interlocks tab of the faceplate (if empty, the Interlocks tab isn't shown). This property is only appropriate for predefined faceplates. More detailed information is given in the Category: Faceplate [▶ 32] paragraph.
Header Text	Content of the header textbox in the faceplate.
Show Header Description	Enable/disable to show **Header Description** button on the control Faceplate:  The button switches on and off presenting **Header Description**.

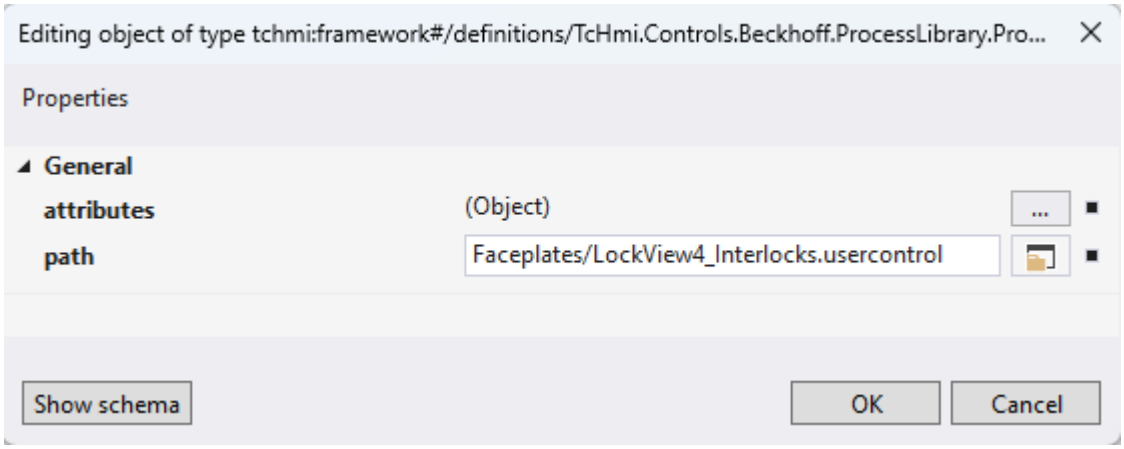
Property	Description
Header Description	Contains a description of the control. If switched on by the dedicated button, it is shown on the faceplate instead of other Tabs: 
Show Reset Button	If active, the **Reset** button is shown on the message footer independent of the emergency state of the control.
Faceplate Left	Faceplate is initially positioned on the left (default position is the control left position).
Faceplate Top	Faceplate initially positioned at top (default position is the control top position).
Faceplate Border Radius	Radius of the faceplate border.
Tabs	A list of tabs to be displayed in the faceplate. More detailed information is given in the Tabs property [▶ 34] paragraph.
Faceplate Control	Path to a custom faceplate instead of the one taken from the **Data Symbol** property. Overwrites all faceplate content which also means that the Tabs will not be shown. More detailed information is given in the Faceplate Control property [▶ 37] paragraph.

Interlocks Tab property

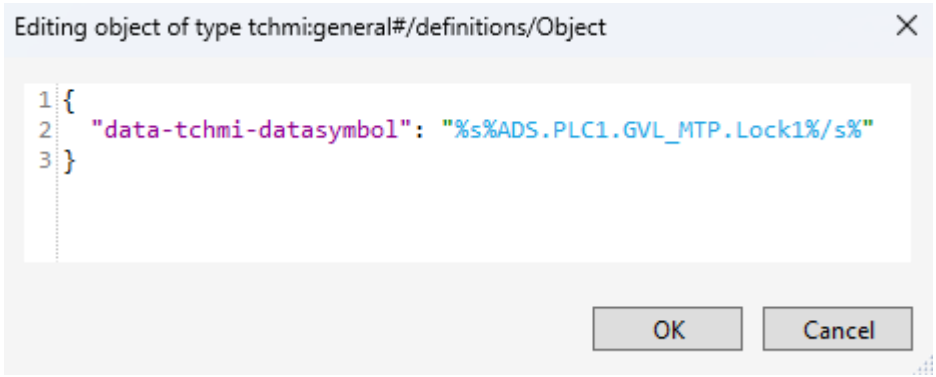
Use the "**...**" button in the Control properties to open the window for Interlocks tab [▶ 71] configuration.



Two properties should be configured in the configuration window: **attributes** and **path:**



The **attributes** property should contain a SymbolExpression with a PLC variable which is an instance of LockView FBs from Tc3_MTP library: FB_MTP_LockView4, FB_MTP_LockView8, FB_MTP_LockView16 (depending on the quantity of required interlock reasons). The sample of such a record for the PLC variable PLC1.GVL_MTP.Lock1 (`Lock1:FB_MTP_LockView4`) is shown below:

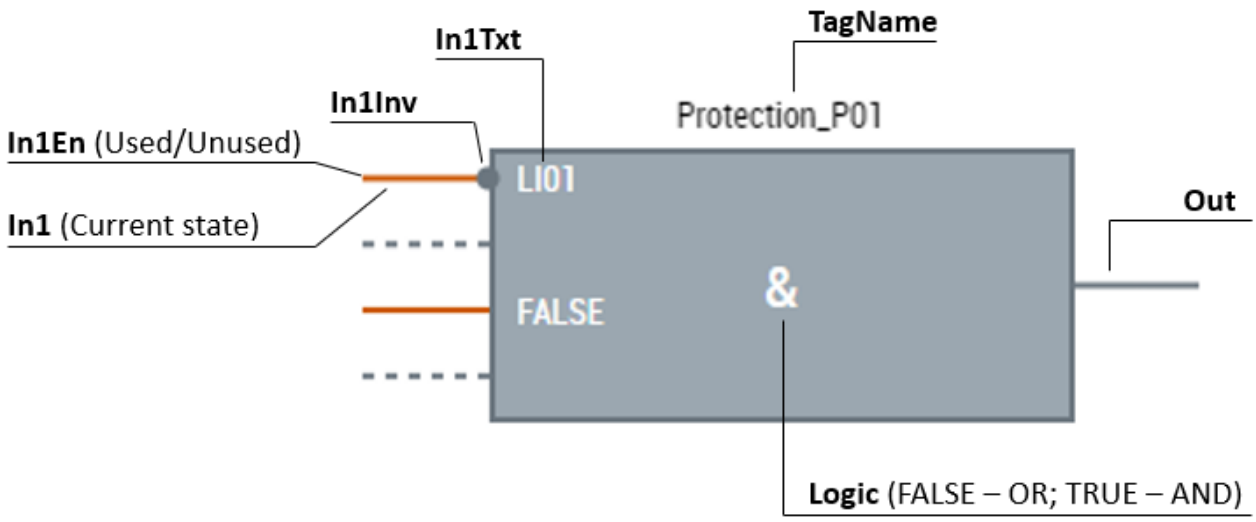


The **path** property should contain the LockView UserControl with the corresponding number of inputs:

- *Faceplates/LockView4_Interlocks.usercontrol*
- *Faceplates/LockView8_Interlocks.usercontrol*
- *Faceplates/LockView16_Interlocks.usercontrol*

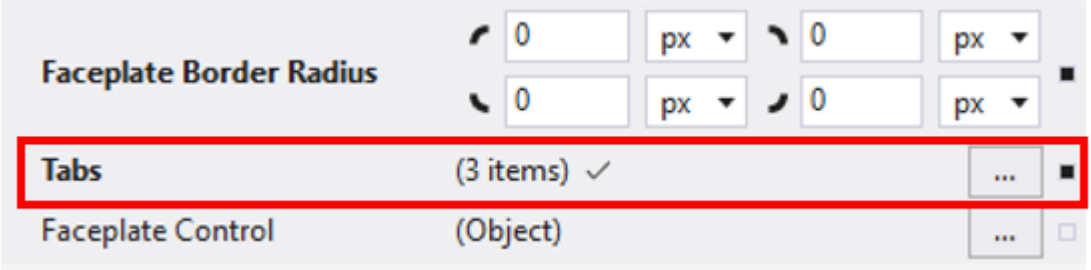
Properties and current states of the interlock control should be specified in the LockView FB instances of a PLC program. A sample of correspondence between the FB data and the control states is shown below (the involved variables of the FB are outlined in red):

Lock1	FB_MTP_LockView4	
TagName	STRING(255)	'Protection_P01'
TagDescription	STRING(255)	'Shows Interlock...
WQC	BYTE	0
Logic	BOOL	TRUE
In1En	BOOL	TRUE
In1	BOOL	TRUE
In1QC	BYTE	0
In1Inv	BOOL	TRUE
In1Txt	STRING(255)	'LI01'
Out	BOOL	FALSE

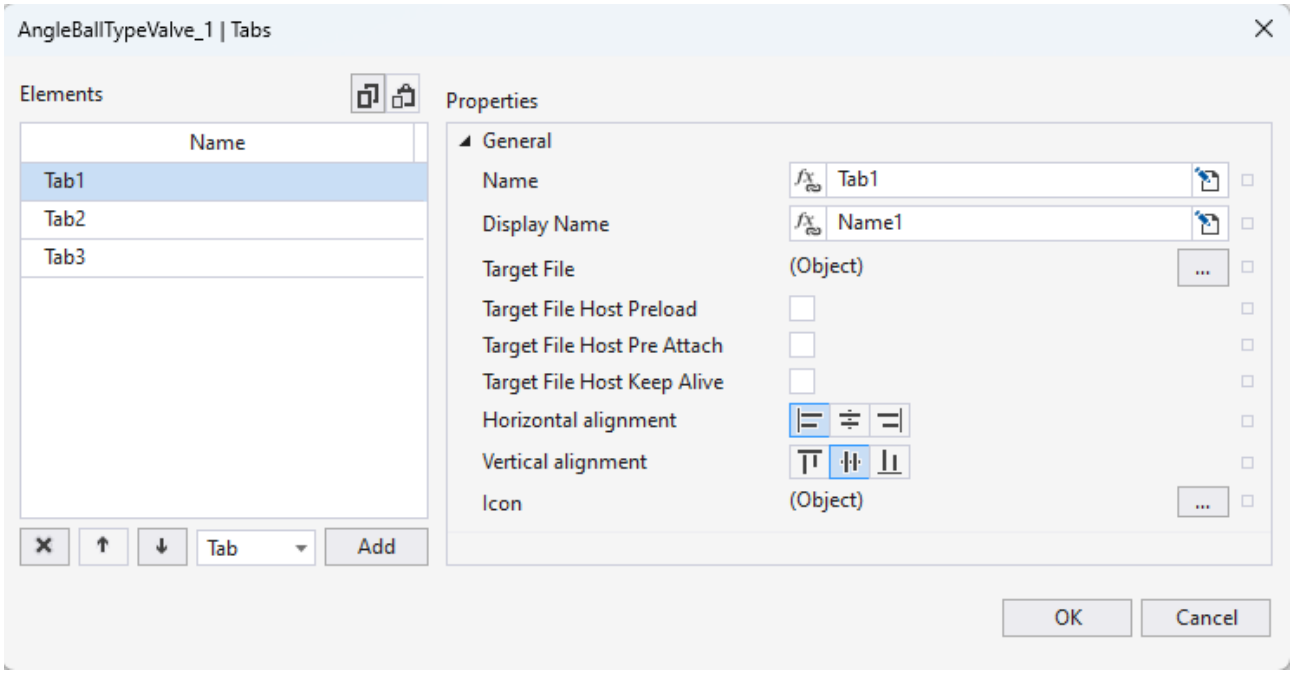


Tabs property

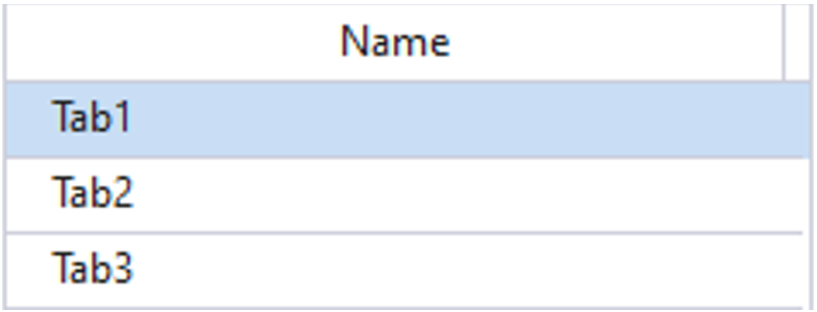
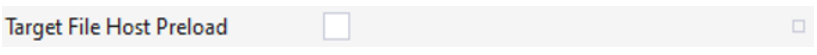
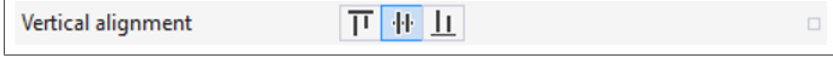
Tabs configuration window is available via "…" button.



After pushing the "…" button, the faceplate settings window opens:



The following table shows the description of the faceplate setting window elements tab:

Elements	Description
	Delete the selected tab
	Change the tab order (higher, lower)
	Type of object (only tab)
	Add a new tab
	List of current tabs. Tab1 is active (its properties are available for editing).
	Copy/paste current tabs.
	Internal tab name
	Tab name displayed on HMI
	Choose *.usercontrol or *.content element as a content for the tab
	Enable preload for the tab content *)
	Enable pre-attach for the tab content *)
	Enable keep alive for the tab content *)
	Horizontal alignment for the displayed tab name
	Vertical alignment for the displayed tab name

Elements	Description
Icon (Object) ...	Icon for the tab

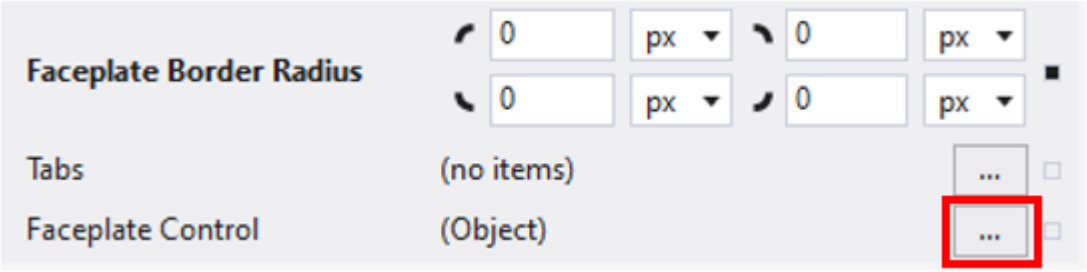
***)** – See the "Control life cycle" paragraph in the TE2000 TwinCAT 3 HMI manual for detailed information

Examples of how to configure the **Tabs** property for:

- ready-to-use Operate and Monitor tabs [▶ 58];
- ready-to-use Chart tab [▶ 67];
- ready-to-use Events tab [▶ 70].

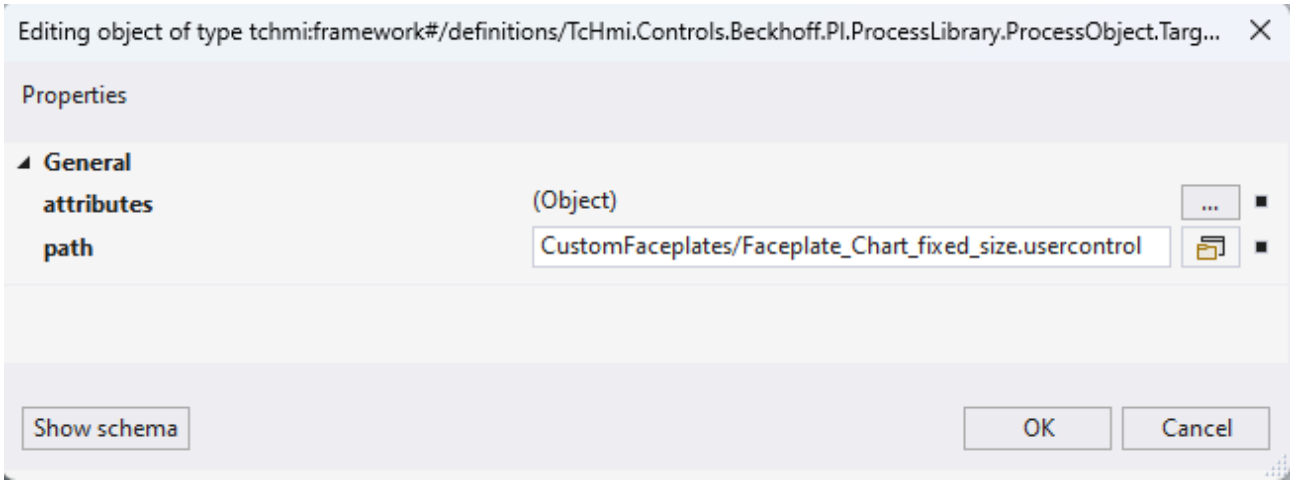
Faceplate Control property

Use the "**…**" button in the Control properties to open the window for **Faceplate Control** property configuration.

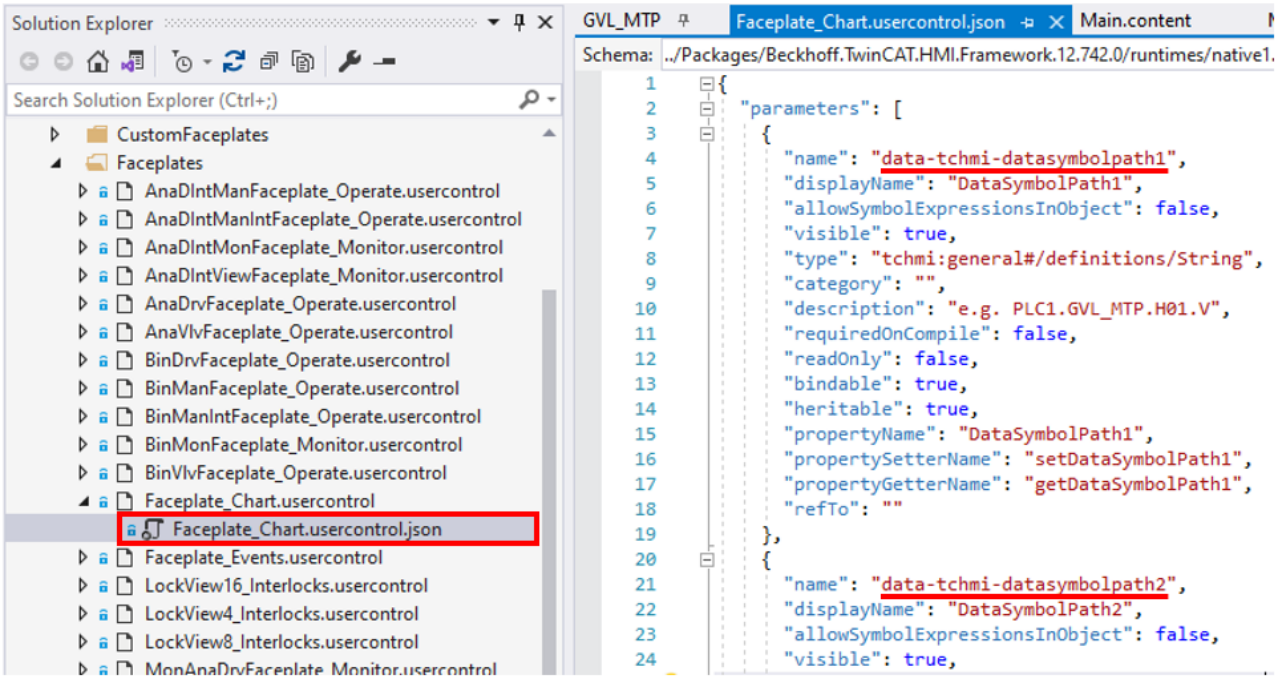


In the configuration window:

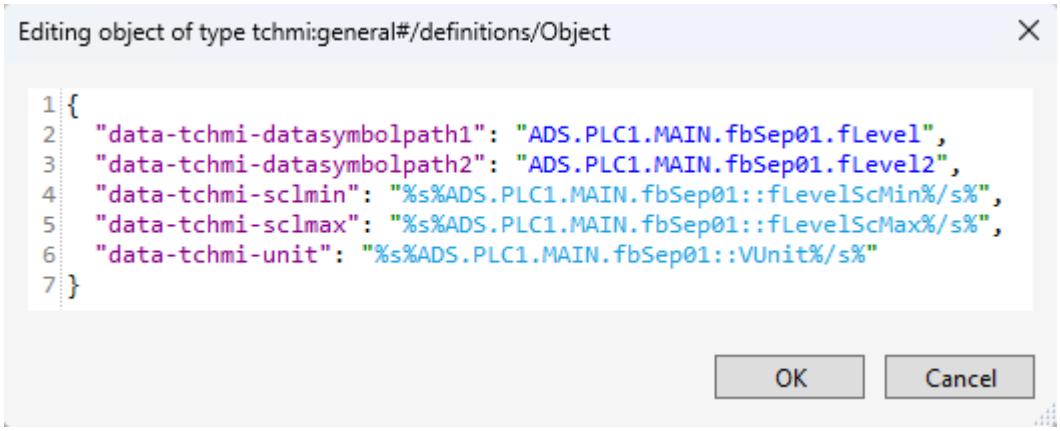
- property **attributes** contains a JS object with faceplate UserControl parameters
- property **path** refers to the UserControl which is intended to be used for the faceplate



Names of the faceplate parameters which are required for the **attributes** property can be taken from the **.usercontrol.json* file of the faceplate (**View Code** in context menu should be used to open the file in a text mode).



