SIEMENS

Tips and tricks for configuring faceplates with WinCC Unified

SIMATIC WinCC Unified V18 / Unified Comfort Panels

https://support.industry.siemens.com/cs/ww/en/view/109812366

Siemens
Industry
Online
Support

Legal information

Table of contents

Legal information ... 2

1 Introduction ... 5

1.1 Overview ... 5

1.2 Requirements ... 5

1.3 Components used ... 5

2 Faceplate in faceplate ... 7

2.1 General remarks ... 7

2.2 Lowest device version ... 7

2.3 Example – Nested faceplates ... 8
2.3.1 Creation of User Data Types in the controller ... 9
2.3.2 Transfer the structure to a data block ... 11
2.3.3 Create HMI tags ... 12
2.3.4 Create faceplates ... 14
2.3.4.1 Pop-up faceplate "Temp_Sensor_DETAIL" ... 14
Configure the tag interface ... 17
2.3.4.2 Inner faceplate "Temp_Sensor" ... 18
Configure the tag interface ... 20
2.3.4.3 Main faceplate "Motor" ... 21
Configure the tag interface ... 23

3 Engineering ... 25

3.1 The "Configuration string" data type ... 25

3.2 Translation of texts in connection with faceplates ... 26

3.3 Changing visibility of the screen layers in faceplates ... 31

3.4 Replacing the version used in the project ... 32

3.5 Sharing faceplates via global libraries ... 34

3.6 Faceplates in pop-up form ... 34
3.6.1 Open faceplate as a pop-up ... 36
3.6.2 Modify faceplate interconnection in the screen ... 40
3.6.3 Open a faceplate from a faceplate ... 42
3.6.4 Close a faceplate ... 43
3.6.5 The Invisible parameter ... 43
3.6.6 The "IndependentWindow" parameter ... 44
3.6.7 Parent function ... 45
3.6.8 Header ... 46

3.7 Examples with the "IndependentWindow" parameter ... 48
3.7.1 Launching in (main) screen ... 49
3.7.1.1 Absolute and persistent (screen, large) ... 49
3.7.1.2 Relative and temporary (screen, large) ... 50
3.7.1.3 Relative and persistent (screen, small) ... 51
3.7.1.4 Relative and temporary (screen, small) ... 52
3.7.2 Launch in screen window ... 53
3.7.2.1 Absolute and persistent (main screen) ... 53
3.7.2.2 Relative and temporary (main screen) ... 54
3.7.2.3 Absolute and persistent (screen window) ... 55
3.7.2.4 Relative and temporary (screen window) ... 56

4 Appendix ... 57

4.1 Service and support ... 57

4.2 Industry Mall 58
4.3 Links and literature 58
4.4 Change documentation 58

1 Introduction

1.1 Overview

Standardization plays an ever greater role in today's industrial environments. Particularly with respect to the engineering of plants and machinery, requirements for a unified operator and control concept have risen steeply. One reason for this lies in short training times for plant operators, in addition to high operator safety.

A unified operator and control concept is also important for plant maintenance. Such an approach can reduce downtimes in case of faults, maintenance work or when extending the plant. Configurable functions such as Drive On/Off, Automatic/Manual mode switchover and the like come up again and again when designing serial machines. These functions usually only have to be adapted to the specific machine in question and its control tags. This is a good opportunity for the configuration engineer to take advantage of prefabricated objects to save time and money.

When it comes to HMIs, you can use so-called "faceplates" for this purpose.

In this document you will learn how to configure and apply various use cases and the associated configuration options for faceplates in the WinCC Unified environment.

1.2 Requirements

For increased clarity in the application example, we recommend the following two Online Support articles, or SITRAIN courses.

These can familiarize you with general handling of WinCC Unified and faceplates.

- Document SIMATIC HMI WinCC Unified Getting Started
- Application example SIMATIC WinCC Unified - Tips and Tricks for Scripting (JavaScript)
- SITRAIN system course: WinCC Unified & Unified Comfort Panels (Article ID: 109773211)
- SITRAIN advanced course: SIMATIC WinCC Unified for PC systems (Article ID: 109781323)

1.3 Components used

The application example was created using the following software components:

Table 1-1

Component	Quantity	Item number	Note
SIMATIC WinCC Unified V18 PC Engineering (TIA Portal)	1	6AV2153-2FB01-8LA5	Engineering system (any license size)
SIMATIC WinCC Unified V18 PC Runtime	1	6AV2154-2FB01-8BA0	PC runtime (any license size)

This application example consists of the following components:

Table 1-2

Component	File name	Note
This document	109812366_Unified_FaceplatesTT_en_V10.docx	
TIA Portal Project	109812366_Unified_FaceplatesTT_V18.zip	Created with V18

2 Faceplate in faceplate

2.1 General remarks

As of TIA Portal V18, you can nest faceplates inside of other faceplates and hierarchically inherit the associated interfaces. You can map User Data Types (UDTs) from a controller and their internal nested structures identically within the data structures of the faceplates, thus allowing you to easily create more complex configurations. The emphasis here is on standardization.

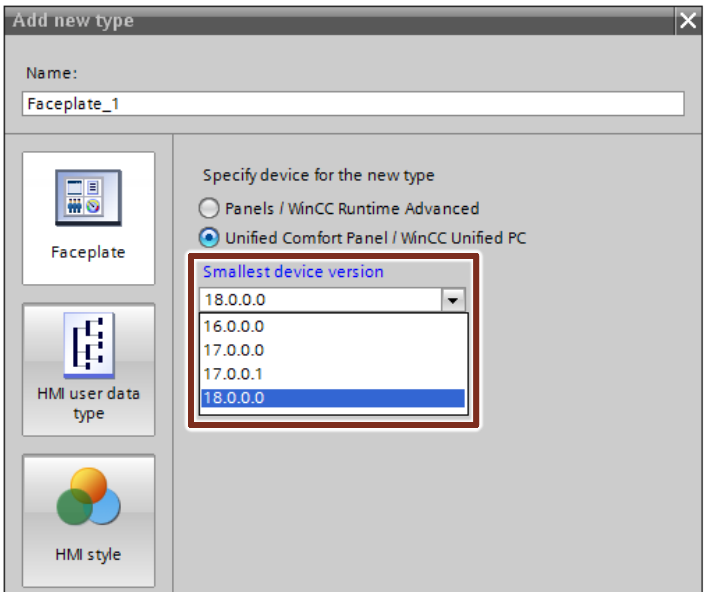
Note There is no technical limit to how deep you can nest faceplates.
We recommend you not exceed a maximum depth of eight (8) faceplates.