An example of how to configure the **attributes** property of **Faceplate Control** property to utilize *Faceplate_Chart.usercontrol* for the customized faceplate is shown below:

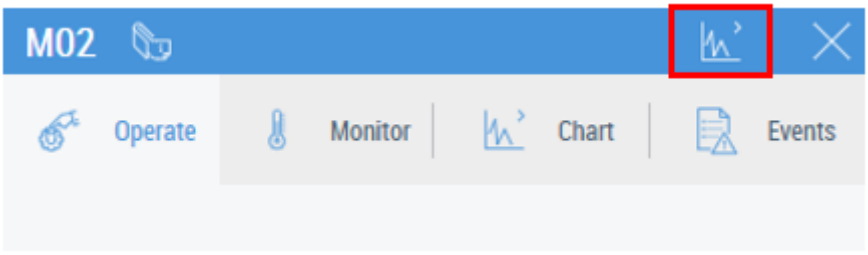


Also see about this

- Category: Faceplate [▶ 32]
- Category: Faceplate [▶ 34]
- Category: Faceplate [▶ 37]
- Chart tab [▶ 65]
- Events tab [▶ 69]
- Operate tab [▶ 56]
- Monitor tab [▶ 59]

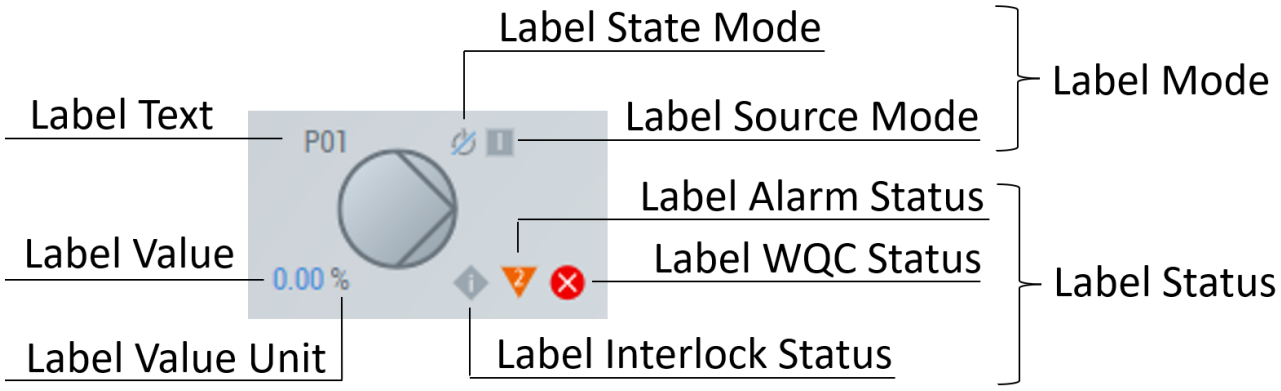
4.2.1.3 Category: Scope

The following table shows the Scope category properties:

Property	Description
Show Scope	Enable/disable to show **Scope** button on the control faceplate  which opens Scope view for the chosen configuration (1 – enable; 0 – disable). Detailed information on configuring Scope is given in the Scope configuration [▶ 24] chapter. If the **Show Faceplate** property is disabled, the Scope will be opened directly when clicking on the control.
Scope Config	The definition of the Scope config with config name and config chart name.
Server Interval	Time interval in milliseconds for data acquisition from the Scope server.
Enable Record Controls	Toggles whether the record controls should be displayed.
Show Trigger Window	Toggles whether the trigger window is visible or invisible.
Chart Color Source	Toggles whether the ScopeConfig or theme colors are used.
Trigger Window Position	Specifies where the trigger window is displayed. Top, bottom, right, or left.
Server Domain	The domain of the scope extension in the server. Defaults to 'TcHmiScope'.

4.2.1.4 Category: Label

An example of the control labels is shown below.



The following table shows the Label category properties

Property	Description
Show Label Text	Enable/disable label text (1 – enable; 0 – disable).
Label Text	Defines an optional label text instead of the one taken from the **Data Symbol** property.
Label Text Horizontal Alignment	Defines the horizontal alignment of the label text.
Label Text Vertical Alignment	Defines the vertical alignment of the label text.
Show Label Value	Enable/disable label value (1 – enable; 0 – disable).
Label Value	Defines an optional label value instead of the one taken from the **Data Symbol** property.
Label Value Unit	Defines an optional label value unit instead of the one taken from the **Data Symbol** property.
Label Value Format	A formatting function to format the label value (no HTML allowed).
Label Value Unit Format	A formatting function to format the label value unit (no HTML allowed).
Label Value Horizontal Alignment	Defines the horizontal alignment of the label value.
Label Value Vertical Alignment	Defines the vertical alignment of the label value.
Show Label Mode	Enable/disable label mode (1 – enable; 0 – disable).
Label State Mode	Defines an optional label state mode instead of the one taken from the **Data Symbol** property. State Modes [▶ 40].
Label Source Mode	Defines an optional label source mode instead of the one taken from the **Data Symbol** property. Source Modes [▶ 40].
Label Mode Horizontal Alignment	Defines the horizontal alignment of the label mode.
Label Mode Vertical Alignment	Defines the vertical alignment of the label mode.
Show Label Status	Enable/disable label status (1 – enable; 0 – disable).
Label Interlock Status	Defines an optional label interlock status for the one taken from the **Data Symbol** property. Interlock Statuses [▶ 41].
Label Interlock Message	Message for faceplate footer active lock element.
Label Alarm Status	Defines an optional label alarm status instead of the one taken from **the Data Symbol** property. Alarm Statuses [▶ 41].
Label Alarm Message	Message for faceplate footer status element.
Label WQC Status	Defines an optional label WQC status instead of the one taken from the **Data Symbol** property. WQC Statuses [▶ 41].
Label WQC Message	Message for faceplate footer WQC element.
Label Status Horizontal Alignment	Defines the horizontal alignment of the label status.
Label Status Vertical Alignment	Defines the vertical alignment of the label status.
Label Text Padding	The distance of the label text to the original position.
Label Value Padding	The distance of the label value to the original position.
Label Mode Padding	The distance of the label mode to the original position.
Label Status Padding	The distance of the label status to the original position.

In addition to label state mode (explanations are given for behavior according to the MTP standard, but a user can change the behavior by implementing custom FB):

Num	Icon	Meaning	Explanation
1		Offline	The control is blocked. No switching requests or value transfers are respected.
2		Operator	Manual control is activated. Manual (operator) switching requests are respected.
3		Automatic	Automatic control is activated. Internal (automatic) switching requests are respected.
Other	-	None	

In addition to label source mode (explanations are given for behavior according to the MTP standard, but a user can change the behavior by implementing custom FB):

Num	Icon	Meaning	Explanation
1		Internal	Control by internal parameters is activated. Internal control requests are made visible externally using automatic indicators.
2		Manual	Control by parameters being set by an operator is activated.
Other	-	None	

In addition to label interlock status (explanations are given for behavior according to the MTP standard, but a user can change the behavior by implementing custom FB):

Num	Icon	Meaning	Explanation
1		Permit	Implements a switch-on lock. This prevents activation of DataAssembly when in the locked state. However, an already activated DataAssembly in the locked state is not set to the safe position.
2		Interlock	Implements a lock that both prevents switching on and puts a DataAssembly in its safe position. An active interlock does not need to be reset.
3		Protect	Implements a lock like interlock, but with the difference that the protection must be reset.
Other	-	None	

In addition to label alarm status (explanations are given for behavior according to the MTP standard, but a user can change the behavior by implementing custom FB):

Num	Icon	Meaning	Explanation
1		Tolerance	Out of tolerance limit.
2		Warning	Out of warning limit.
3		Alarm	Out of alarm limit.
Other	-	None	

In addition to label WQC status (status names are given according to the NAMUR recommendations (NE 184), but a user can change the behavior by implementing custom FB):

Num	Icon	Status
0		Failure
96		Function Check
164		Maintenance Request
104		Out of Specification
28		Out of Service
Other	-	None

4.2.2 Functions

The ProcessObject adds some functions which are available for users in the same way as other functions.

4.2.2.1 Category: Common

Functions from the Common category are designated for working with the faceplate pop-up. They are shown in the following table:

Function	Description
open	Open the faceplate.
close	Close the faceplate.
resetBounds	Resets the size and position of the pop-up and clears that data from the local storage.

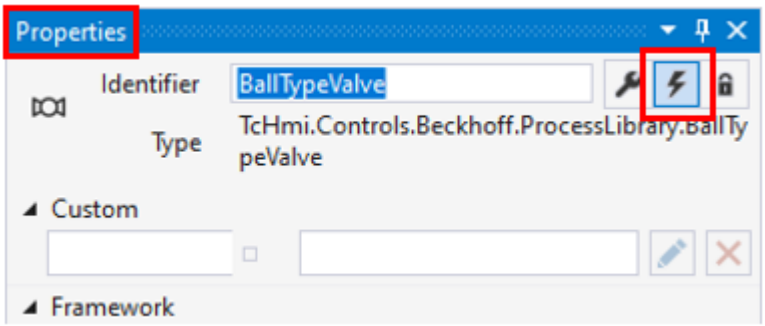
4.2.2.2 Category: Scope

Functions from the Scope category help working with Scope view functionality. They are shown in the following table:

Function	Description
openScope	Open the Scope.
closeScope	Close the Scope.
startRecord	Begin the record on the TwinCAT 3 Scope Server.
stopRecord	End the record on the TwinCAT 3 Scope Server.
startDisplay	Start to get the live data from scope record.
stopDisplay	Stop to get the live data from scope record.
zoomToDefault	Zoom to the default display width.
zoomOutMax	Zoom to the highest display width at this time.
goTo	Go to the valueNew start time. Parameters: • valueNew: the new start time. • type: *String* • format: d,HH:mm:ss,fff:uuu (no break spaces) , where: • d – days; • HH – hours; • mm – minutes; • ss – seconds; • fff – milliseconds; • uuu – microseconds.
setDisplayWidth	Sets a valueNew for the display width. Parameters: • valueNew : the new display width. • type: *String* • format: d,HH:mm:ss,fff:uuu (no break spaces) , where: • d – days; • HH – hours; • mm – minutes; • ss – seconds; • fff – milliseconds; • uuu – microseconds.
scroll	Scroll forward (or reverse). Parameters: • valueNew : the direction of scrolling • Type: *TcHmi.Controls.Beckhoff.TcHmiScopeControl.ScrollDirection.*
scrollBig	Scroll big: (a full page) forward (or reverse) for the Scope Control. Parameters: • valueNew : the direction of scrolling • type: *TcHmi.Controls.Beckhoff.TcHmiScopeControl.ScrollDirection.*
undo	Undo the last interaction.
redo	Redo the last interaction.
setOverviewMode	Toggle the overview.
setMouseMode	Set the mouse mode for interactions. Parameters: • valueNew : the new mouseMode • type: *TcHmi.Controls.Beckhoff.TcHmiScopeControl.MouseMode.*
getMouseMode	get the mouse mode

4.2.3 Events

Events of the controls are available if the button **Show Events** of the **Properties** window is pushed. The ProcessObject adds some events (see table below) in a control category.



The following table shows ProcessObject events

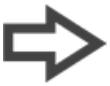
Event	Description
.onResetPressed	The onResetPressed event is fired after a click (mouse) or tap (touch) event.
.onOpened	The onOpened event is raised after the faceplate is opened.
.onClosed	The onClosed event is raised after the faceplate is closed.
.onScopeOpened	The onScopeOpened event is raised after the scope is opened.
.onScopeClosed	The onScopeClosed event is raised after the scope is closed.
.onResetStateChanged	The onResetStateChanged event is raised when the state of the reset button has changed.
.onResetStatePressed	The onResetStatePressed event is raised when the state of the reset button has changed to true.
.onResetStateReleased	The onResetStateReleased event is raised when the state of the reset button has changed to false.
.onValueChanged	The onValueChanged event is raised when the LabelValue of the control has changed.

4.3 Controls

Registration numbers of the controls' graphical symbols which are mentioned in their description are taken from the DIN EN ISO 10628-2 standard.

4.3.1 General controls

The following table lists general controls and their descriptions.

Control	Icon	Description
Service		Control that can be linked to a service.
Lock View		Displaying lock views up to 16 inputs. The functionality is described in more detail in the Interlocks tab [▶ 71] chapter.
PID Controller		A view of PID controller parameters: PV (process variable), SP (setpoint), MV (manipulation variable).
Value View		Displays process values in various formats with engineering units.
Directed Termination		Termination of pipe with indication of direction. Can be used to move to another view when pressed.
Termination		Termination of pipe. Can be used to move to another view when pressed.

4.3.1.1 Lock View

Lock view is derived from the *TcHmi.Controls.System.TcHmiControl* class, thus all inherited properties can be viewed in the documentation for this class. All properties specific to the Lock view that are available for users are described in the table below:

Property	Category	Description
Text Color	Colors	Text color inside the control block.
Active Color	Colors	Fill color of the control block when the lock is active.
Lock View Symbol	Common	An instance of Lock View PLC FB with 4, 8 or 16 inputs to be bound to the control.
Logic	Common	Logical unit operation which is carried out on the control inputs' values (0 – OR; 1 – AND).
Out	Common	Output value of the control (0 – the lock is inactive; 1 – the lock is active).
Show Input [N] ***)**	Common	Show the input N (1 – show; 0 – hide).
Input [N] Enabled	Common	Enable/disable the input N (1 – enable; 0 – disable).
Input [N]	Common	Current value at the input N (1 – enable; 0 – disable).
Input [N] Inverted	Common	Invert the value from the input N (1 – invert; 0 – don't invert).
Input [N] Text	Common	Text label at the input N.
Show Label Text	Label	Enable/disable Label Text (1 – enable; 0 – disable).
Label Text	Label	Defines an optional label text instead of the tag name taken from the Lock View Symbol.

***)** [N] – number of the inputs from 4, 8 or 16

4.3.1.2 PID Controller

There are no specific properties, functions, or events for the PID controller control, but special formatting is used for the **Label Value** and **Label Value Unit** properties which are to be split up into the three values (PV, SP, and MV):

The values/units should be separated by semicolon:

```
PV value/unit; SP value/unit; MV value/unit
```

4.3.1.3 Value View

There are no specific properties, functions, or events for value view control.

4.3.1.4 Terminations

Element baseTermination inherits ProcessObject and adds new properties, which are listed in the following table. In turn, controls Termination and Directed Termination inherit properties and behavior from baseTermination and don't add anything new.

The following table shows the baseTermination properties:

Property	Category	Description
Target Region	Common	The target region control, where content will be displayed when the event line control is clicked or touched. Type: *TcHmi.Controls.System.TcHmiRegion.*
Target Content	Common	Path to the target content page which will be displayed when the event line control is clicked or touched. Type: *ContentPath.*

4.3.2 Agitators

The following table lists agitator controls and their descriptions.

Control	Icon	Description
Anchor Agitator		Agitator, anchor type (ISO C2022).
Cross-Beam Agitator		Agitator, cross-beam type (ISO C2021).
Disk Agitator		Agitator, disc type (ISO C2026).
Flate-Blade Paddle Agitator		Agitator, flate-blade paddle type (ISO C2019).
Gat Paddle Agitator		Agitator, gat paddle type (ISO C2020).
General Agitator		Agitator (general), stirrer (general) (ISO 2672).
Helical Agitator		Agitator, helical type (ISO C2023).
Impeller Agitator		Agitator, impeller type (ISO C2024).
Propeller Agitator		Agitator, propeller type (ISO C2025).
Turbine Agitator		Agitator, turbine type (ISO C2027).

All additional properties of agitator controls, compared to ProcessObject, are implemented in the baseAgitator element. Other agitator controls inherit these properties and have no additional ones.

4.3.2.1 baseAgitator

Element baseAgitator adds a Rotation category with some properties. See the following table:

Property	Description
Show Rotation	Enable/disable rotation (1 – enable; 0 – disable)
Rotation Duration	Define rotation duration in milliseconds.

Rotation animation

If **Show Rotation** is enabled, the movement and speed of animation depend on other properties.

Whether the agitator should move or not is defined by **Data Symbol** or **Label Value** properties. **Label Value** has priority. If the value equals 0, there is no movement. Otherwise, the agitator should move.

The animation speed fully depends on the **Rotation Duration** property.

Graphic Status property

Graphic Status property is unavailable for the Agitator controls.

4.3.3 Blowers and Fans

The following table lists blower and fan controls and their descriptions.

Control	Icon	Description
General Blower		Blower, fan (general) (ISO X8164).

Element baseBlower doesn't add any other properties, functions or events compared to ProcessObject. Other Blower and Fan controls don't add anything else to baseBlower either.

4.3.4 Columns

The following table lists column controls and their descriptions.

Control	Icon	Description
General Column		Column (general) (ISO X8100).
General Tray Column		Tray column (general) (ISO X8101).

Element baseColumn is inherited from baseTank [▶ 53] and doesn't add any other properties, functions, or events compared to it. Other column controls don't add anything else to baseColumn either.

4.3.5 Compressors and Vacuum Pumps

The following table lists compressor and vacuum pump controls and their descriptions.

Control	Icon	Description
Centrifugal Compressor		Compressor, centrifugal type (ISO X8179).
Diaphragm Compressor		Compressor, vacuum pump diaphragm type (ISO X8099).
General Compressor		Compressor, vacuum pump (general) (ISO 2302).
Jet Compressor		Compressor, ejector type, vacuum pump jet type (ISO X8163).
Liquid Ring Compressor		Compressor, vacuum pump liquid ring type (ISO X8162).
Reciprocating Piston Compressor		Compressor, vacuum pump, reciprocating piston type (ISO X8097).
Roller Vane Compressor		Compressor, vacuum pump, roller vane type, compressor (ISO X8104).
Rotary Compressor		Compressor rotary type, vacuum pump, rotary piston type (ISO X8105).
Screw Compressor		Compressor rotary type, vacuum pump, rotary piston type (ISO X8161).
Turbo Compressor		Compressor, vacuum pump, turbo type (ISO X8102).

Element baseCompressor doesn't add any other properties, functions, or events compared to ProcessObject. Other compressor and vacuum pump controls don't add anything else to baseCompressor either.

4.3.6 Conveyors

The following table lists conveyor controls and their descriptions.

Control	Icon	Description
Screw Conveyor		Conveyor, screw type, closed (ISO X8063).

All additional properties of conveyor controls, compared to ProcessObject, are implemented in the baseConveyor element. Other conveyor controls inherit these properties and have no additional ones.

4.3.6.1 baseConveyor

Element baseConveyor adds a rotation category with some properties. See the following table:

Property	Description
Show Rotation	Enable/disable rotation (1 – enable; 0 – disable)
Rotation Duration	Define rotation duration in milliseconds.

Rotation animation

If **Show Rotation** is enabled, the movement and speed of animation depend on other properties.

Whether the conveyor should move or not is defined by DataSymbol or **Label Value** property. **Label Value** has priority. If the value equals 0, there is no movement. Otherwise, the conveyor should move.

The animation speed fully depends on the **Rotation Duration** property.

4.3.7 Drives

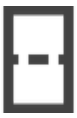
The following table lists drive controls and their descriptions.

Control	Icon	Description
AC Generator		Generator, AC (ISO X8154).
AC Motor		Electric motor, AC (ISO X8157).
DC Generator		Generator, DC (ISO X8155).
DC Motor		Electric motor, DC (ISO X8158).
Gear		Gear (ISO X8167).
General Generator		Generator (general) (ISO C0079).
General Motor		Electric motor (general) (ISO C0082).
General Turbine		Turbine (general) (ISO 2571).

Element baseDrive doesn't add any other properties, functions, or events compared to ProcessObject. Other Drive controls don't add anything else to baseDrive either.

4.3.8 Filters

The following table lists Filter controls and their descriptions.

Control	Icon	Description
Bag Liquid Filter		Liquid filter, bag, candle or cartridge type (ISO X8117).
General Liquid Filter		Liquid filter (general) (ISO X8116).

Element baseFilter doesn't add any other properties, functions, or events compared to ProcessObject. Other filter controls don't add anything else to baseFilter either.

4.3.9 Heating and Cooling

The following table lists heating and cooling controls and their descriptions.

Control	Icon	Description
Electrical Heater for Vessel		Electrical Heater for Vessel with dished ends and electrical heating (ISO X2070). It is expected to be combined with vessels.
General Heat Exchanger		Heat exchanger (general), condenser (ISO X8079).
Heating/Cooling Jacket for Vessel		Heating/Cooling Jacket for Vessel with dished ends and heating/cooling jacket (ISO X2069). It is expected to be combined with vessels.
Heating/Cooling Jacket for Vessel Dished		Heating/Cooling Jacket (dished bottom) for Vessel with dished ends and heating/cooling jacket (ISO X2069). It is expected to be combined with vessels.
M-Shape Heat Exchanger		Heat exchanger (M-shape) (ISO X8017).
Straight Tubes Heat Exchanger		Heat exchanger with straight tubes (fixed-tube plates) (ISO 2511).

Element baseHeatExchanger doesn't add any other properties, functions or events compared to ProcessObject. Heating and cooling controls don't add anything else to baseHeatExchanger either.

4.3.10 Liquid Pumps

The following table lists liquid pump controls and their descriptions.

Control	Icon	Description
Centrifugal Pump		Pump, centrifugal type (ISO 2322).
Diaphragm Pump		Pump, diaphragm type (ISO X8095).
Gear Pump		Pump, gear type (ISO X8091).
Jet Pump		Jet pump, liquid type ejector pump (ISO X8096).
Liquid Pump		Pump, liquid type (general) (ISO 2301).
Progressive Cavity Pump		Pump, progressive cavity type (ISO X8093).
Reciprocating Piston Pump		Pump, reciprocating piston type (ISO X8094).
Screw Pump		Pump, screw type (ISO X8092).

Element baseLiquidPump doesn't add any other properties, functions, or events compared to ProcessObject. Other liquid pump controls don't add anything else to baseLiquidPump either.

4.3.11 Vessels and Tanks

The following table lists vessel and tank controls and their descriptions.

Control	Icon	Description
Container		Container, tank, cistern (ISO 2061).
General Tank		Tank, vessel (ISO 301).
Spherical Vessel		Spherical vessel (ISO 2063).
Spherical Vessel On Legs		Spherical vessel on legs (ISO X8010).
Tank Dished Ends		Tank, vessel with dished ends (ISO 2062).
Vessel Dished Ends On Legs		Vessel with dished ends and support legs (ISO X8002).
Vessel Dished Roof Conical Bottom		Vessel with dished roof and conical bottom (ISO X8008).

All additional properties of vessel and tank controls, compared to ProcessObject, are implemented in baseTank element. Vessel and tank controls inherit these properties and have no additional ones.

4.3.11.1 baseTank

Element baseTank adds a fill level category with some properties and **Fill Level Color** property to the colors category. See following table:

Property	Category	Description
Fill Level Color	Colors	The color of displayed fill level.
Show Fill Level	Fill Level	Enable/disable fill level (1 – enable; 0 – disable).
Fill Level Scale Min	Fill Level	Sets the fill level minimum for scaling the level animation.
Fill Level Scale Max	Fill Level	Sets the fill level maximum for scaling the level animation.

Fill level animation

If **Show Fill Level** is enabled, the fill level in the vessel is animated according to the DataSymbol or **Label Value** property. The **Label Value** property takes priority if it is set. For scaling, the appropriate data from DataSymbol or, in priority order, the **Fill Level Scale Min** and **Fill Level Scale Max** properties are implemented, but only if both of them are set.

4.3.12 Valves

The following table lists valve controls and their descriptions.

Control	Icon	Description
Angle Ball Type Valve		Valve, angle ball type (ISO X8072).
Angle Globe Type Safety Valve		Safety valve, spring loaded, globe angle type (ISO X2125).
Angle Globe Type Valve		Valve, angle globe type (ISO X8069).
Angle Type Valve		Valve, angle type (general) (ISO 2102).
Ball Type Valve		Valve, ball type (ISO X8071).
Butterfly Type Form1 Valve		Valve, butterfly type (Form 1) (ISO X8075).
Diaphragm Valve		Diaphragm Valve
Control Type Valve		Valve, control type, continuously operated (ISO X8087).
Four Way Valve		Four way valve (general).
Gate Type Valve		Valve, gate type (ISO X8074).
General Valve		Valve (general) (ISO 2101).
Globe Type Safety Valve		Safety valve, spring loaded, globe type (ISO X2124).
Globe Type Valve		Valve, globe type (ISO X8068).
Needle Type Valve		Valve, needle type (ISO X8076).
Three Way Ball Type Valve		Valve, three way ball type (ISO X8073).
Three Way Globe Type Valve		Valve, three way globe type (ISO X8070).

Control	Icon	Description
Three Way Type Valve		Valve, three way type (general) (ISO 2103).

Element baseValve doesn't add any other properties, functions, or events compared to ProcessObject. Elements baseThreeWayTypeValve and baseFourWayValve add the **Valve Position** property in the category Common. Thus, Three Way Type Valve and Four Way Valve controls inherit this behavior from their base element, but nevertheless, **Graphic Status** property has priority over the **Valve Position** one. **Valve Position** property is handled in different ways for three and four way valves.

The following table shows the correspondence between **Valve Position** property and appearance of three way valve control:

Valve Position value	0	1	2	3	4	5	6	7	Any other
Appearance									

The following table shows the correspondence between **Valve Position** property and appearance of four way valve control:

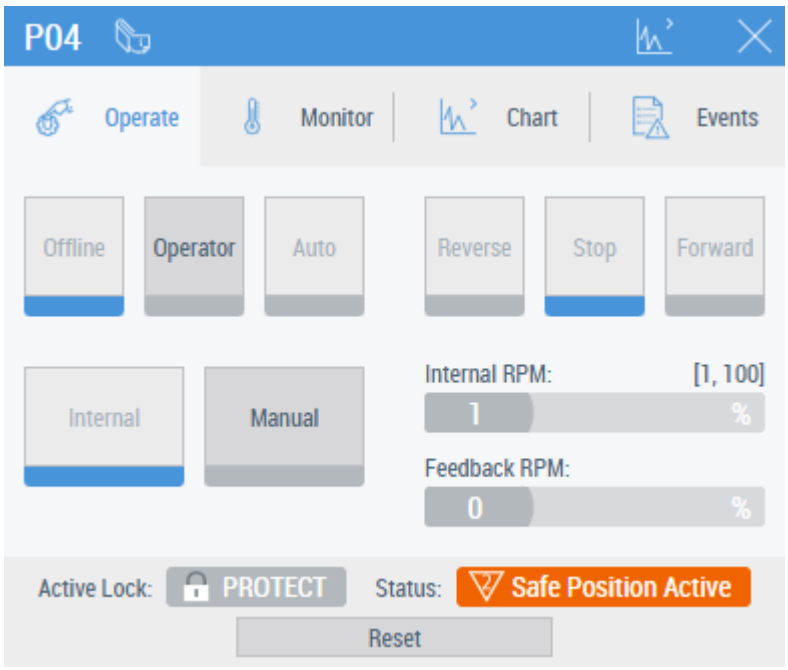
Valve Position value	0	1	2	3	4	5	6	7
Appearance								

Valve Position value	8	9	10	11	12	13	14	15
Appearance								

Valve Position value	16	17	18	19	20	21	Any other	
Appearance								

4.4 Ready-to-use faceplate elements

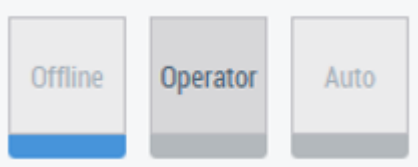
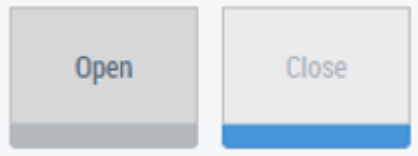
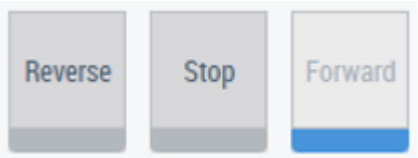
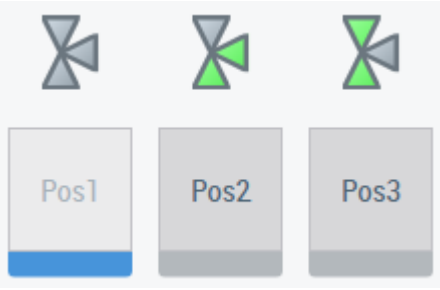
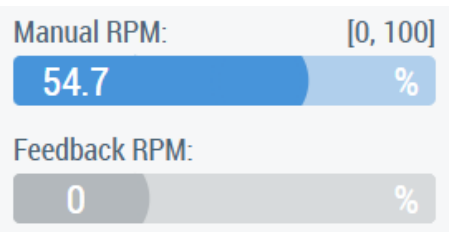
This is a sample of a ready-to-use faceplate

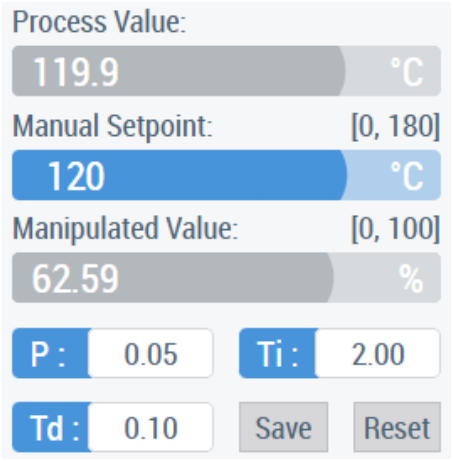


4.4.1 Operate tab

The set of elements on the **Operate** tab depends on the MTP type of the symbol bound to the **Data Symbol** property.

The **Operate** tab of the ready-to-use faceplate has repeating groups of elements for different MTP types. The buttons have an indicator below which shows the corresponding Act or Fbk status (gray = 0, blue = 1).The following table gives a description of the elements.

#	Group of elements (example)	Description
1		Group **State** (according to the MTP Specification Part 3). See State Modes [▶ 40]. The current state of the DataAssembly is OFFLINE. Button **Operator** is shown as enabled for pushing, other buttons are disabled. The reason for this is that, according to the MTP Specification Part 3, possible transitions are: Offline → Operator Operator → Offline Operator → Automatic Automatic → Operator
2		Group **Source** (according to the MTP Specification Part 3). See Source Modes [▶ 40]. The current source of the DataAssembly is MANUAL. The button **Internal** is enabled for pushing. The button **Manual** is disabled because the current source is MANUAL.
3		Open/Close **Switching** group (can be switch ON / switch OFF, etc.). The current state is CLOSED. The button **Open** is enabled for pushing. The button **Close** is disabled because the valve is already closed. Whether the group is active or not depends on the current state of the **State** group (#1 in this table): If the **State** group isn't present: always active. If the **State** group is present: active only when the state is OPERATOR.
4		Reverse/Stop/Forward **Switching** group. The current state is FORWARD. **Forward** button is disabled, **Stop** and **Reverse** buttons are enabled for pushing. Depending on FwdEn/RevEn, the corresponding buttons can be hidden. The group is only active when the current state of the **State** group (#1 in this table) is OPERATOR.
5		Pos1/Pos2/Pos3 **Switching** group for TriPos Valves. Graphical elements on top show the parts which should be active for every position. The current state is POSITION 1. Button **Pos1** is disabled, buttons **Pos2** and **Pos3** are enabled for pushing. The group is active only when the current state of the **State** group (#1 in this table) is OPERATOR.
6		**Value** group consists of a control which allows a value to be set manually and a control which gives feedback on the actual current value. The name, scale, and engineering units of the value can be different. Whether the group is allowed to set the value or not depends on the current state of the **Source** group (#2 in this table): If the **Source** group isn't present: always allowed. If the **Source** group is present: active only when the state is MANUAL. If the **Source** group state is INTERNAL, setting value control just shows the value which is given internally.

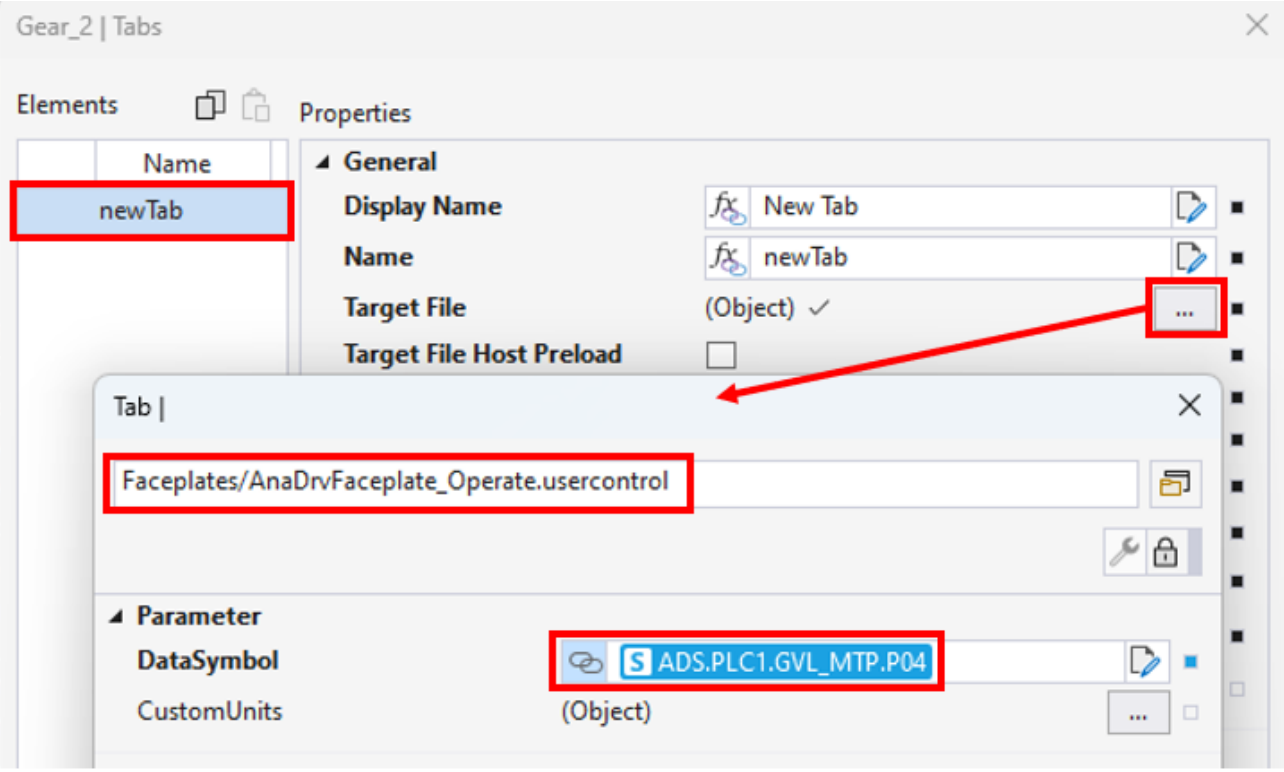
#	Group of elements (example)	Description
7		**PID Controller** group consists of such elements: • **Process Value** is the current measured value of a particular part of a process. It's always read-only. • **Setpoint** sets the desired value for the **Process Value**. Behavior of the **Setpoint** depending on the states of the **State** and **Source** groups is described in a table [▶ 58]. • **Manipulated Value** is an output of the PID controller, which influences the **Process Value** directly. Behavior of the **Manipulated Value** depending on the states of the **State** and **Source** groups is described in a table [▶ 58]. • **P**, **Ti**, and **Td** are proportional, integral, and derivative terms, respectively. • **Save** button saves **P**, **Ti**, and **Td** to the PLC program (not persistent memory) and to the web client memory which is reset on page reload. **Reset** button returns previous values for **P**, **Ti**, and **Td**, which are stored in the web client memory.