2.2 Lowest device version

TIA Portal V18 lets you define the lowest runtime version that must be installed on an HMI device in order to ensure your faceplates (WinCC Unified) render properly. Since TIA Portal V16, new features for WinCC Unified faceplates have been introduced with each version. Since these runtime functions are not backwards compatible, it is necessary to run the right runtime version on the end device if you are using certain functions.

To ensure this, when creating a new faceplate type you can use the "Smallest device version" dropdown menu to set the lowest runtime version needed for error-free operation.
This is typically the version in which the feature was introduced.

Figure 2-1



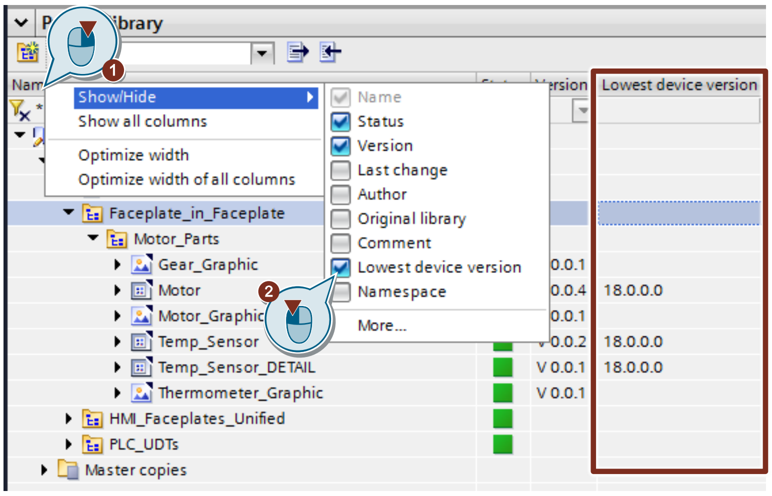
Name: Faceplate_1

Specify device for the new type
[✓ Unified Comfort Panel / WinCC Unified PC]

Smallest device version: 18.0.0.0

To display the configured lowest device version in the library, right-click on the column header (1) and, under the "Show/Hide" entry, select the corresponding option (2).

Figure 2-2



Name	Version	Lowest device version
Gear_Graphic	V 0.0.1	
Motor	V 0.0.4	18.0.0.0
Motor_Graphic	V 0.0.1	
Temp_Sensor	V 0.0.2	18.0.0.0
Temp_Sensor_DETAIL	V 0.0.1	18.0.0.0
Thermometer_Graphic	V 0.0.1	

Note

Further information on the correlation between features and versions can be found in the TIA Portal V18 manual in the chapter entitled "Lowest device version of a faceplate type" (https://support.industry.siemens.com/cs/de/en/view/109813308/160875372683).

2.3 Example – Nested faceplates

In this example we would like to nest two faceplates – one inside the other – and open another faceplate as a pop-up from one faceplate.
For illustration we will use a motor (outer faceplate) with its temperature sensor (inner faceplate) along with a detail view on the specific temperatures (pop-up faceplate).

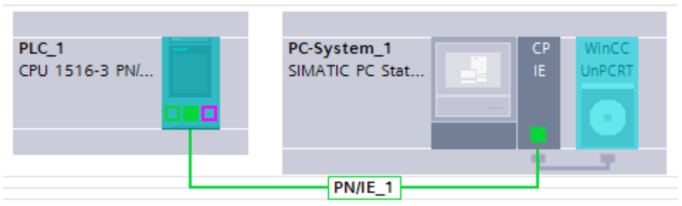
Requirements

This example will not address the creation and setup of the necessary devices and connections in TIA Portal.
The following requirements must be met in order to follow along with the example:

- A project has been created and opened in the Project view;
- A controller has been created (here: CPU 1516-3 PN/DP (6ES7 516-3AN00-0AB0, V1.8);
- A Unified HMI device has been created (here: SIMATIC WinCC Unified PC Station (6AV2 155-xxxxx-xxxx, Version 18.0.0.0, with "IE General" communication module));

- A network connection (PN/IE) has been created between the two devices;
- And an HMI connection has been created between the two devices.

Figure 2-3



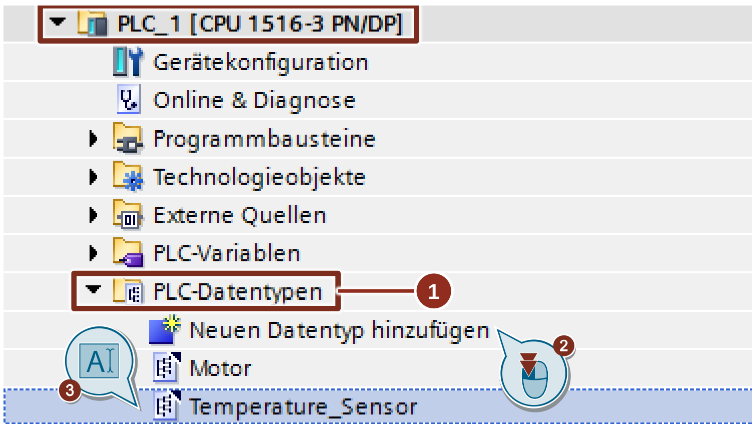
2.3.1 Creation of User Data Types in the controller

Temperature sensor

Since the User Data Types also need to be nested, first create the User Data Type for the inner faceplate (temperature sensor).

1. Navigate to the "PLC data types" folder in the controller (PLC).
2. Double-click on "Add new data type" to create a new User Data Type. It will open in the editor automatically. Give the User Data Type a meaningful name ("Temperature_Sensor").