The following table shows the description of **PID Controller** group elements' behavior

• State of the **State** group, #1 in the table [▶ 57]	• State of the **Source** group, #2 in the table [▶ 57]	**Setpoint** behavior	**Manipulated value** behavior
OFFLINE	INTERNAL	Read only	Changeable, without taking effect
OFFLINE	MANUAL	Changeable, without taking effect	Changeable, without taking effect
OPERATOR	INTERNAL	Read only	Changeable, takes effect on the **Manipulated value**
OPERATOR	MANUAL	Changeable, without taking effect	Changeable, takes effect on the **Manipulated value**
AUTOMATIC	INTERNAL	Read only, takes effect on the **Manipulated value**	Read only, calculated based on **Setpoint**
AUTOMATIC	MANUAL	Changeable, takes effect on the **Manipulated value**	Read only, calculated based on **Setpoint**

Reusing the Operate tab with the Tabs property

Operate ready-to-use faceplates can be reused in the Tabs property [▶ 34] without any changes if they work with a FB instance which has the same set of variables (same names and same types) as that defined for the **DataSymbol** parameter of the faceplate. Below is an example of the settings for the AnaDrv MTP type **DataSymbol** parameter for *AnaDrvFaceplate_Operate.usercontrol*:



Also see about this

- Category: Label [▶ 40]
- Category: Label [▶ 40]

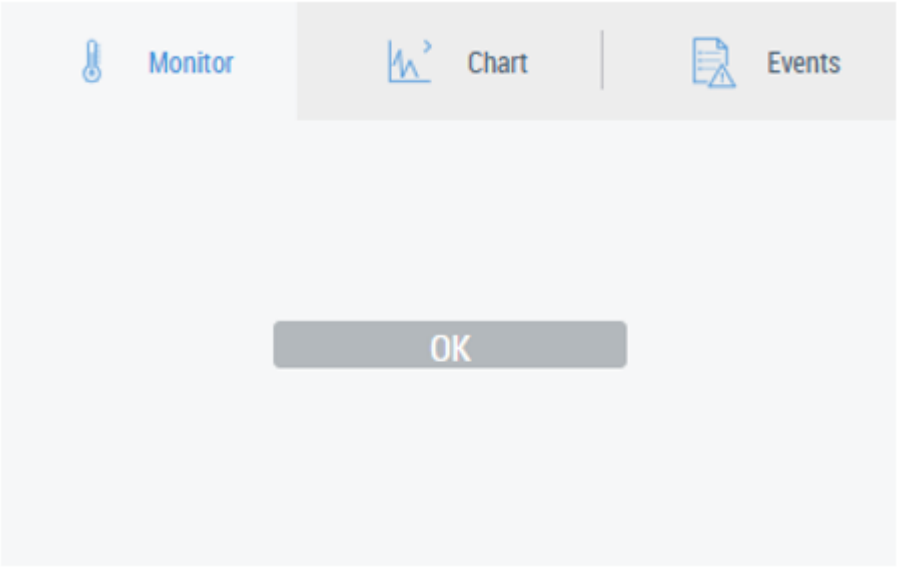
4.4.2 Monitor tab

Depending on the MTP type of the symbol bound to the **Data Symbol** property, the **Monitor** tab can be shown automatically on the faceplate. Its view is implemented with different faceplate UserControls depending on the MTP type.

Monitor tabs can be reused with the Tabs property [▶ 34] the same way as the Operate tabs [▶ 58].

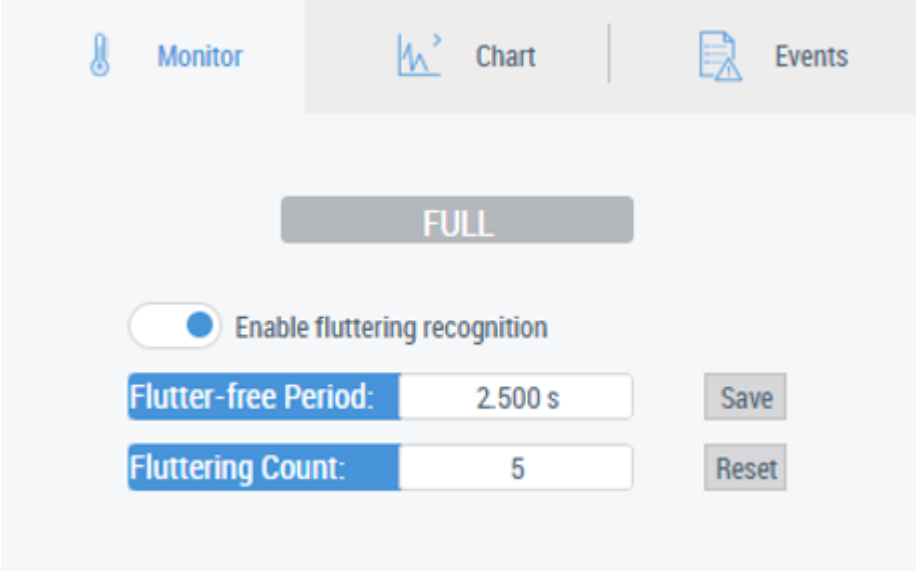
4.4.2.1 BinViewFaceplate

BinViewFaceplate only shows the current value according to VState0 and VState1:



4.4.2.2 BinMonFaceplate

An example of BinMonFaceplate:

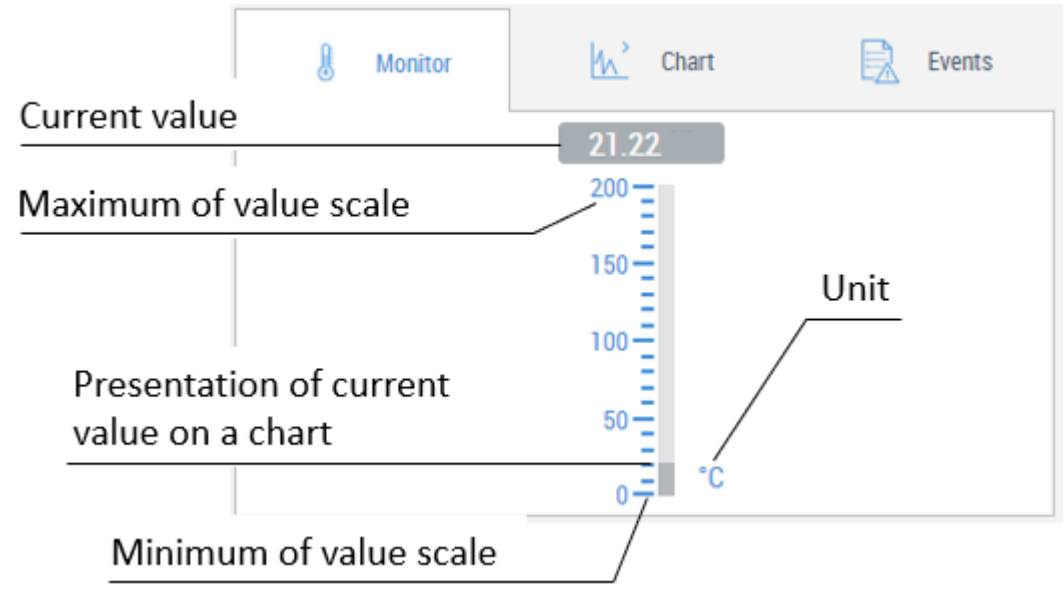


There is a description of the elements in the following table.

#	Element	Description
1	Value	Processed value.
2	Enable fluttering recognition	Switch recognition of signal fluttering on/off.
3	Flutter-free Period	Period of an active signal before it can be recognized as flutter-free.
4	Fluttering Count	Counts of the allowed fluttering signals in the defined period **Flutter-free Period** (#3 in this table).
5	Save button	**Save** button saves **Flutter-free Period** and **Fluttering Count** to the PLC program (not persistent memory) and to the web client memory which is reset upon page reload.
6	Reset button	**Reset** button returns previous values for **Flutter-free Period** and **Fluttering Count**, which are stored in the web client memory.

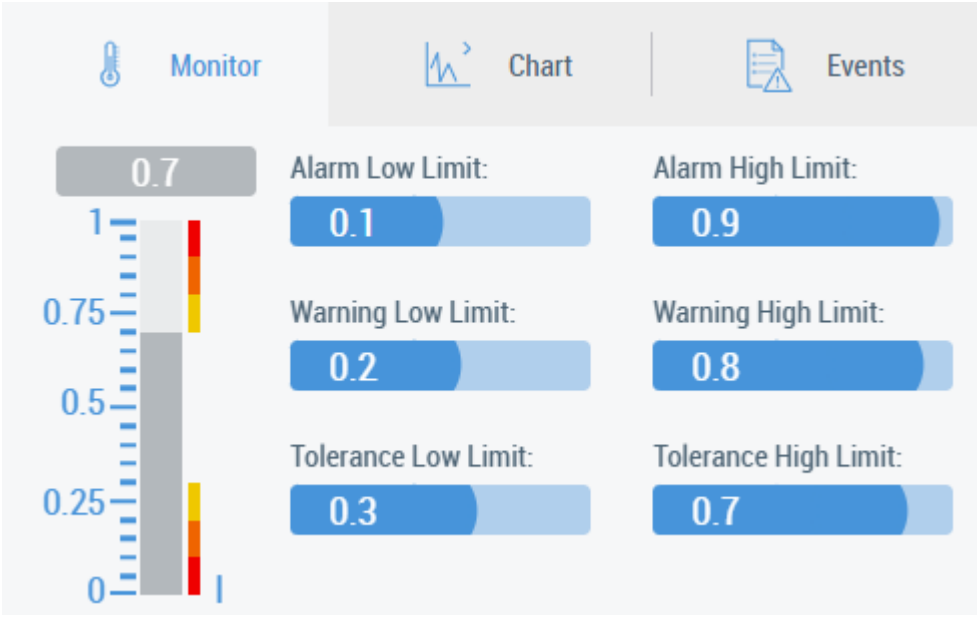
4.4.2.3 AnaViewFaceplate and DIntViewFaceplate

AnaViewFaceplate and DIntViewFaceplate look the same, but DInt one shows the integer value.

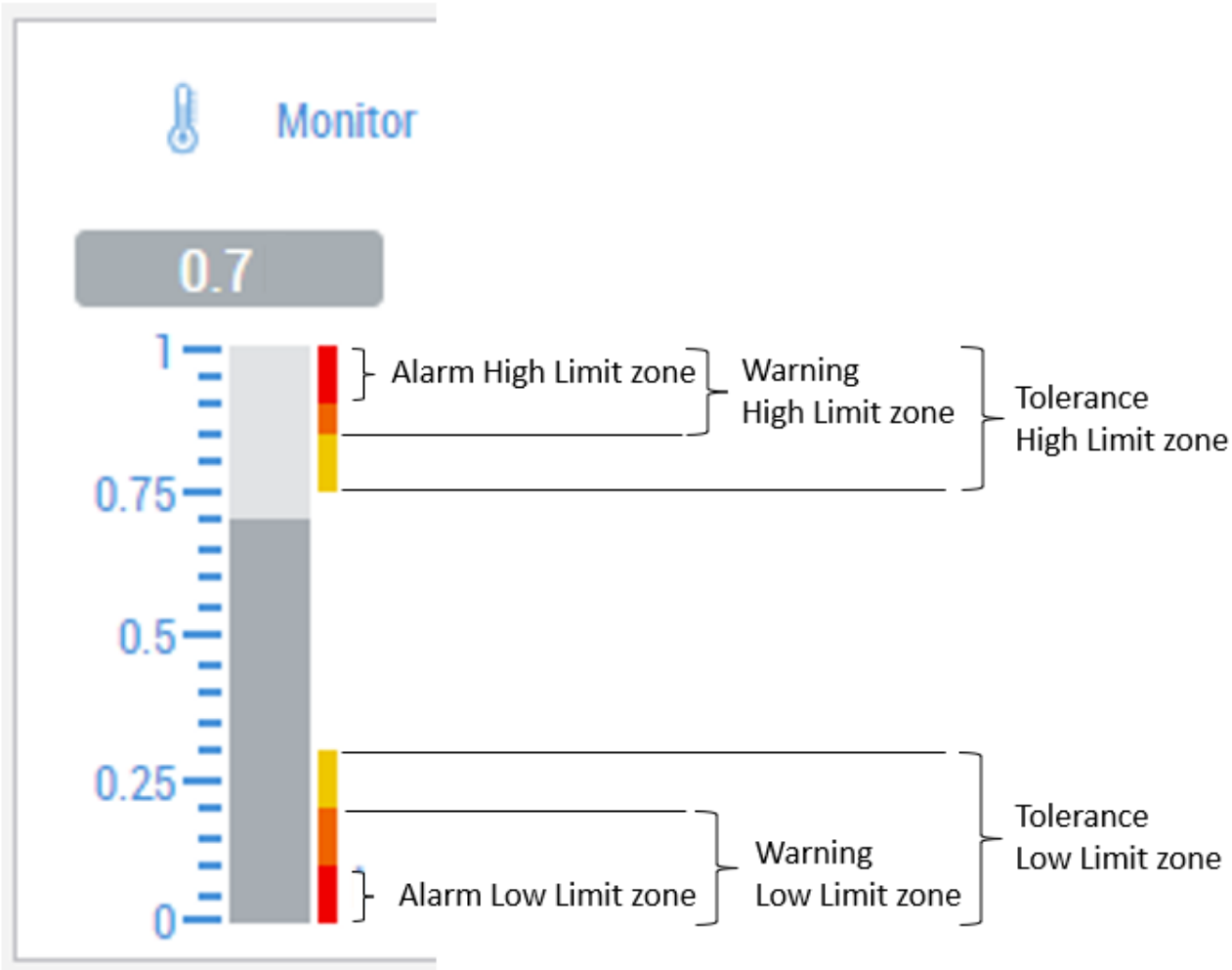


4.4.2.4 AnaMonFaceplate and DIntMonFaceplate

AnaMonFaceplate and DIntMonFaceplate look the same, but DInt one shows integer value and sets limits which also are integer values.



Elements for alarm, warning, and tolerance limits serve to show and change the limits of the value. The figure below shows zones on the chart which correspond to the limit settings. Other elements of the chart are explained for AnaViewFaceplate and DIntViewFaceplate [▶ 60].



4.4.2.5 MonBinDrvFaceplate and MonBinVlvFaceplate

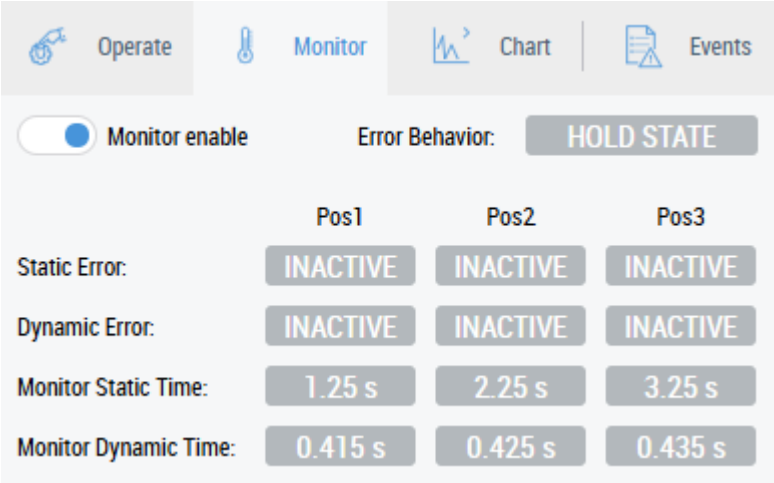
MonBinDrvFaceplate and MonBinVlvFaceplate have the same content. An example of the faceplate is shown below.

The following table shows the description of MonBinDrvFaceplate and MonBinVlvFaceplate elements:

#	Element	Description
1	Monitor enable	Enable monitoring
2	Static Error	Static error indicator
3	Dynamic Error	Dynamic error indicator
4	Monitor Static Time	Time parameter (MonStatTi) for the static error
5	Monitor Dynamic Time	Time parameter (MonDynTi) for the dynamic error
6	Error Behavior	Behavior in the event of an error (hold state or go to SafePos)

4.4.2.6 MonTriPosVlvFaceplate

A view of MonTriPosVlvFaceplate is shown below.

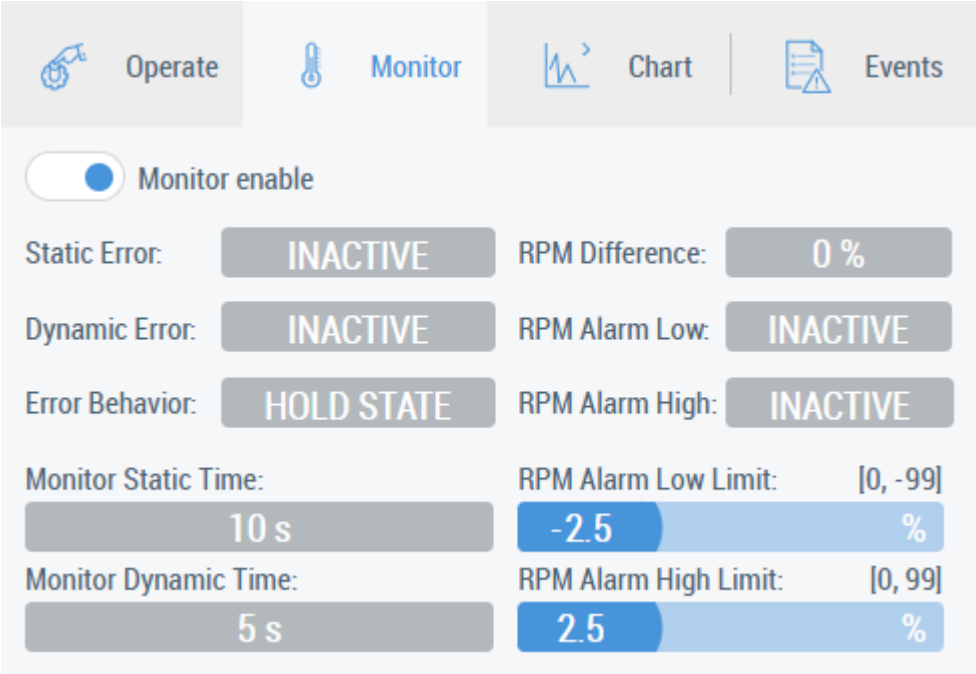


The following table shows the description of MonTriPosVlvFaceplate elements:

#	Element	Description
1	Monitor enable	Enable monitoring
2	Error Behavior	Behavior in the event of an error (hold state or go to SafePos)
3	Static Error	Static error indicator for Pos1…3
4	Dynamic Error	Dynamic error indicator for Pos1…3
5	Monitor Static Time	Time parameter (MonStatTi) for Pos1…3 static error
6	Monitor Dynamic Time	Time parameter (MonDynTi) for Pos1…3 dynamic error

4.4.2.7 MonAnaDrvFaceplate

An example of MonAnaDrvFaceplate is shown below.