Figure 2-4

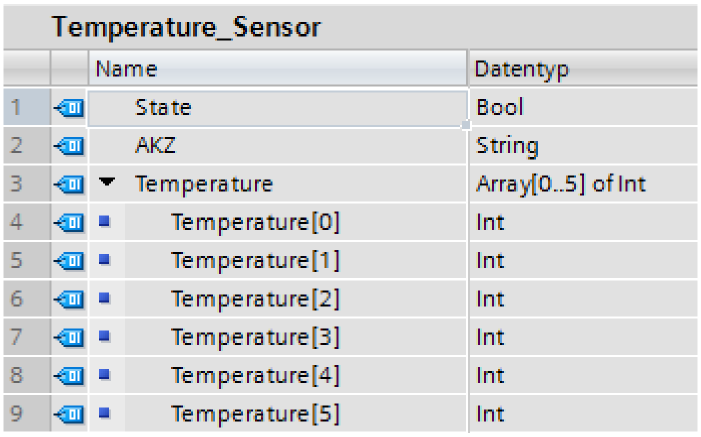


3. In the data type editor, create the following data structure via "Add":

Table 2-1

Name	Data type
State	Bool
AKZ	String
Temperature	Array[0..5] of Int

Figure 2-5



Temperature_Sensor

		Name	Datentyp
1		State	Bool
2		AKZ	String
3		▼ Temperature	Array[0..5] of Int
4		Temperature[0]	Int
5		Temperature[1]	Int
6		Temperature[2]	Int
7		Temperature[3]	Int
8		Temperature[4]	Int
9		Temperature[5]	Int

Motor

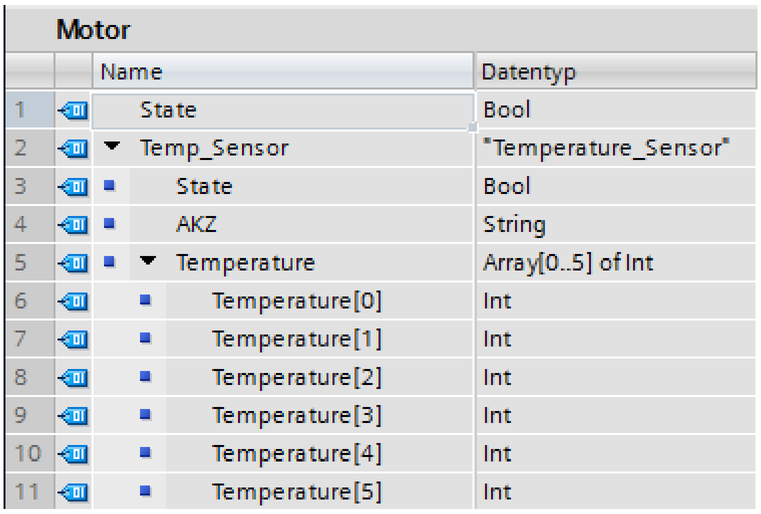
Repeat steps 1-3 from the previous section (temperature sensor), name the data type "Motor" and create the following data structure:

Table 2-2

Name	Data type
State	Bool
Temp_Sensor	"Temperature_Sensor"

With the data structure "Temperature_Sensor", you will embed the data structure of the previously created User Data Type (Temperature_Sensor) as its own element and thereby create a nested structure of the two User Data Types.

Figure 2-6



Motor

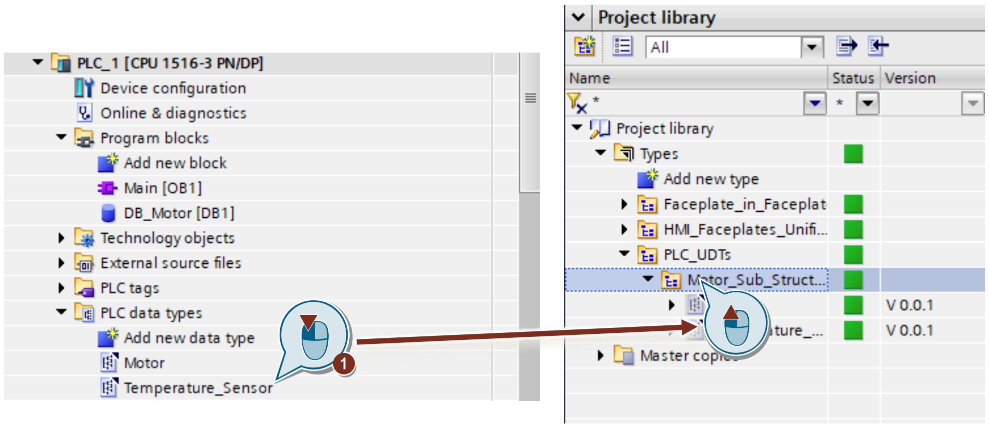
		Name	Datentyp
1		State	Bool
2		▼ Temp_Sensor	"Temperature_Sensor"
3		State	Bool
4		AKZ	String
5		▼ Temperature	Array[0..5] of Int
6		Temperature[0]	Int
7		Temperature[1]	Int
8		Temperature[2]	Int
9		Temperature[3]	Int
10		Temperature[4]	Int
11		Temperature[5]	Int

Create types in the project library

To also use the created User Data Types in connection with HMI objects (such as faceplates), you must version them in the project library.

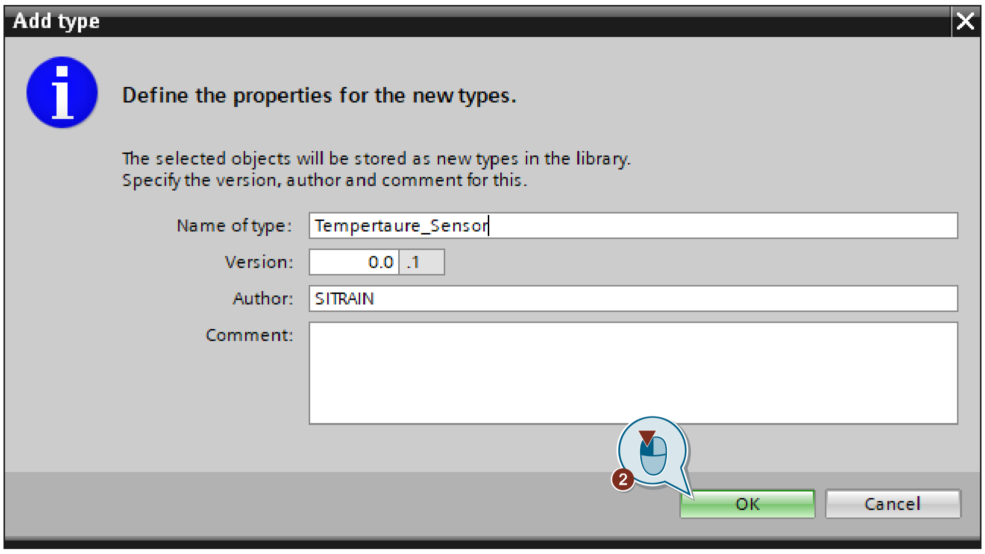
1. Select the User Data Type "Temperature_Sensor" and drag & drop it into the "Types" folder in the project library (here: subfolder "PLC_UDTs > Motor_Sub_Structures")

Figure 2-7



2. The "Add type" dialog opens. Confirm this by clicking "OK".

Figure 2-8



Define the properties for the new types.

The selected objects will be stored as new types in the library.
Specify the version, author and comment for this.

Name of type: Tempertaure_Sensor
Version: 0.0 .1
Author: SITRAIN
Comment:

3. Repeat steps 1 and 2 for the User Data Type "Motor".

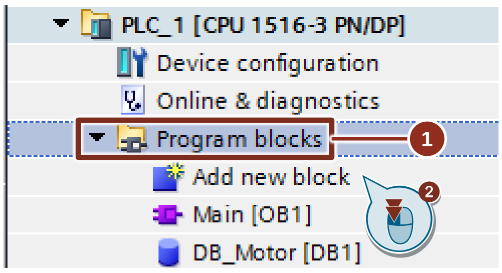
Result

You have created a nested data structure within the controller and versioned it in the project library. In the next step you will transfer this structure to a data block. The data block is needed for communication with the HMI device.

2.3.2 Transfer the structure to a data block

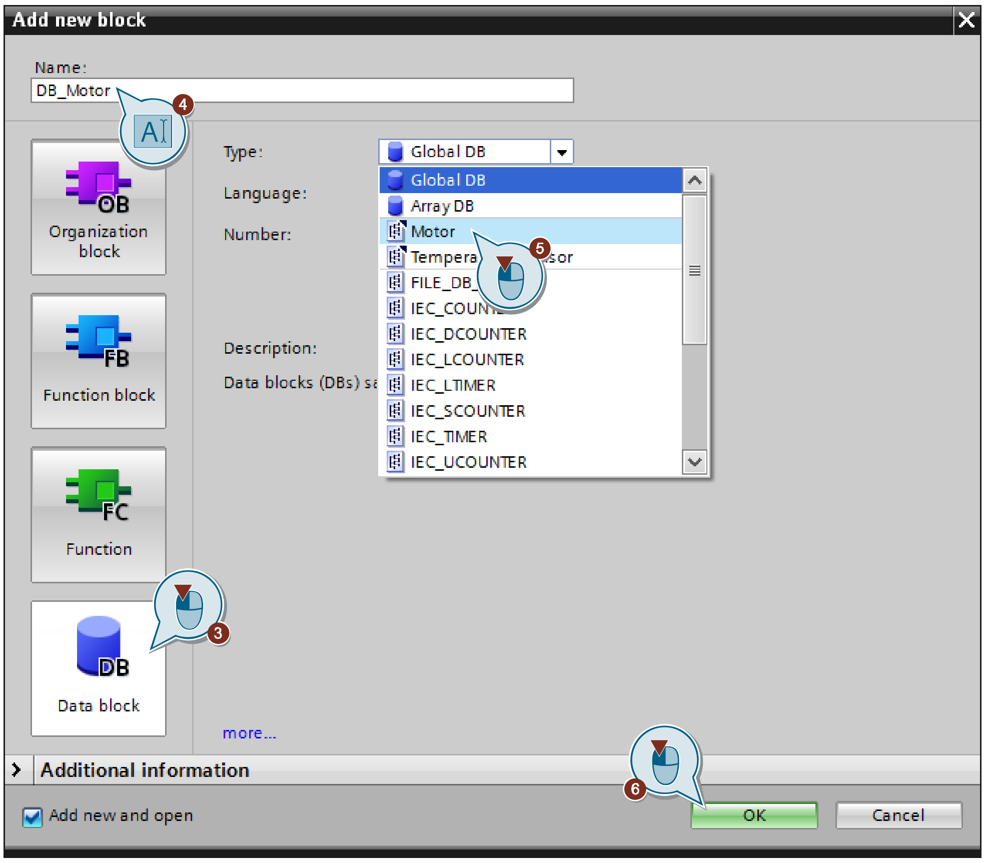
1. Navigate to the "Program blocks" folder in the controller.
2. Double click on "Add new block" to open the dialog of the same name where you can create a program block.

Figure 2-9



3. Select the "Data block" type in the dialog.
4. Change the name of the data block ("DB_Motor").
5. Select the type "Motor" from the dropdown menu.
6. Click "OK" to generate the data block.

Figure 2-10



Result

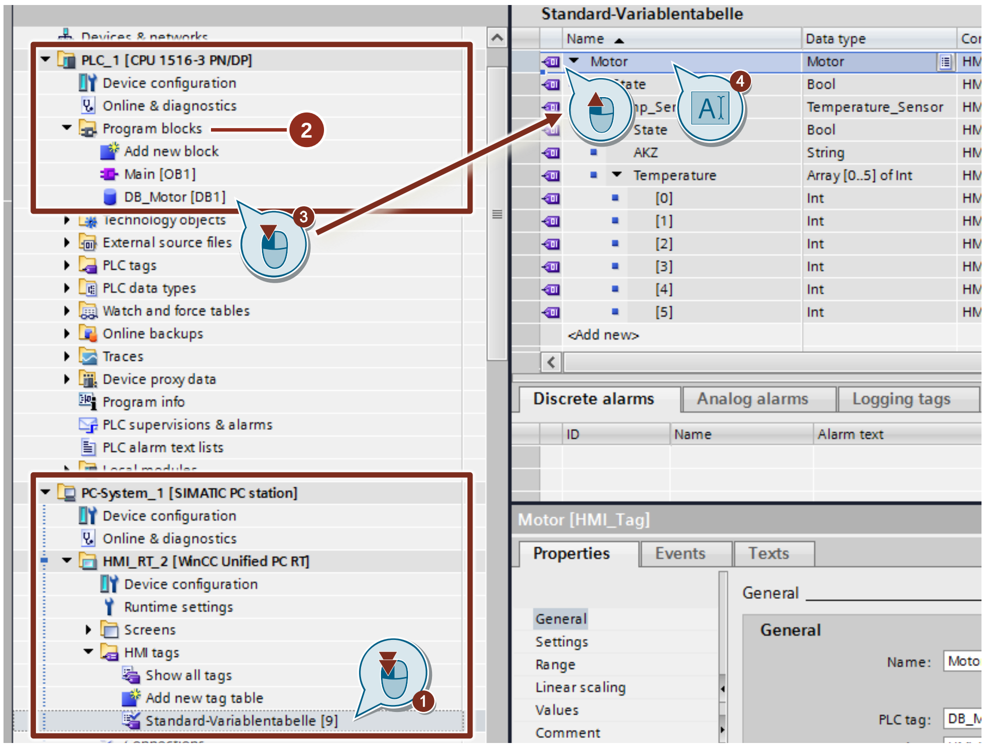
You have created the PLC data block "DB_Motor" with the structure of the User Data Type "Motor". In the next step you will create HMI tags for the same structure and generate a connection to the data block, which is needed for communication between the devices.