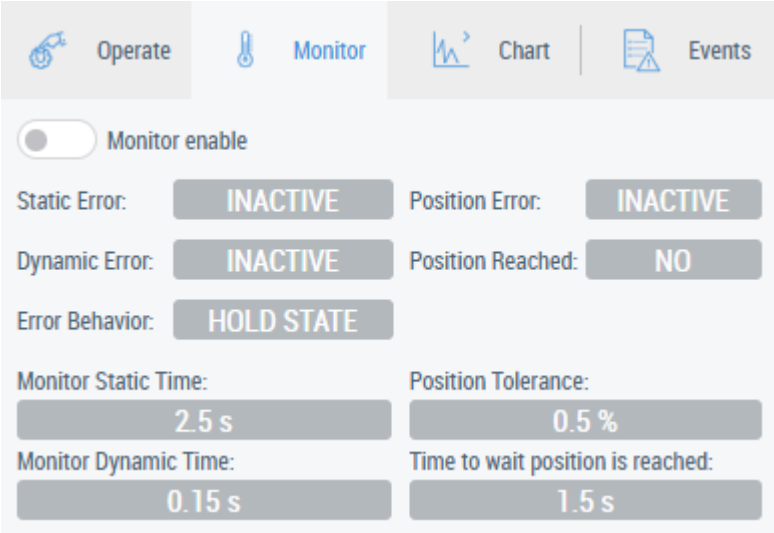


The following table shows the description of MonAnaDrvFaceplate elements:

#	Element	Description
1	Monitor enable	Enable monitoring
2	Static Error	Static error indicator
3	Dynamic Error	Dynamic error indicator
4	Error Behavior	Behavior in the event of an error (hold state or go to SafePos)
5	RPM Difference	Difference between set and current speed (speed deviation)
6	RPM Alarm Low	Indicator for speed deviation that exceeds the lower limit (#11 in this table)
7	RPM Alarm High	Indicator for speed deviation that exceeds the upper limit (#10 in this table)
8	Monitor Static Time	Time parameter (MonStatTi) for the static error
9	Monitor Dynamic Time	Time parameter (MonDynTi) for the dynamic error
10	RPM Alarm Low Limit	Lower limit for the speed deviation **RPM Difference** (#5 in this table)
11	RPM Alarm High Limit	Upper limit for the speed deviation **RPM Difference** (#5 in this table)

4.4.2.8 MonAnaVlvFaceplate

An example of MonAnaVlvFaceplate is shown below.

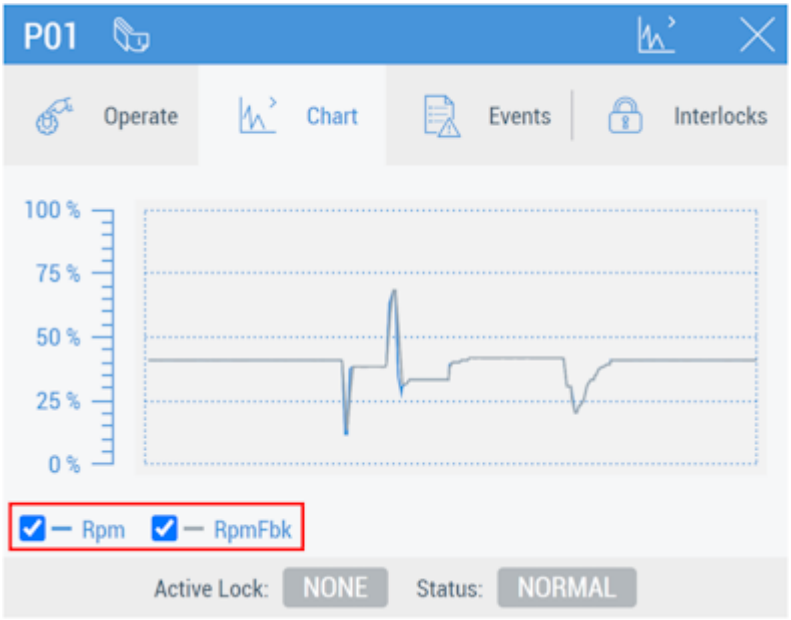


The following table shows the description of **MonAnaVlvFaceplate** elements:

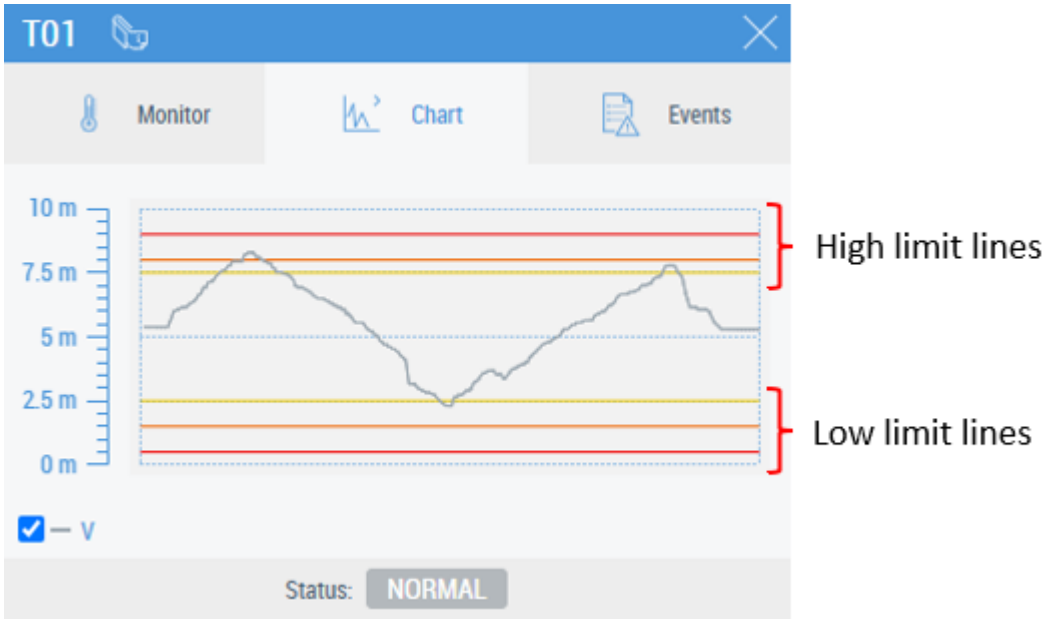
#	Element	Description
1	Monitor enable	Enable monitoring
2	Static Error	Static error indicator
3	Dynamic Error	Dynamic error indicator
4	Error Behavior	Behavior in the event of an error (hold state or go to SafePos)
5	Position Error	Indicator that the maximum travel time (#10 in this table) has been exceeded
6	Position Reached	Indicator that the target position has been reached with the set **Position Tolerance** (#9 in this table)
7	Monitor Static Time	Time parameter (MonStatTi) for the static error
8	Monitor Dynamic Time	Time parameter (MonDynTi) for the dynamic error
9	Position Tolerance	Position tolerance parameter for determining that the setpoint position has been reached
10	Time to wait position is reached	Parameter to set the maximum travel time (MonPosTi)

4.4.3 Chart tab

The set of charts and limit lines on the Chart tab depends on the type of DataAssembly that MTP PLC Library FB presents. The charts can be hidden using the corresponding controls:



An example on limit lines:



Parameters

The following table shows the description of the *Faceplate_Chart.usercontrol* parameters:

Parameter	Description	Type	Example
DataSymbolPath1	SymbolPath to a variable 1	String	ADS.PLC1.GVL_MTP.H01.V
DataSymbolPath2	SymbolPath to a variable 2	String	ADS.PLC1.GVL_MTP.H01.VFbk
DataSymbolPath3	SymbolPath to a variable 3	String	ADS.PLC1.GVL_MTP.H01.VInt
SclMin	Minimum value of chart scale	REAL	%s%ADS.PLC1.GVL_MTP.V03::PosSclMin%/s%, which looks like this while binding: 
SclMax	Maximum value of chart scale	REAL	%s%ADS.PLC1.GVL_MTP.V03::PosSclMax%/s%, which looks like this while binding: 
Unit	Units	Number	%s%ADS.PLC1.GVL_MTP.V03::PosUnit%/s%, which looks like this while binding: 
HHHEn	Show the line for HiHiHi Limit	BOOL	
HHH	Line for HiHiHi Limit	REAL	90 or 
HHEn	Show the line for HiHi Limit	BOOL	
HH	Line for HiHi Limit	REAL	See examples for HHH
HEn	Show the line for Hi Limit	BOOL	
H	Line for Hi Limit	REAL	See examples for HHH
LEn	Show the line for Low Limit	BOOL	
L	Line for Low Limit	REAL	See examples for HHH
LLEn	Show the line for LowLow Limit	BOOL	
LL	Line for LowLow Limit	REAL	See examples for HHH
LLLEn	Show the line for LowLowLow Limit	BOOL	
LLL	Line for LowLowLow Limit	REAL	See examples for HHH

Variables which are set in DataSymbolPath[N] (N is a number, e.g. 1, 2, 3) are automatically historized when the faceplate is configured in the TwinCAT HMI Engineering. The same occurs when using PLC attributes, but the variable should be preceded by the **Chart** attribute (see example of configuring Chart tab using PLC attributes [▶ 21]).

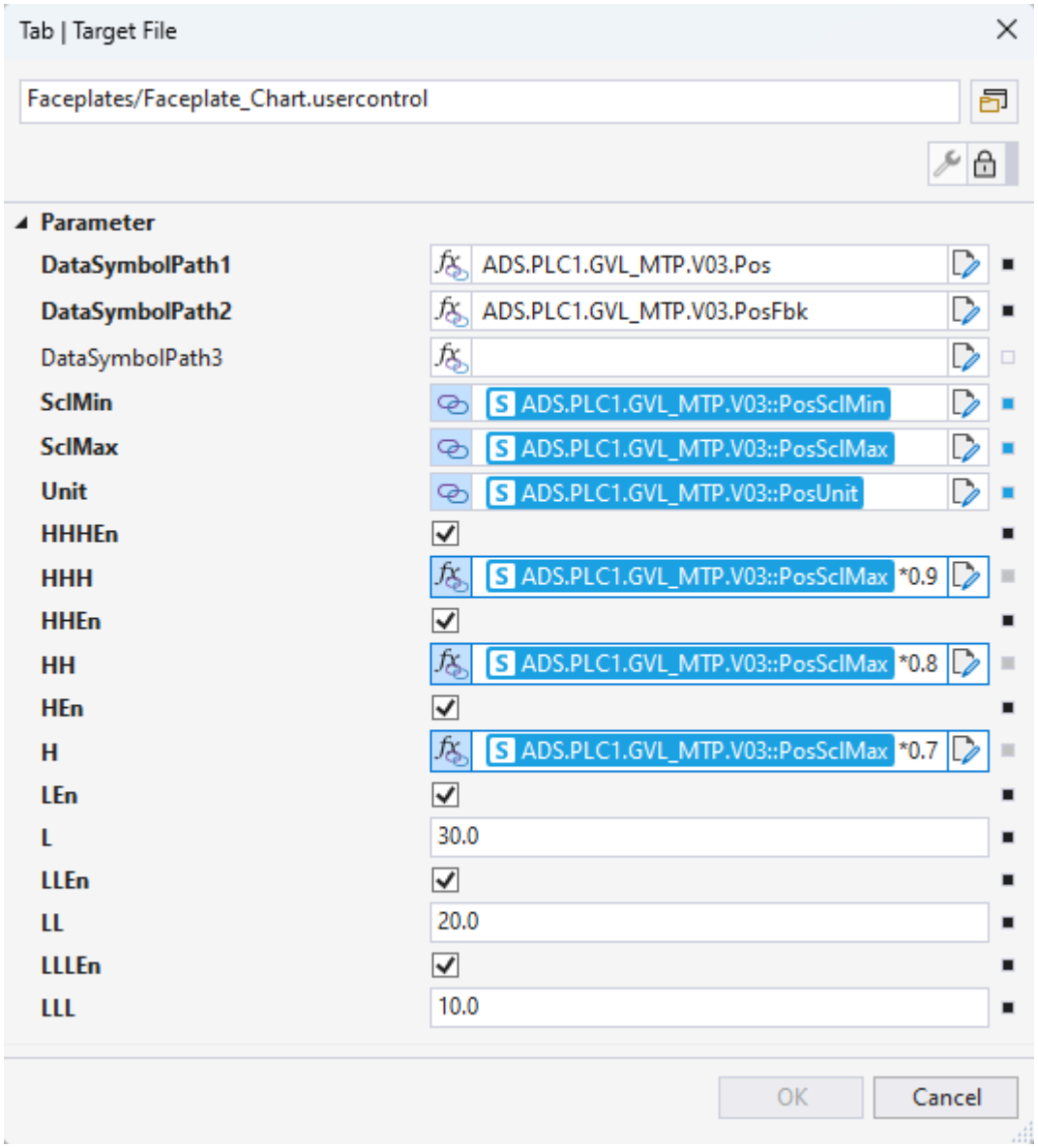
Reusing the Chart tab with the Tabs property

Faceplate_Chart.usercontrol can be used as a customized tab [▶ 34] for a faceplate. A maximum of three variables can be chosen to draw charts.

Setting symbols as historized

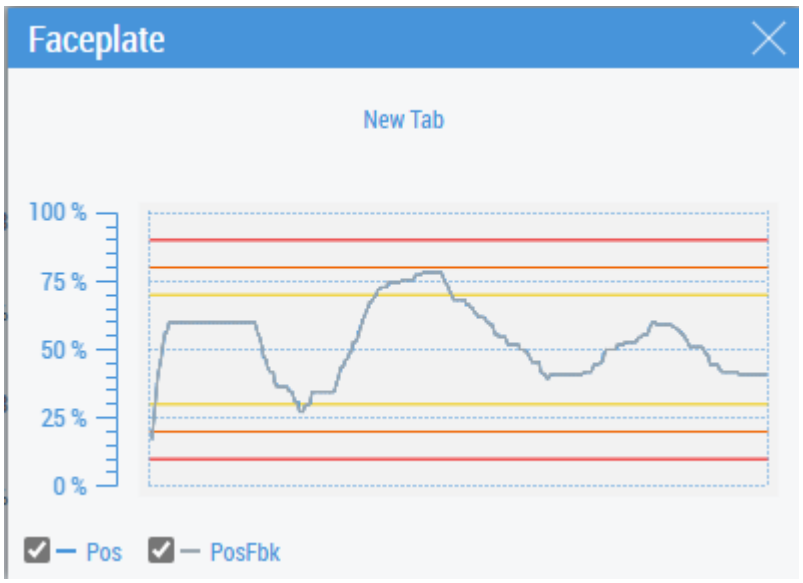
To show the chart, the symbols, bound to the **DataSymbolPath1 … DataSymbolPath3** parameters should be set as **Historized**.

Faceplates/Faceplate_Chart.usercontrol should be chosen for the **Target File** property in the Tabs configuration window. An example of parameter configuration is shown below:



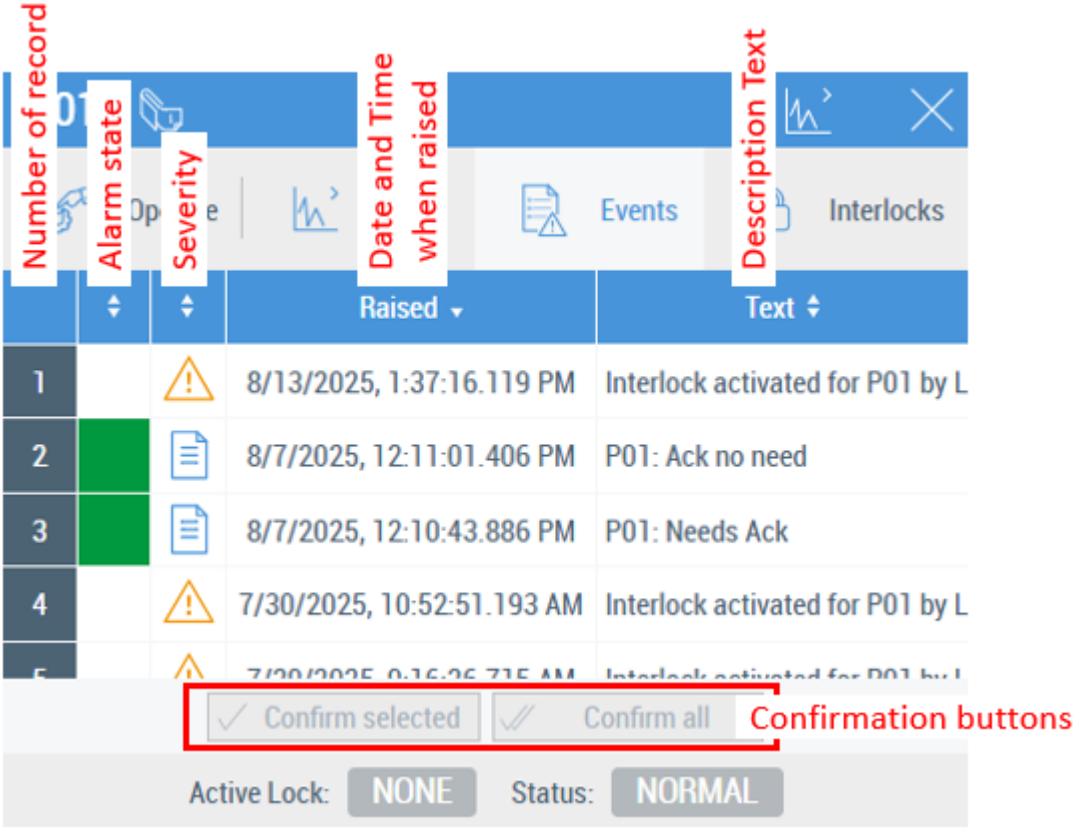
- **DataSymbolPath1** … **DataSymbolPath3** should be entered (manually, not chosen from the drop-down list) as a path to the PLC variable.
- **SclMin, SclMax, Unit** and limit values should be linked as symbols or entered as JavaScript expressions.

The result of the configuration is shown below:



4.4.4 Events tab

The set of events in the Events tab corresponds to the P&ID element which the control presents. Columns marked with an arrow/arrows can be sorted by order. An example of the Events tab with a description of columns and buttons is given below.



This table explains the meanings of the colors for the **Alarm state** column:

View	Color name	Event type	Needs acknowledgement	Meaning
	White	Message	No	Doesn't have any alarm states.
	Red	Alarm	Yes	The alarm is active and not yet confirmed.
			No	The alarm is active.
	Orange	Alarm	Yes	The alarm is active and already confirmed.
	Yellow-green	Alarm	Yes	The alarm is inactive but not yet confirmed.
	Green	Alarm	Yes	The alarm is inactive and confirmed.
			No	The alarm is inactive.

Parameters

Faceplate_Events.usercontrol is based on the TwinCAT HMI Event Grid control. There is only one parameter, **Filter** (Datatype is eventFilter), which can be configured according to the TwinCAT HMI Engineering manual.

Reusing the Events tab with the Tabs property

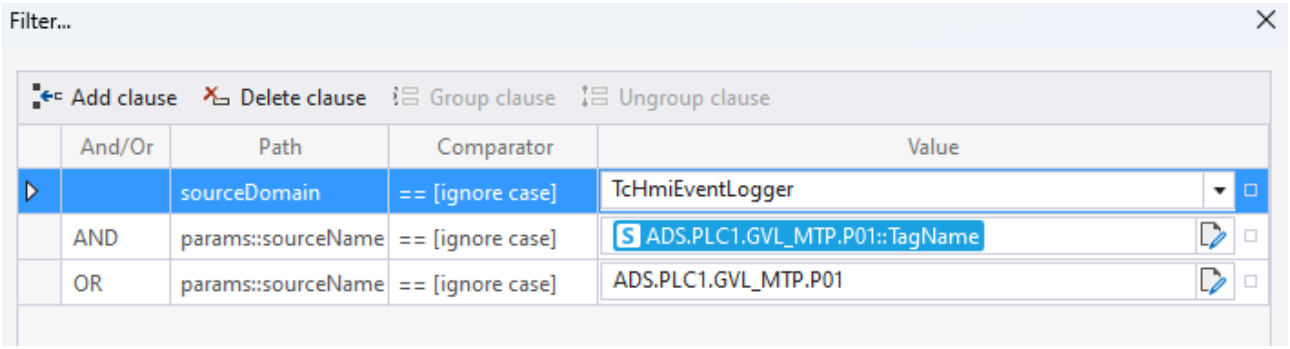
Faceplate_Events.usercontrol can be used as a customized tab [▶ 34] for a faceplate.

Generating events

Necessary events should be generated in the PLC program (see TC3 EventLogger Manual).

Faceplates/Faceplate_Events.usercontrol should be chosen for the **Target File** property in the Tabs configuration window. The **Filter** parameter should be configured according to the TwinCAT HMI Engineering manual (event grid control).

An example of the configuration is shown below:



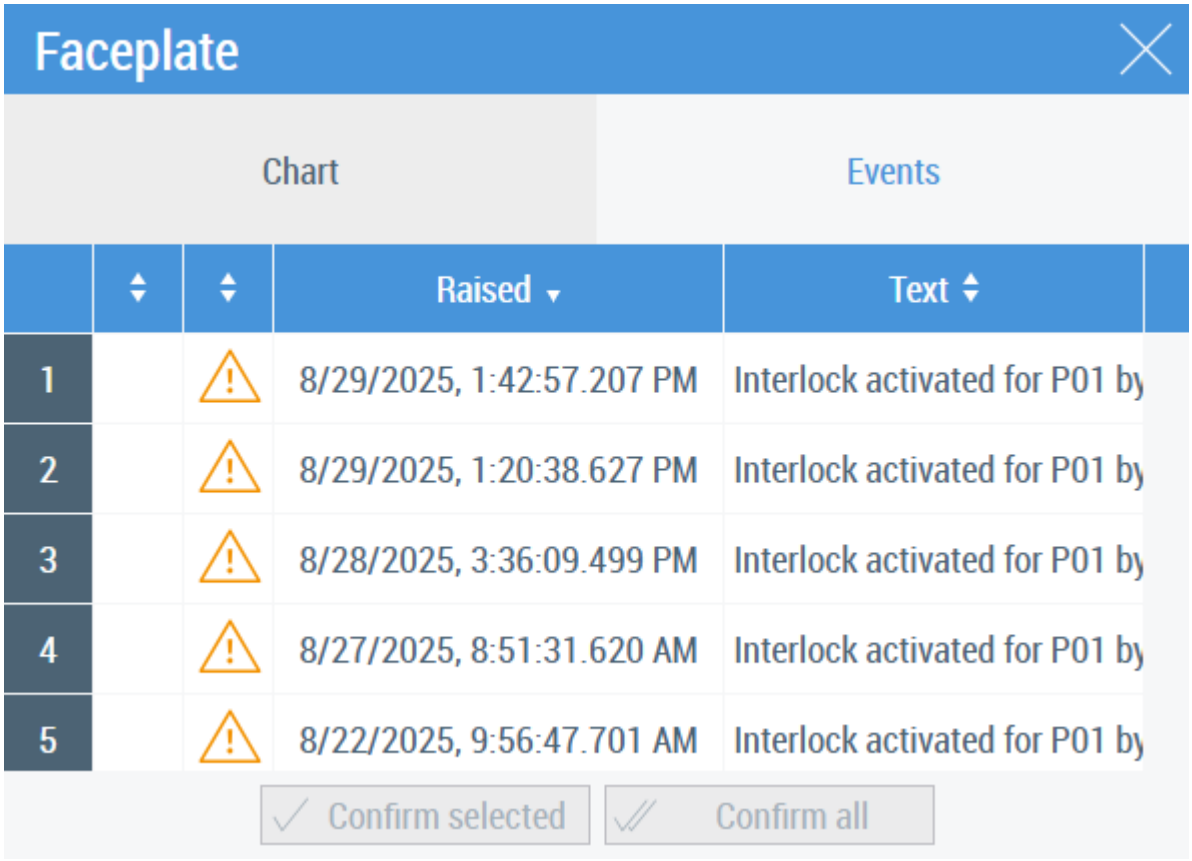
The following are chosen as displayed event sources:

- SymbolExpresson for the TagName value of DataAssembly FB P01 (should be input manually or copied from somewhere)
  `%s%ADS.PLC1.GVL_MTP.P01::TagName%/s%`
  This can be any other SymbolExpression which gives a name of the event source. This is a general-purpose method for referring to the event source.

- SymbolPath for the DataAssembly FB P01 (should be input manually or copied from somewhere)
  `ADS.PLC1.GVL_MTP.P01`
  It can be a SymbolPath for an FB which represents MTP DataAssembly. This is a specific method for referring to events of the DataAssembly for the MTP PLC Library. This method is not currently implemented on the MTP PLC Library side.

The value `param::sourceName` in the **Path** column should be input manually.

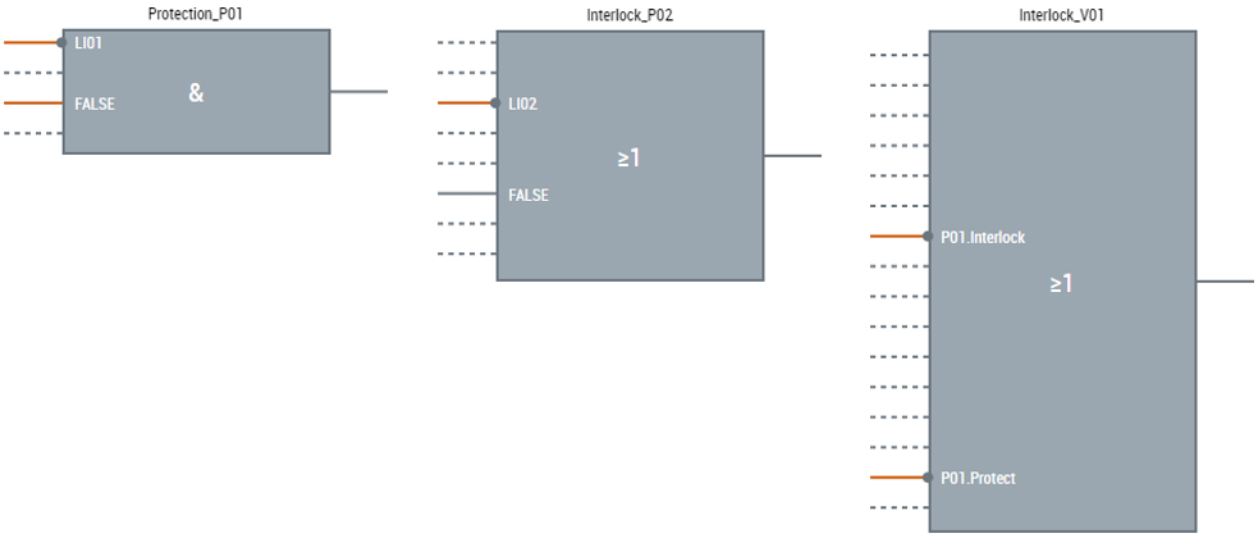
The result of the configuration is shown below:



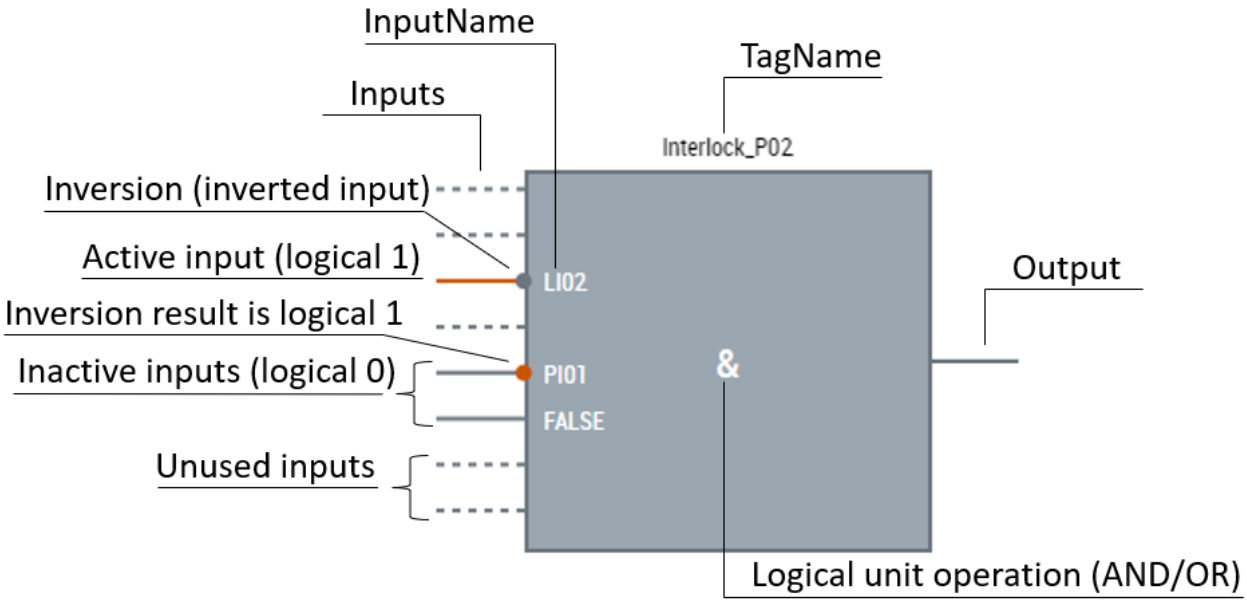
4.4.5 Interlocks tab

To appear in a predefined faceplate, the Interlocks tab should be configured using the **Interlocks Tab** property (see table [▶ 31]) of the HMI Process Library control. The configuration of the Interlocks tab is explained in the Category: Faceplate [▶ 32] chapter.

Depending on the quantity of interlock reasons, the control may have 4, 8, or 16 inputs:



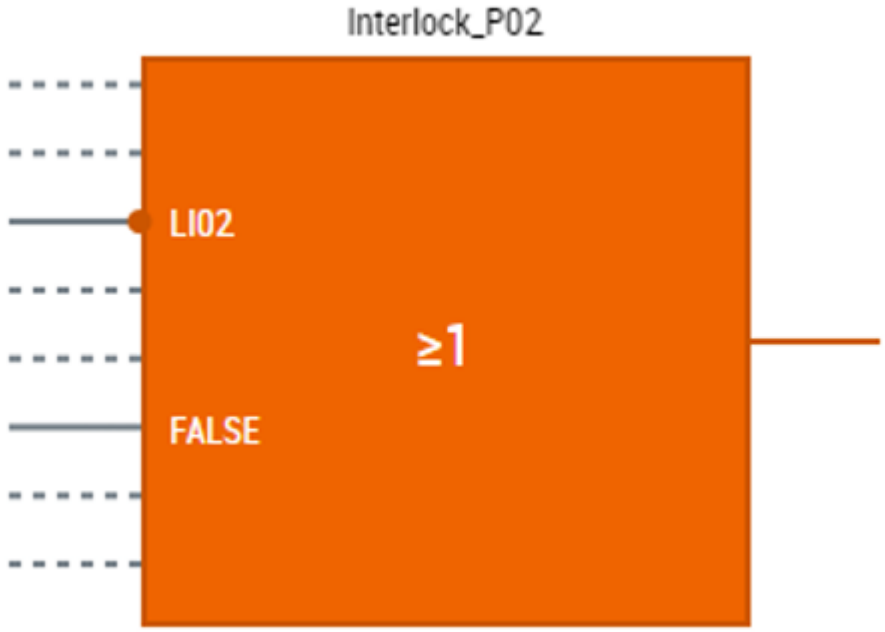
The picture below describes elements of the interlock control:



Designation for AND operation is **&**, OR is **≥1**. If the result of input inversion operation is:

- logical 0, it is presented in grey;
- logical 1, it is presented in orange.

Logical unit operation (AND or OR) is carried out on the control inputs' values, excluding unused inputs. The inversion of the input is considered. If the result of the logical operation is 0, the control stays grey. If the result is 1, it means that the lock is active, and the control and the output turn orange:

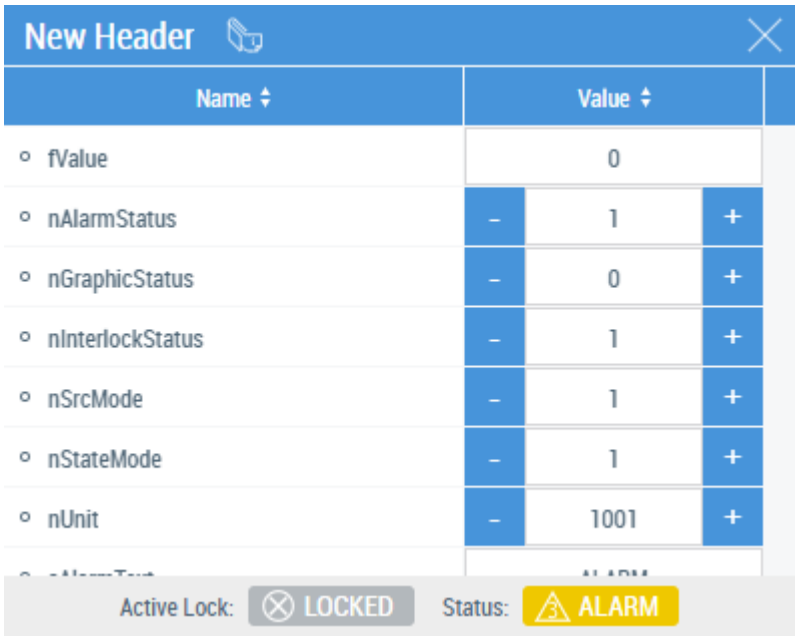


Parameters

LockView[N]_Interlocks.usercontrol (where N can be 4, 8 or 16) only has one parameter **DataSymbol** which should be an instance of the corresponding MTP LockView FB with the corresponding number of the interlock reasons.

4.4.6 ObjectBrowser view on a Faceplate

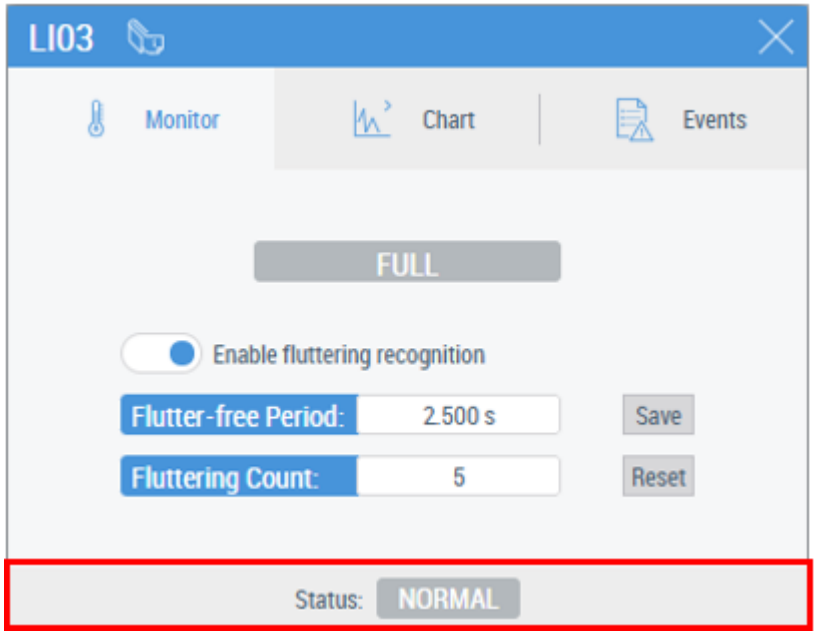
When no faceplate is defined for the HMI Process Library control (via **Data Symbol** property MTP type, **Tabs**, or **Faceplate Control** properties), the faceplate shows data which is defined using PLC attribute functionality (chapter PLC attribute functionality [▶ 82]). This data can be both read and written using ObjectBrowser view:



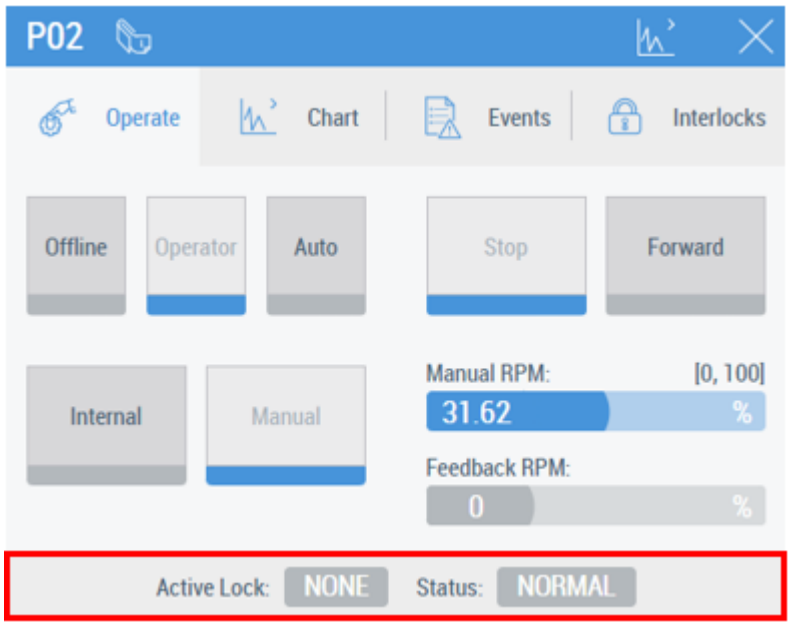
4.4.7 Footer

Some of the predefined faceplates have a footer to show current locks, statuses, warnings and alarms for their P&ID element. A set of the footer elements depends on the type of DataAssembly which MTP PLC Library FB presents.