2.3.3 Create HMI tags

In this step we create the HMI tags that are linked to the interfaces of the faceplates and which facilitate communication between HMI and PLC.

1. In the HMI device, navigate to the "HMI tags" folder and double-click to open the "Standard tag table".
2. Navigate to the "Program blocks" folder in the PLC.
3. Drag the data block "DB_Motor" and drop it into the open Standard tag table.
4. The structure of the data block will pass identically to the HMI tags. Change of the tag structure from "DB_Motor" to "Motor".

Figure 2-11



Standard-Variablentabelle

Name	Data type	Co
Motor	Motor	HM
tate	Bool	HM
p_Ser	Temperature_Sensor	HM
State	Bool	HM
AKZ	String	HM
Temperature	Array [0..5] of Int	HM
[0]	Int	HM
[1]	Int	HM
[2]	Int	HM
[3]	Int	HM
[4]	Int	HM
[5]	Int	HM
<Add new>		

Result

You transferred the data structure of the PLC data block to HMI tags and created a connection between the PLC tags and the HMI tags. In the next step, you will create and configure the faceplates.

2.3.4 Create faceplates

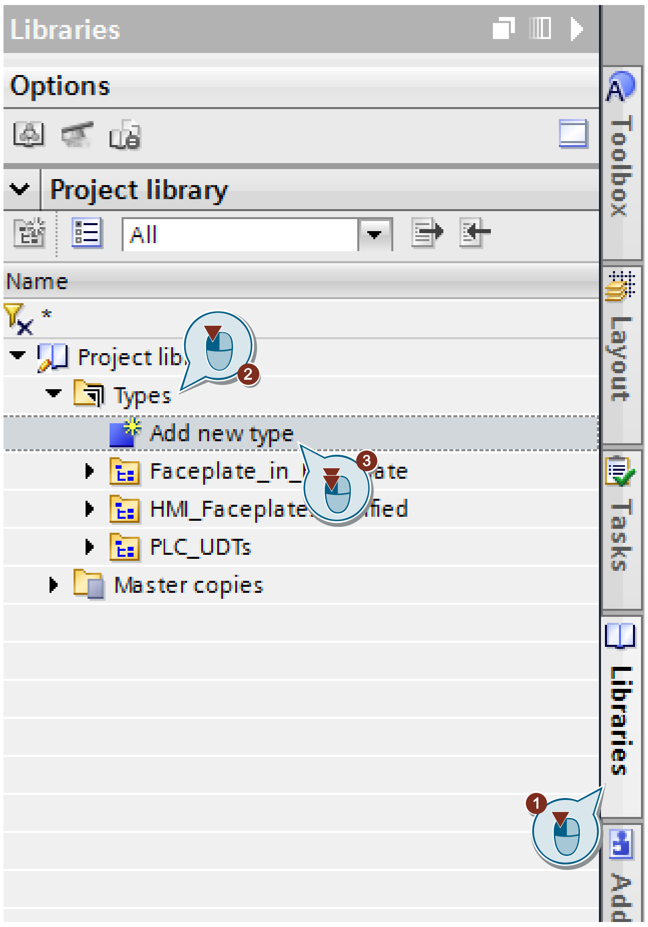
In this section you will create the faceplates, configure them and nest them inside of each other. Here you will work "from the inside out", starting from the "inner" faceplate (pop-up faceplate, temperature details) and moving to the embedded faceplate (temperature sensor) (toward the motor).

2.3.4.1 Pop-up faceplate "Temp_Sensor_DETAIL"

Create the faceplate type

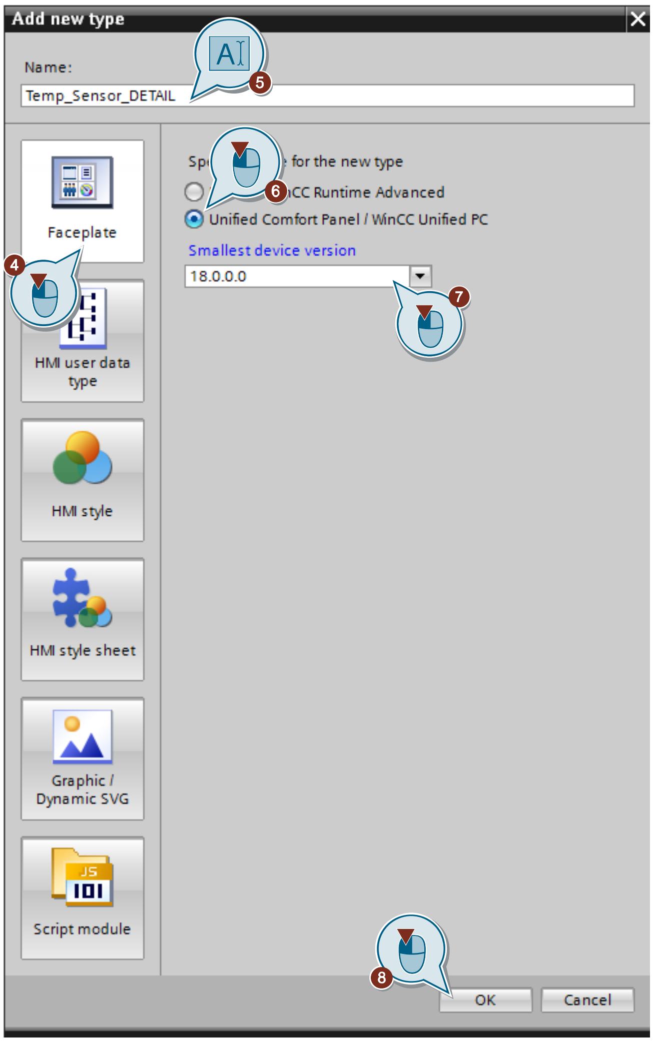
1. Switch to the "Libraries" tab.
2. Navigate to the folder "Project library > Types".
3. Double-click on "Add new type" to open the editor of the same name.

Figure 2-12



4. In the "Add new type" dialog, select the "HMI faceplate" type.
5. Change the name to "Temp_Sensor_DETAIL".
6. Select "Unified Comfort Panels / WinCC Unified PC" as the device.
7. Select "18.0.0.0" for the lowest device version.
8. Click "OK" again to confirm your selection. The faceplate will be generated and opened in the editor.