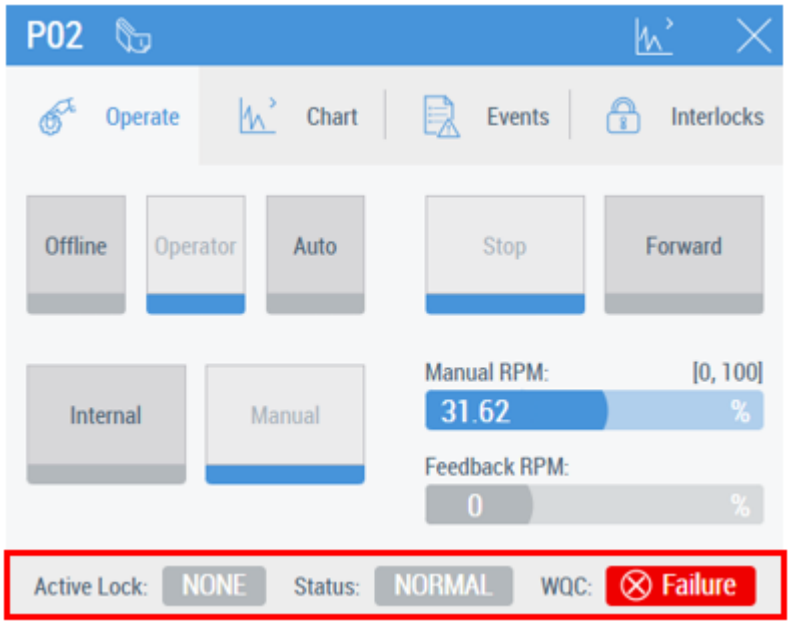
This is an example of a faceplate with the footer containing only the **Status** element (in “NORMAL” state):



This is an example of a faceplate with the footer containing the **Active Lock** element (in state “NONE”) and the **Status** element (in “NORMAL” state):



For faceplates which have the footer, the **WQC** message element also appears in addition to other messages if the WQC status has an active value (see table [▶ 77]). An example of an active **WQC** message element:



The following table shows the **Status** element states and gives a list of MTP types which may have the specific state on their predefined faceplates:

#	Status element state	Description	MTP types
1		Normal state	AnaMon DIntMon AnaDrv MonAnaDrv AnaVlv MonAnaVlv BinMon BinDrv MonBinDrv BinVlv MonBinVlv TriPosVlv MonTriPosVlv
2		Alarm High Active	AnaMon DIntMon
3		Alarm Low Active	AnaMon DIntMon
4		Warning High Active	AnaMon DIntMon
5		Warning Low Active	AnaMon DIntMon
6		Tolerance High Active	AnaMon DIntMon
7		Tolerance Low Active	AnaMon DIntMon
8		Safe Position is activated (safe operation)	AnaDrv MonAnaDrv AnaVlv MonAnaVlv BinDrv MonBinDrv BinVlv MonBinVlv TriPosVlv MonTriPosVlv
9		Drive Protection triggered: there is a problem with the drive	AnaDrv MonAnaDrv BinDrv MonBinDrv
10		RPM alarm high is active	MonAnaDrv
11		RPM alarm low is active	MonAnaDrv

#	Status element state	Description	MTP types
12		Hold state is active after monitoring error triggered	MonAnaDrv MonAnaVlv MonBinDrv MonBinVlv MonTriPosVlv
13		Safe position is active after monitoring error triggered	MonAnaDrv MonAnaVlv MonBinDrv MonBinVlv MonTriPosVlv
14		Fluttering signal recognized	BinMon
15		Fallback message if alarm is active	
16		Fallback message if warning is active	
17		Fallback message if tolerance is active	

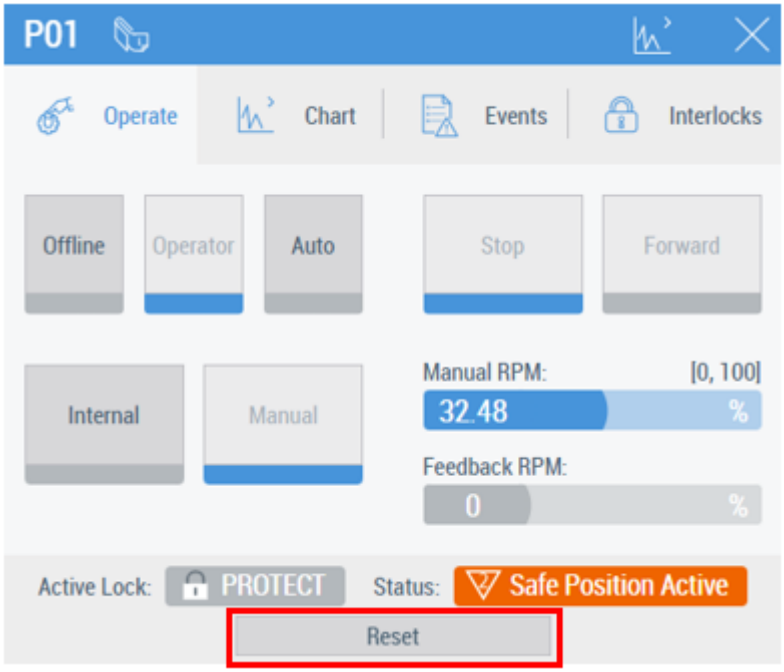
The following table describes the **Active Lock** element states and gives a list of MTP types which can have the specific state on their predefined faceplates:

#	Active Lock element state	Description	MTP types
1		No active locks	AnaDrv MonAnaDrv AnaVlv MonAnaVlv BinDrv MonBinDrv BinVlv MonBinVlv TriPosVlv MonTriPosVlv
2		Allows control	
3		Prevents from being activated and sets it to safe position	
4		Prevents from being activated and sets it to safe position, requires a reset command	

The following table describes the **WQC** element states and gives a list of MTP types which can have the specific state on their predefined faceplates:

#	WQC element state	Description	MTP types
1	Failure	Failure	AnaDIntMon DIntMon AnaDrv MonAnaDrv AnaVlv MonAnaVlv BinMon BinDrv MonBinDrv BinVlv MonBinVlv TriPosVlv MonTriPosVlv
2	Function Check	Function Check	
3	Maintenance Request	Maintenance Request	
4	Out of Specification	Out of Specification	
5	Out of Service	Out of Service	

When an emergency state of the control is active, the reset button appears:



4.5 Functions

A number of functions come with the HMI Process Library. The following table shows additional general functions from the HMI Process Library:

Name	Category	Description
AnalogCompressionValueFormatter	Formatting	Return string for the passed value in compressed format (not bigger than given number of characters, excluding sign and dot). Return value type: *String*. Arguments: • value : The value being presented in compressed format. From -10^{15} to $+10^{15}$ (excluding limits). The smallest absolute value can be 10^{-15}. type: *String.* • maxCharacters : Maximum number of characters the value should have (excluding sign and dot; can't be less than 4). type: *Number.* • showPlus : If true, show plus sign for positive values. type: *Boolean.* • fallbackValue : The result returned if the value is undefined or null. The default fallback value is 'NaN'. type: *String.*
AnalogValueFormatter	Formatting	Return formatted string for the value passed in. Return value type: *String*. Arguments: • value : The value to format. type: *String.* • maxDecimals : How many decimal places the value should have. type: *Number.* • showPlus : If true, show plus sign for positive values. type: *Boolean.* • fallbackValue : The result returned if the value is undefined or null. The default fallback value is 'NaN'. type: *String.*
BinaryValueFormatter	Formatting	Return formatted string for the value passed in. Return value type: *String*. Arguments: • value : The value to format. type: *String.* • trueValue : Replacement string for value true. type: *String.* • falseValue : Replacement string for value false. type: *String.* • fallbackValue : The result returned if the value is not TRUE or FALSE. The default fallback value is empty string. type: *String.*

Name	Category	Description
TernaryValueFormatter	Formatting	Return formatted string for the value passed in. [bVal1.toString() + bVal2.toString()]. Return value type: *String*. Arguments: • value : The value to format [bVal1.toString() + bVal2.toString()], valid values ◦ 'truefalse' ◦ 'falsetrue' ◦ 'falsefalse' type: *String*. • truefalseValue : Replacement string for value ‘truefalse’. type: *String*. • falsetrueValue : Replacement string for value ‘falsetrue’. type: *String*. • falsefalseValue : Replacement string for value ‘falsefalse’. type: *String*. • fallbackValue : The result returned if the value is not in the range of possible ones. The default fallback value is empty string. type: *String*.
MapValueFormatter	Formatting	Returns a value which corresponds to the passed key. Return value type: *String*. Arguments: • key : The value to substitute type: *Any*. • mapObj : Map for replacements **key** -> **value**. For example: {"key1": "value1", "key2": "value2"} type: *Object*. • fallbackValue : The result returned if the key is not among keys of the mapObje. The default fallback value is an empty string. type: *String*.

The following table shows the additional MTP specific functions from the HMI Process Library:

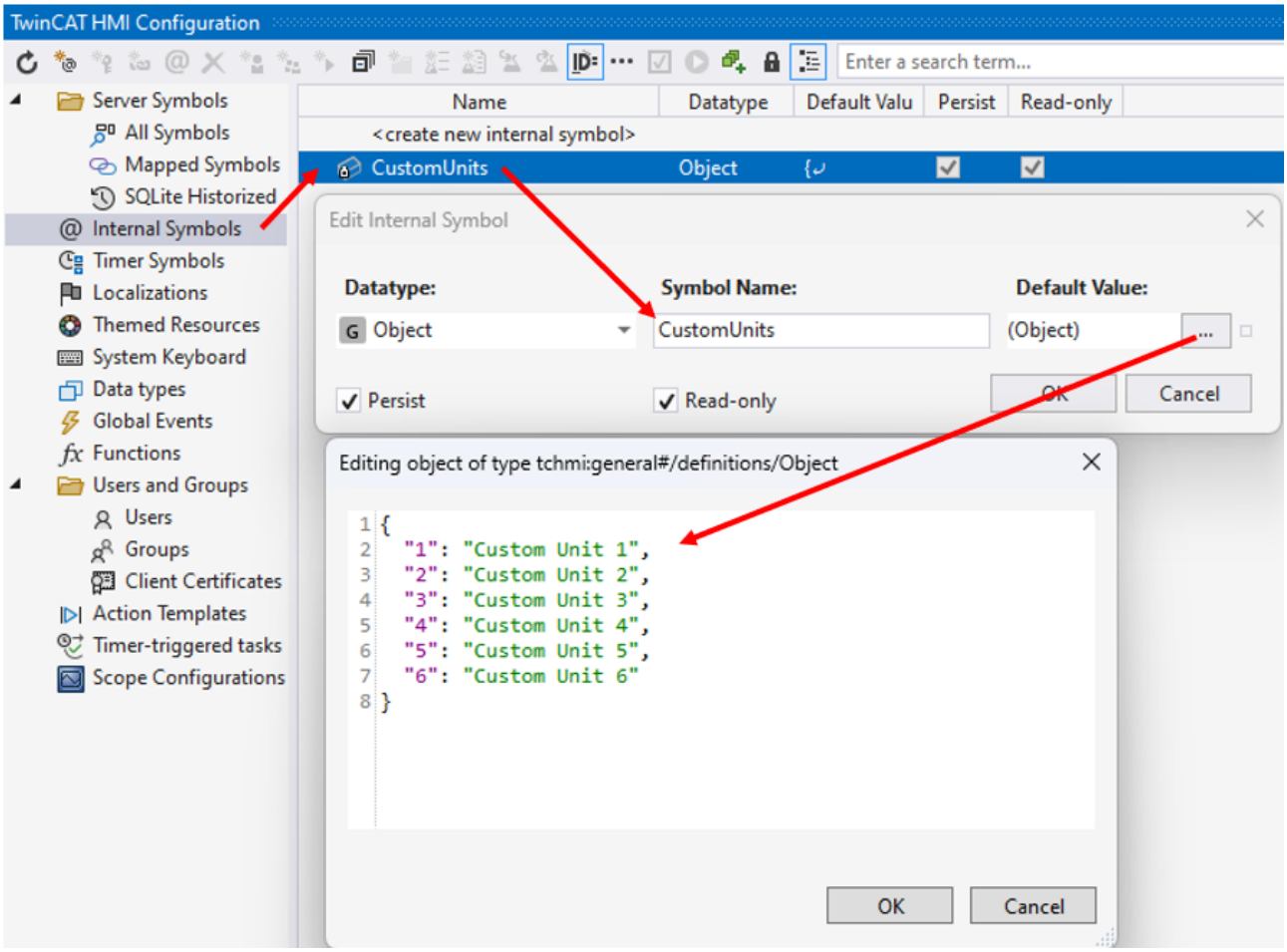
Name	Category	Description
MTPIsCommandEnabled	Checking	Return true if command is enabled. Return value type: *Boolean*. Arguments: • CommandEN : CommandEN variable of service. type: *Number.* • Command : Command to be checked. type: *String.*
MTPStateValueFormatter	Formatting	Return formatted state string for the value passed in. Return value type: *String*. Arguments: • value : The value to format. type: *String.*
MTPUnitValueFormatter	Formatting	Return formatted unit string for the value passed in. Return value type: *String*. Arguments: • value : The unit value to format. type: String. • customUnits: Dictionary for replacements **num** -> **unit**. **num** should be a positive integer number enclosed in quotation marks. The default value is {"1": "Custom unit 1", "2": "Custom unit 2"} type: Object. • fallbackValue : The result returned if the value is undefined or null. The default fallback value is an empty string. type: *String.*

4.6 Non-standard engineering units

HMI Process Library components can present non-standard engineering units. The table below lists such components:

Element	Name of Property/Parameter
Function MTPUnitValueFormatter	customUnits
Controls inherited from ProcessObject	Custom Units
Service control	Custom Units
Faceplates which present engineering units	CustomUnits

It's possible to set the **Custom Units** properties and parameters for every element individually but the most convenient way is to create, for example, an internal symbol in the HMI project, which defines all the required non-standard units, and then bind it to the corresponding properties/parameters of the elements. An example of an internal symbol and its initialization is shown below:



4.7 PLC attribute functionality

PLC attributes can be used to configure the TwinCAT Process HMI control and pass some current data to its elements from an FB created by the user. Two actions are necessary for this:

- add the PLC attributes in the FB
- map the FB to the control

Naming FBs with PLC attributes

To work correctly, user FB names **should not** contain DataAssembly names from the MTP Specification (Part 3), i.e. “BinVlv”, “AnaView”, etc. as part of them. These names are a part of MTP Runtime Library FB names, for example, FB_MTP_BinVlv, FB_MTP_AnaView (see manual for the TF8400 | MTP Runtime).

PLC attributes can be divided into two types:

- Configuration PLC attributes: define some properties and behavior of the control when the TwinCAT HMI page starts.
- Data PLC attributes: define variables which represent values of the control elements when the TwinCAT HMI page is running.

In this chapter, a list of the attributes is given. Detailed information on how to use the attributes in the PLC program is given in the Using PLC attribute functionality [▶ 21] chapter.

Using [ ] in the code templates

Text enclosed in square brackets [ ] in templates for code means a placeholder which should be changed to relevant text.

The full attribute name is `'TcHmi.ProcessLibrary.[Attribute Name]'`, where:

- `TcHmi.ProcessLibrary` is a fixed part.
- `[Attribute Name]` can contain only one name (i.e. **RestoreBounds**) or several names in hierarchical order, separated by '.' (i.e. **FaceplateControl.TargetFile**).

4.7.1 Special placeholder description

4.7.1.1 Placeholder PLPATH^

Placeholder **PLPATH^** refers to the root directory of the TwinCAT HMI Process Library.

4.7.1.2 Placeholders THIS^ and THISEXP^

Placeholder **THIS^** refers to the PLC symbol mapped to the control. It substitutes SymbolPath, i.e.:

```
ADS.PLC1.GVL_MTP.V03
```

Placeholder **THISEXP^** refers to the PLC symbol mapped to the control. It substitutes SymbolExpression, i.e.:

```
%s%THIS^%/s% or %s%ADS.PLC1.GVL_MTP.V03%/s%
```

An example of a value assignment inside the PLC attribute with **THISEXP^** can look like this:

```
DataSymbol:=THISEXP^
```

4.7.1.3 Placeholders STAG^ and ETAG^

Placeholder **STAG^** substitutes start tags of SymbolExpression `%s%`, `%pp%`, etc.

Placeholder **ETAG^** substitutes end tags of SymbolExpression `%/s%`, `%/pp%`, etc.

an example of a value assignment inside the PLC attribute using **STAG^** and **ETAG^** can look like this:

```
nUnit:=STAG^THIS^::nUnitETAG^
```

4.7.2 Configuration PLC attributes description

Almost all of the configuration PLC attributes represent the properties of the ProcessObject, described in the Properties [▶ 30] chapter and some which are specific for the inherited controls. Not all of the properties are implemented as the configuration PLC attributes, but implemented ones have the same name (just without break spaces) and functionality. The following table contains a list of the configuration PLC attributes (excluding the Faceplate tab and referencing function attributes).

Attribute Name	Category	Description	Possible Values
GraphicControl	Common		
FaceplateControl.TargetFile	Faceplate	**Faceplate Control** property: path to *.usercontrol file	
FaceplateControl.Parameter	Faceplate	**Faceplate Control** property: a SymbolExpression for the symbol providing data for the faceplate	
ShowFaceplate	Faceplate		False, True
Modal	Faceplate		False, True
Movable	Faceplate		False, True
RestorePosition	Faceplate		False, True
HideWithControl	Faceplate		False, True
ReshowWithControl	Faceplate		False, True
LabelTextHorizontalAlignment	Label		Left, Right, Center
LabelTextVerticalAlignment	Label		Top, Bottom, Center
LabelValueHorizontalAlignment	Label		Left, Right, Center
LabelValueVerticalAlignment	Label		Top, Bottom, Center
LabelModeHorizontalAlignment	Label		Left, Right, Center
LabelModeVerticalAlignment	Label		Top, Bottom, Center
LabelStatusHorizontalAlignment	Label		Left, Right, Center
LabelStatusVerticalAlignment	Label		Top, Bottom, Center
LabelTextPadding.Left	Label		[Number]
LabelTextPadding.LeftUnit	Label		px, %
LabelTextPadding.Top	Label		[Number]
LabelTextPadding.TopUnit	Label		px, %
LabelTextPadding.Right	Label		[Number]
LabelTextPadding.RightUnit	Label		px, %
LabelTextPadding.Bottom	Label		[Number]
LabelTextPadding.BottomUnit	Label		px, %
LabelValuePadding.Left	Label		[Number]
LabelValuePadding.LeftUnit	Label		px, %
LabelValuePadding.Top	Label		[Number]
LabelValuePadding.TopUnit	Label		px, %
LabelValuePadding.Right	Label		[Number]
LabelValuePadding.RightUnit	Label		px, %
LabelValuePadding.Bottom	Label		[Number]
LabelValuePadding.BottomUnit	Label		px, %
LabelModePadding.Left	Label		[Number]
LabelModePadding.LeftUnit	Label		px, %
LabelModePadding.Top	Label		[Number]
LabelModePadding.TopUnit	Label		px, %
LabelModePadding.Right	Label		[Number]
LabelModePadding.RightUnit	Label		px, %
LabelModePadding.Bottom	Label		[Number]
LabelModePadding.BottomUnit	Label		px, %
LabelStatusPadding.Left	Label		[Number]
LabelStatusPadding.LeftUnit	Label		px, %
LabelStatusPadding.Top	Label		[Number]

Attribute Name	Category	Description	Possible Values
LabelStatusPadding.TopUnit	Label		px, %
LabelStatusPadding.Right	Label		[Number]
LabelStatusPadding.RightUnit	Label		px, %
LabelStatusPadding.Bottom	Label		[Number]
LabelStatusPadding.BottomUnit	Label		px, %
Chart	Chart	Variable automatically becomes historized in TwinCAT HMI	Doesn't need any values
ShowRotation	Rotation	Specific for Agitators and Conveyors	False, True
RotationDuration	Rotation	Specific for Agitators and Conveyors	[Number]

4.7.2.1 Function attributes description

Some of the configuration attributes are designated to define a function to handle the value. The **TcHmi.ProcessLibrary.Format** attribute is an attribute that defines a formatting function. The following table contains a list of the PLC attributes which can be used together with the **TcHmi.ProcessLibrary.Format** attribute.