Figure 2-13



Name: Temp_Sensor_DETAIL

Sp... for the new type
...CC Runtime Advanced
(●) Unified Comfort Panel / WinCC Unified PC

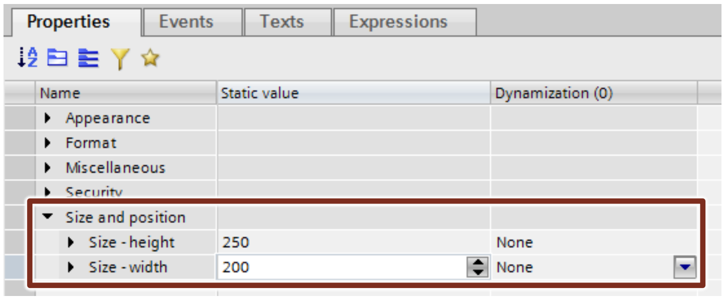
Smallest device version: 18.0.0.0

Change the visualization

In the faceplate properties, change the size to the following values:

- Size – width: 200
- Size – height: 250

Figure 2-14



Name	Static value	Dynamization (0)
Appearance		
Format		
Miscellaneous		
Security		
Size and position		
Size - height	250	None
Size - width	200	None

Now add the basic objects listed in Table 2-3 and use them to configure the following visualization:

Figure 2-15

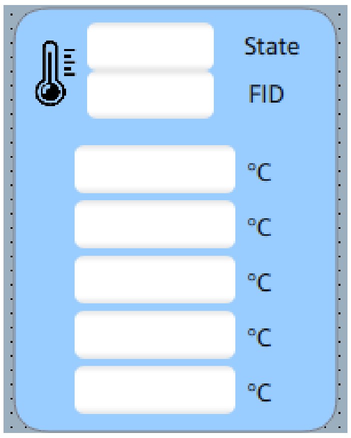


Table 2-3

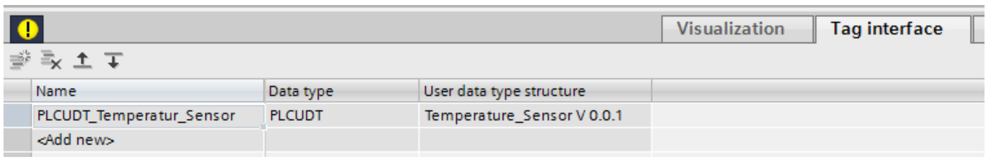
Basic object	Quantity	Remarks
Rectangle	1	Background color: 153; 204; 255 Corner radius: 20
I/O field	7	Mode: Output
Text field	7	-
Graphic view	1	Graphic: "Thermometer_Graphic V 0.0.1" (found in the application example library)

Configure the tag interface

Change to the "Tag interface" tab, click "<Add>" to create a new interface and assign the following parameters:

- Name: "PLCUDT_Temperatur_Sensor"
- Data type: "PLCUDT"
- User Data Type structure: "Temperature_Sensor V 0.0.1"

Figure 2-16

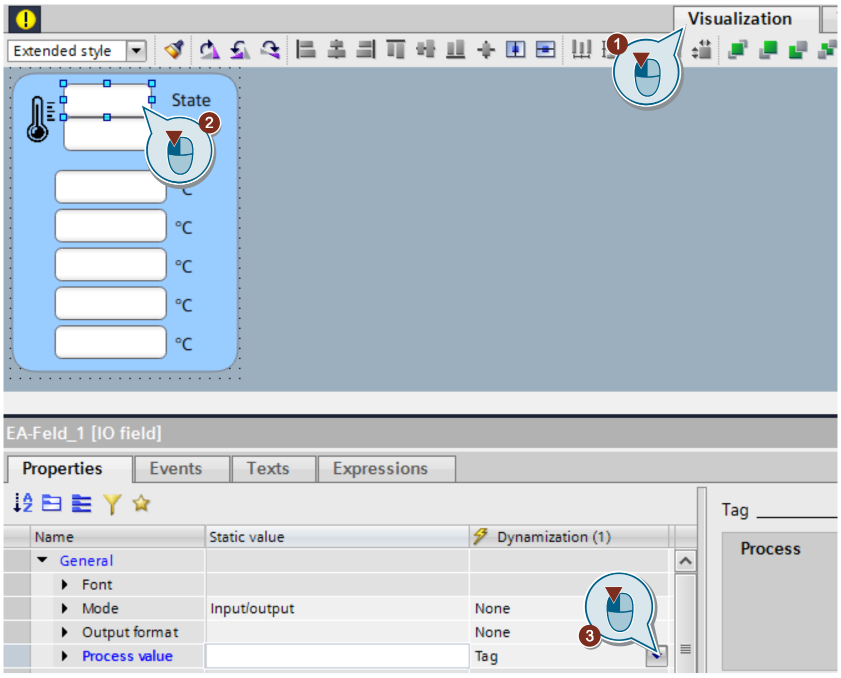


Name	Data type	User data type structure
PLCUDT_Temperatur_Sensor	PLCUDT	Temperature_Sensor V 0.0.1
<Add new>		

Link visualization and interface

1. Change back to the "Visualization" tab.
2. Select the top IO field.
3. In the settings, navigate to "General > Process value" and change the dynamization to "Tag".

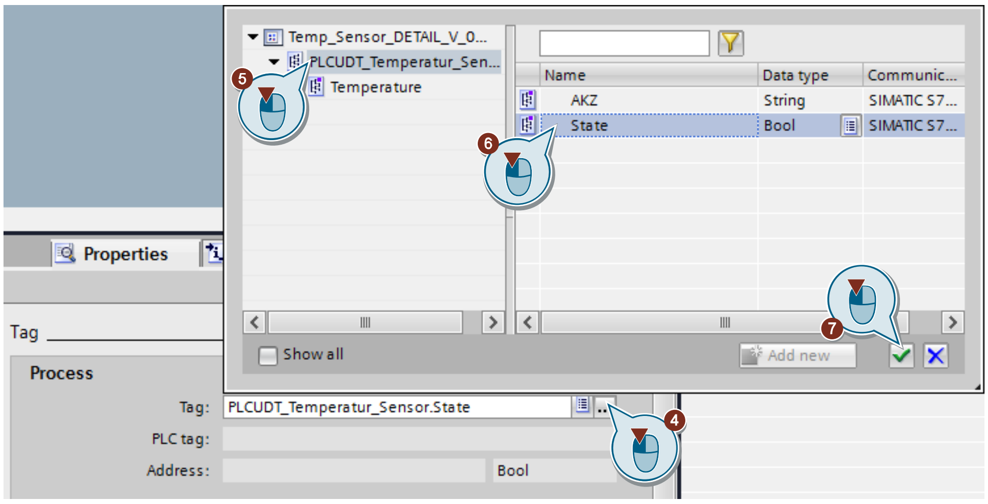
Figure 2-17



Name	Static value	Dynamization (1)
General		
Font		
Mode	Input/output	None
Output format		None
Process value		Tag

4. In the "Tag > Process > Tag" area, open the selector window for the tag you wish to link.
5. In the left pane, navigate to "PLCUDT_Temperatur_Sensor" and select it.
6. Select "State" in the right pane.
7. Confirm your selection by clicking on the checkbox.

Figure 2-18



8. Repeat steps 2 to 7 for all remaining IO fields.

   For this, use the tags "AKZ" and "Temperature[0]" through "Temperature[4]" (the latter are located in the higher-level folder "Temperature").

   Then, approve the faceplate version, thereby concluding the editing process.

2.3.4.2 Inner faceplate "Temp_Sensor"

Create the faceplate type

1. In the library, double-click on "Add new type" to open the editor of the same name and add a new faceplate.
2. In the "Add new type" dialog, select the "HMI faceplate" type.
3. Change the name to "Temp_Sensor".
4. Select "Unified Comfort Panels / WinCC Unified PC" as the device.
5. Select "18.0.0.0" for the lowest device version.
6. Click "OK" again to confirm your selection. The faceplate will be generated and opened in the editor.

Change the visualization

In the faceplate properties, change the size to the following values:

- Size – width: 200
- Size – height: 50

Now add the basic objects listed in Table 2-4 and use them to configure the following visualization:

Figure 2-19

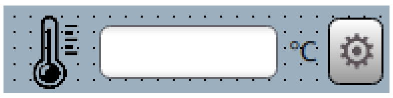


Table 2-4

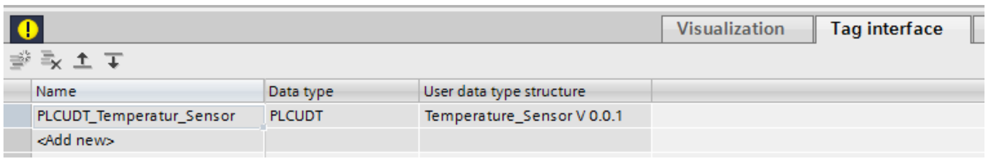
Basic object	Quantity	Remarks
Graphic view	1	Graphic: "Thermometer_Graphic V 0.0.1" (found in the application example library)
I/O field	1	Mode: Output
Text field	1	-
Button	1	Graphic: "Gear_Graphic V 0.0.1" (found in the application example library)

Configure the tag interface

Change to the "Tag interface" tab, click "<Add>" to create a new interface and assign the following parameters:

- Name: "PLCUDT_Temperatur_Sensor"
- Data type: "PLCUDT"
- User Data Type structure: "Temperature_Sensor V 0.0.1"

Figure 2-20



Name	Data type	User data type structure
PLCUDT_Temperatur_Sensor	PLCUDT	Temperature_Sensor V 0.0.1
<Add new>		

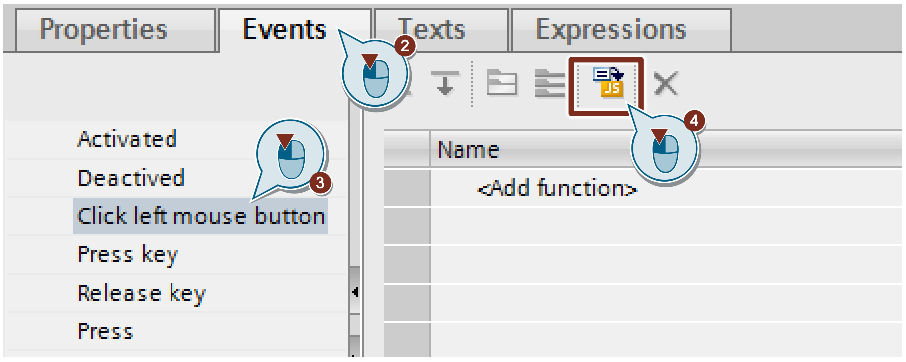
Link visualization and interface

1. Change back to the "Visualization" tab.
2. Select the IO field.
3. In the settings, navigate to "General > Process value" and change the dynamization to "Tag".
4. In the "Tag > Process > Tag" area, open the selector window for the tag you wish to link.
5. In the left pane, navigate to "Temperature" and select it.
6. Select "[0]" in the right pane.
7. Confirm your selection by clicking on the checkbox.

Set up the pop-up faceplate

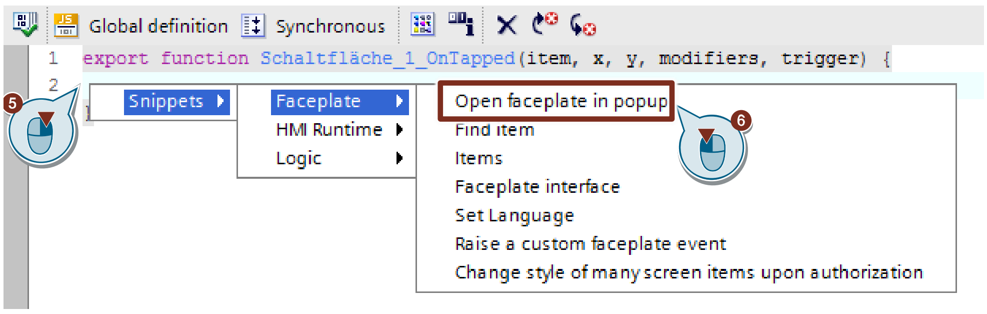
1. Select the button.
2. Switch to the "Events" tab.
3. Select "Click left mouse button."
4. Press the button shown to change the function into a script.

Figure 2-21



5. Open the selection menu for code snippets by right-clicking in an empty row ("2").
6. Navigate to "Snippets > Faceplate > Open faceplate in popup" and left-click to select the snippet.