Attribute Name	Category	Example of TcHmi.ProcessLibrary.Format value
LabelValue	Label	`TcHmi.Functions.Beckhoff.PI.ProcessLibrary.AnalogCompressionValueFormatter,4`
LabelValueUnit	Label	`TcHmi.Functions.Beckhoff.PI.ProcessLibrary.MTPUnitValueFormatter`

4.7.2.2 Faceplate tab attributes description

Faceplate tab configuration attributes define the number and content of the faceplate tabs. The full Faceplate tab attribute name is

'TcHmi.ProcessLibrary.Tab[Number].[Attribute Name]',

where:

- **TcHmi.ProcessLibrary.Tab** is a fixed part
- **[Number]** is a number of the Tab (1, 2, 3, etc.)
- **[Attribute Name]** contains the Tab attribute name

The values of the Tab attributes can only contain one parameter or several parameters separated by a comma. In the following table, the parameters are described in in the order they appear.

Attribute Name	Description	Example
Name	Title of the tab	
TargetFile	Parameters: • path to *.usercontrol file representing the tab • enable preload for the tab content (false, true) ***)** • enable pre attach for the tab content (false, true) ***)** • enable keep alive for the tab content (false, true) ***)**	
Parameter	SymbolExpression or SymbolPath for the symbol providing data for the faceplate	
Alignment	Alignment of the tab title. Parameters: • horizontal alignment (Left, Right, Center) • vertical alignment (Top, Bottom, Center)	
Icon	Icon next to the tab title. Parameters: • path to the image file *.svg • width (Number) • height (Number) • width units (px, %) • height units (px, %)	

***)** – See the "Control life cycle" paragraph paragraph in the TE2000 TC3 HMI manual for detailed information.

4.7.3 Data PLC attributes description

Almost all of the data PLC attributes represent the properties of the ProcessObject described in the Properties [▶ 30] chapter and some which are specific for the inherited controls. Not all of the properties are implemented as the data PLC attributes, but implemented ones have the same name (just without breakspaces) and functionality. The following table contains a list of the data PLC attributes.

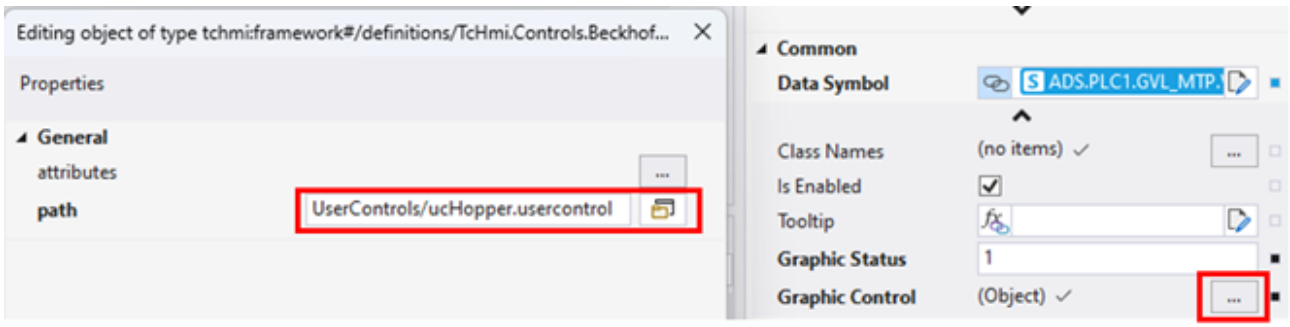
Attribute Name	Category	Description
GraphicStatus	Common	
HeaderDescription	Faceplate	
HeaderText	Faceplate	
LabelText	Label	
LabelValue	Label	
LabelValueAdd1	Label	Value for additional LabelValue 1 (see use case at the section Data PLC attributes [▶ 23])
LabelValueAdd2	Label	Value for additional LabelValue 2
LabelValueUnit	Label	
LabelValueUnitAdd1	Label	Units for additional LabelValue 1
LabelValueUnitAdd2	Label	Units for additional LabelValue 2
LabelStateMode	Label	
LabelSourceMode	Label	
LabelInterlockStatus	Label	
LabelAlarmStatus	Label	
LabelWQCStatus	Label	
LabelInterlockMessage		
LabelAlarmMessage		
LabelWQCMessage		
ValvePosition	Common	Specific for 3- and 4-way valves

5 Control appearance customization

The **Graphic Control** property can be used to change the default graphical appearance of the control.

Using SVG images within UserControls designated for the **Graphic Control** property allows styles which are defined in the HMI Process Library to be followed. Some recommendations should be implemented here.

In general, the following should be selected for the **Graphic Control** property **.usercontrol* element with desired graphical presentation of the control:

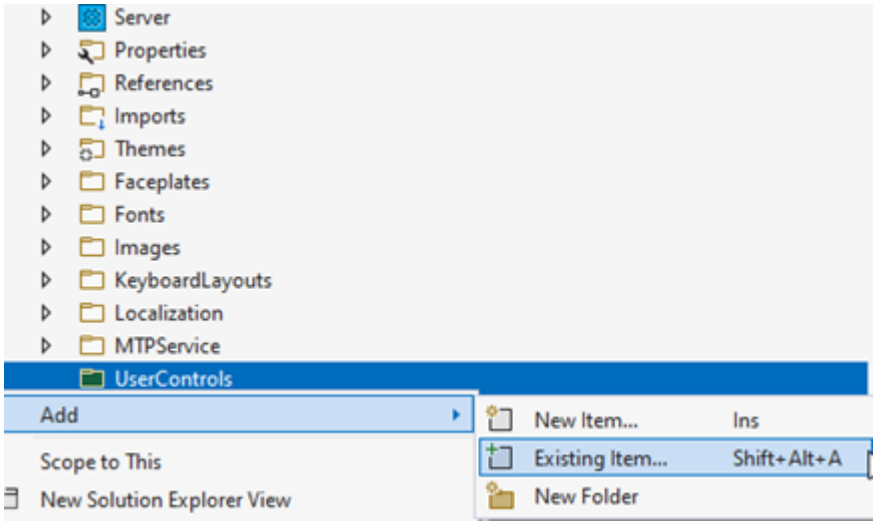


5.1 UserControl files

Every UserControl is represented by 2 files:

- **.usercontrol*
- **.usercontrol.json*

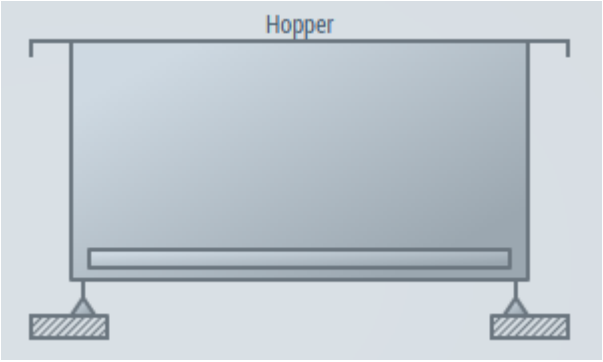
Several UserControls to use with the **Graphic Control** property can be requested as an example by e-mailing processindustry@beckhoff.com. They should just be added to the HMI project which uses the HMI Process Library. For example, "*UserControls*" folder can be added to the HMI project and then desired UserControls can be added in it as Existing Items:



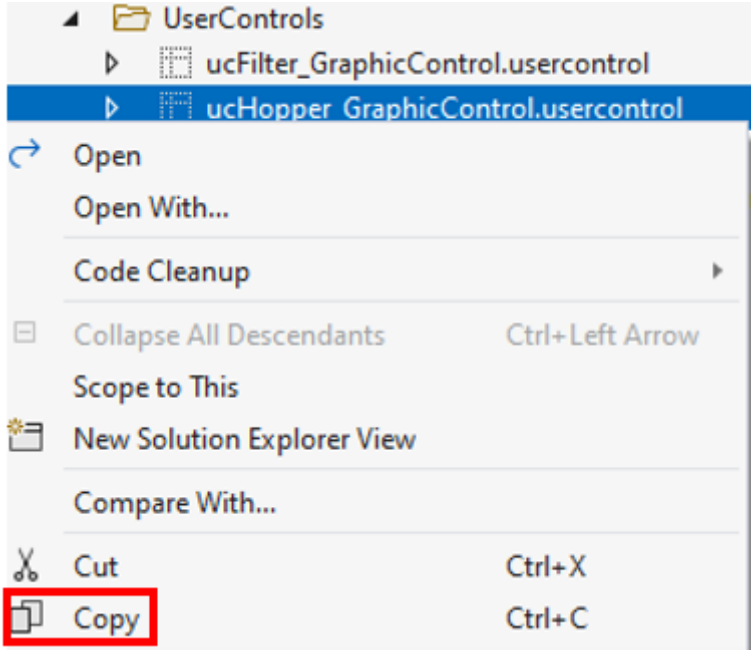
Simply choose **.usercontrol* files and the corresponding **.json* files will be added automatically.

5.2 Creating a new UserControl with SVG based on the existing sample

Further explanations are based on the ucHopper_GraphicControl sample



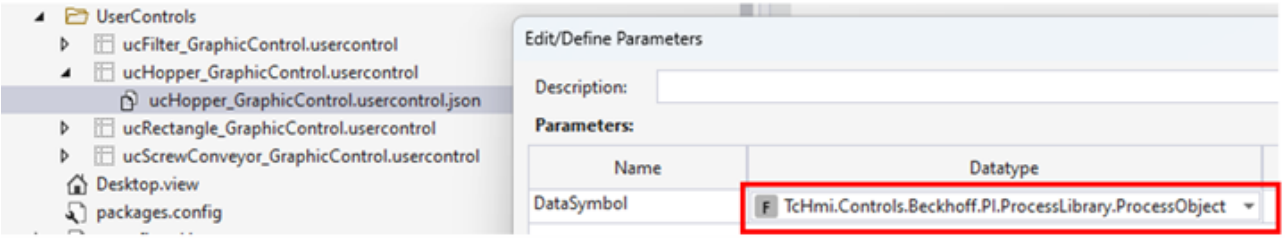
- ucHopper_GraphicControl should be copied



- then pasted (can be pasted in the same project folder)
- the new UserControl name should be changed to the desired one

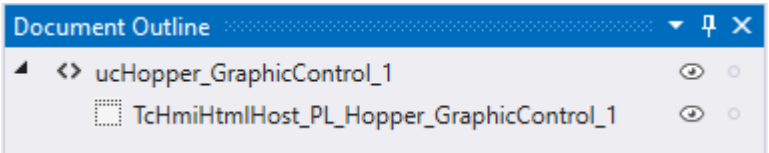
5.3 Editing the *.usercontrol.json file

The type of **DataSymbol** parameter can be changed to the required one if necessary:



5.4 Editing the *.usercontrol file

The **.usercontrol* file contains HTML host control (TcHmi.Controls.System.TcHmiHtmlHost):



SVG code should be edited directly in HTML. Attributes and parts which are **highlighted in yellow** can be edited.

5.4.1 and <svg> attributes

```
<div id="ucHopper_GraphicControl_1" data-tchmi-type="TcHmi.Controls.System.TcHmiUserControl"
     data-tchmi-top="0" data-tchmi-left="0" data-tchmi-width="100" data-tchmi-height="100"
     data-tchmi-width-unit="%" data-tchmi-height-unit="%"
     data-tchmi-creator-viewport-width="90" data-tchmi-creator-viewport-height="50"
     data-tchmi-width-mode="Value" data-tchmi-height-mode="Value">
<div id="TcHmiHtmlHost_PL_Hopper_GraphicControl_1" data-tchmi-type="TcHmi.Controls.System.TcHmiHtmlHost"
     data-tchmi-height="100" data-tchmi-height-unit="%" data-tchmi-left="0" data-tchmi-left-unit="px"
     data-tchmi-top="0" data-tchmi-top-unit="px" data-tchmi-width="100" data-tchmi-width-unit="%"
     data-tchmi-width-mode="Value" data-tchmi-height-mode="Value" data-tchmi-is-enabled="true">
    <svg class="SVG" width="100%" height="100%" viewBox="0 0 90 50" preserveAspectRatio="none">
```

- The **id** should be changed so that it is unique in the HMI project.
- **Width** and **height** should be the same for UserControl and SVG viewBox. SVG viewBox sets dimensions of the canvas that svg elements are placed on.
- Attribute **preserveAspectRatio**="none" allows the SVG to be scaled without keeping its default proportions. Deleting this attribute means that the proportions will be maintained while scaling.

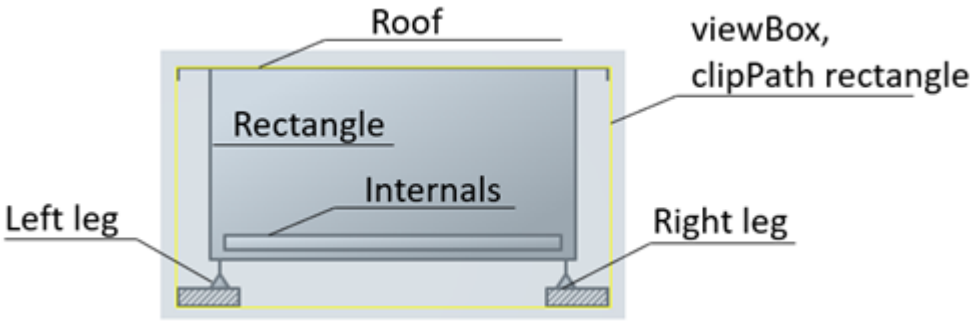
5.4.2 <defs> part

```
        <defs>
            <linearGradient id="SVGBackground_{Id}" x1="0" y1="1" x2="0" y2="0"
                            class="tchmi-styleprovider-linear-gradient SVGGradient-Rotate"
                            gradientTransform="rotate(135, 0.5, 0.5)">
                <stop class="Color0" offset="0%" />
                <stop class="Color100" offset="100%" />
            </linearGradient>
        </defs>
        <defs>
            <clipPath id="SVGBorderPath_Hopper">
                <rect width="90" height="50" x="0" y="0" />
            </clipPath>
        </defs>
        <defs>
            <pattern id="diagonalHatch_Hopper" width="1" height="1"
                     patternTransform="rotate(45)" patternUnits="userSpaceOnUse">
                <line class="SVGElement" x1="0" y1="0" x2="0" y2="2" />
            </pattern>
        </defs>
```

- The <linearGradient> section should not be changed. It provides the HMI Process Library default style for the **fill** attribute of SVG elements. **Id**=”SVGBackground_{Id}” should not be changed.
- The <clipPath> section can be changed or deleted. Its **id** attribute should be changed to make it unique within the HMI project. The <clipPath> is needed to cut the boundaries of svg elements. Several <clipPath> elements can be added. In this case, their **id** attributes should be different and unique within the HMI project.
- The <pattern> section can be changed or deleted. Its **id** attribute should be changed to make it unique within the HMI project. The <pattern> is needed to fill svg elements with some patterns, for example, hatching. Several <pattern> elements can be added. In this case, their **id** attributes should be different and unique within the HMI project.

5.4.3 The part with SVG elements

The picture below shows the SVG elements of the usHopper_Graphic control:




```
        <!--Rectangle    -->
        <rect class="SVGElement" width="76" height="40" x="7" y="0"
              fill="url(#SVGBackground_{Id})" />
        <!--Roof-->
        <path class="SVGElement SVGElement-Border" d="m0,3 0,-3 90,0 0,3"
              clip-path="url(#SVGBorderPath_Hopper)" fill="none" />
        <!--Internals-->
        <rect class="SVGElement" width="70" height="3" x="10" y="35"
              fill="url(#SVGBackground_{Id})" />
        <!--Left leg-->
        <path class="SVGElement" d="m9,40 0,3" />
        <path class="SVGElement" d="m9,43 -2,3 4,0 Z" fill="url(#SVGBackground_{Id})" />
        <rect class="SVGElement" width="13" height="4" x="0" y="46" fill="url(#diagonalHatch_Hopper)" />
        <path class="SVGElement SVGElement-Border" d="m0,46 0,4 13,0"
              clip-path="url(#SVGBorderPath_Hopper)" fill="none" />
        <!--Right leg-->
        <path class="SVGElement" d="m81,40 0,3" />
        <path class="SVGElement" d="m81,43 -2,3 4,0 Z" fill="url(#SVGBackground_{Id})" />
        <rect class="SVGElement" width="13" height="4" x="77" y="46" fill="url(#diagonalHatch_Hopper)" />
        <path class="SVGElement SVGElement-Border" d="m90,46 0,4 -13,0"
              clip-path="url(#SVGBorderPath_Hopper)" fill="none" />
    </svg>
</div>
</div>
```

Different svg element types can be used to draw a picture. “SVGElement” should be used for all the **class** elements,. It provides default styles of the HMI Process Library, for example, stroke color, width, etc. The “SVGElement-Border” **class** can be added when the stroke width should be doubled. This is generally the case when the stroke coincides with the border of the SVG viewBox or the <clipPath> element, which is set as **clip-path** attribute for this element.

More information on drawing SVGs can be found here: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorials/SVG_from_scratch.

6 Appendix

6.1 Glossary

Abbreviation/Term	Meaning
DCS	Distributed control system
DataAssembly	This is an MTP-specific term which means the set of data that describes the physical or virtual entity represented by a control, i.e., pumps, vessels, valves, PID controllers, etc.
FB	Function block
HMI	Human machine interface
JS	JavaScript programming language
MTP	Module Type Package
OS	Operating system
P&ID	Piping & instrumentation diagram
PC	Personal computer
PID	Proportional integral derivative
PLC	Programmable logic controller
POL	Process orchestration layer
SCADA	Supervisory control and data acquisition
Symbol	In TwinCAT HMI, a Symbol is an addressable data point exposed by the HMI server. Most commonly this is a PLC variable exported via the TMC, but it can also be a dynamic or internal Symbol. See the Terminology [▶ 27] chapter for more detailed information. A Symbol is functionally analogous to a SCADA tag.
SymbolExpression	A representation of Symbol as a string, for example: %s%ADS.PLC1.GVL_MTP.V03%/s% or %s%ADS.PLC1.GVL_MTP.V03::TagName%/s% See the Terminology [▶ 27] chapter for detailed information
SymbolPath	A representation of a Symbol as a path, for example: ADS.PLC1.GVL_MTP.V03 See the Terminology [▶ 27] chapter for detailed information
TMC	TMC (symbol file) is an XML-based symbol table generated from the PLC project. It defines which PLC variables (names, data types, attributes) are published by the TwinCAT runtime and thus become addressable symbols for clients such as the TwinCAT HMI server (ADS/OPC UA).
UserControl	User control element in the TwinCAT HMI Engineering, which is represented by files with **.usercontrol* and **.usercontrol.json* extensions
VDI	Verein Deutscher Ingenieure (English: Association of German Engineers)

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

Excel, IntelliSense, Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

More Information:
www.beckhoff.com/mtp

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com