Figure 2-22

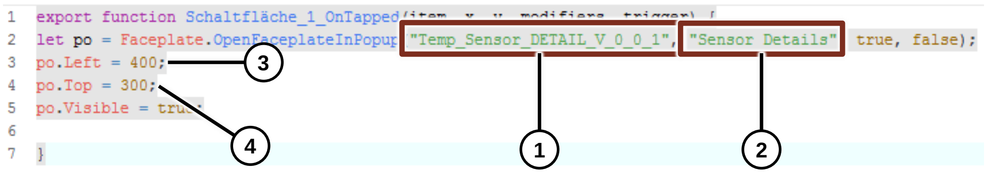


```
export function Schaltfläche_1_OnTapped(item, x, y, modifiers, trigger) {
```

Now adapt the parameters of the snippet as follows:

- Faceplate Type: "Temp_Sensor_DETAIL_V_0_0_1" (1)
- Title: "Sensor Details" (2)
- po.Left: 400 (3)
- po.Top: 300 (4)

Figure 2-23



```
export function Schaltfläche_1_OnTapped(item, x, y, modifiers, trigger) {
let po = Faceplate.OpenFaceplateInPopup("Temp_Sensor_DETAIL_V_0_0_1", "Sensor Details", true, false);
po.Left = 400;
po.Top = 300;
po.Visible = true;

}
```

Then, approve the faceplate version, thereby concluding the editing process.

Note For more information on using and configuring faceplates as pop-ups, refer to chapter 3.5 Faceplates as pop-ups.

2.3.4.3 Main faceplate "Motor"

Create the faceplate type

1. In the library, double-click on "Add new type" to open the editor of the same name and add a new faceplate.
2. In the "Add new type" dialog, select the "HMI faceplate" type.
3. Change the name to "Motor".
4. Select "Unified Comfort Panels / WinCC Unified PC" as the device.
5. Select "18.0.0.0" for the lowest device version.

6. Click "OK" again to confirm your selection. The faceplate will be generated and opened in the editor.

Change the visualization

In the faceplate properties, change the size to the following values:

- Size – width: 250
- Size – height: 250

Now add the basic objects listed in Table 2-5 and use them to configure the following visualization:

Figure 2-24

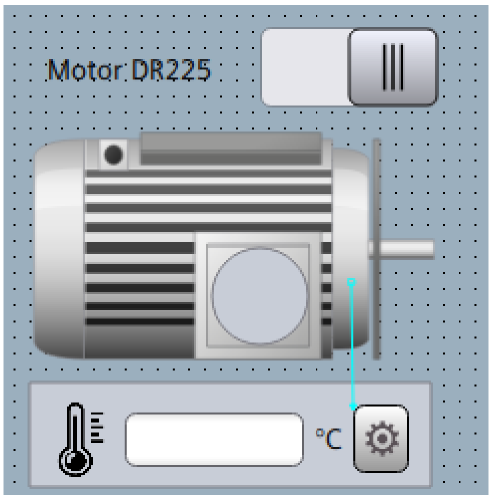


Table 2-5

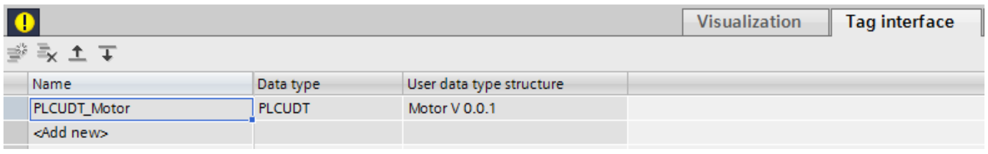
Basic object	Quantity	Remarks
Graphic view	1	Graphic: "Motor_Graphic V 0.0.1" (found in the application example library)
Circle	1	-
Text field	1	-
Switch	1	-
Line	1	Color: 0; 255; 255
Faceplate container	1	Drag and drop to add the faceplate type "Temp_Sensor" from the library

Configure the tag interface

Change to the "Tag interface" tab, click "<Add>" to create a new interface and assign the following parameters:

- Name: "PLCUDT_Motor"
- Data type: "PLCUDT"
- User Data Type structure: "Motor V 0.0.1"

Figure 2-25



Name	Data type	User data type structure
PLCUDT_Motor	PLCUDT	Motor V 0.0.1
<Add new>		

Link visualization and interface

Change back to the "Visualization" tab.

Switch

1. Select the switch.
2. In the settings, navigate to "General > State switch" and change the dynamization to "Tag".
3. In the "Tag > Process > Tag" area, open the selector window for the tag you wish to link.
4. In the left pane, navigate to "PLCUDT_Motor" and select it.
5. Select "State" in the right pane.
6. Confirm your selection by clicking on the checkbox.

Circle

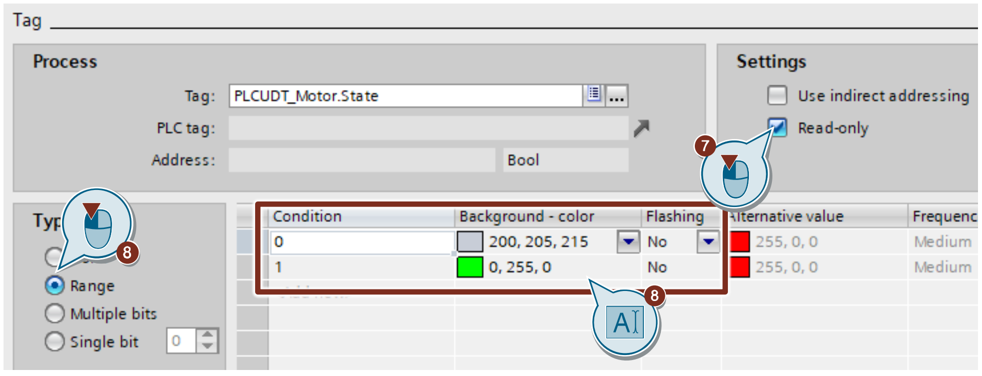
1. Select the circle.
2. In the settings, navigate to "Appearance > Background color" and change the dynamization to "Tag".
3. In the "Tag > Process > Tag" area, open the selector window for the tag you wish to link.
4. In the left pane, navigate to "PLCUDT_Motor" and select it.
5. Select "State" in the right pane.
6. Confirm your selection by clicking on the checkbox.

7. In the "Tag > Process > Settings" area, tick the box for "Read only".
8. In the "Tag > Process > Type" area, select the type "Area" and assign the following parameters:

Table 2-6

Condition	Background color	Flashing
0	200; 205; 215	No
1	0; 255; 0	No

Figure 2-26



Tag

Process

Tag: PLCUDT_Motor.State
PLC tag:
Address: Bool

Settings

☐ Use indirect addressing
☑ Read-only

Typ
● Range
○ Multiple bits
○ Single bit 0

Condition	Background - color	Flashing	lternative value	Frequenc
0	▨ 200, 205, 215	No	🟥 255, 0, 0	Medium
1	🟩 0, 255, 0	No	🟥 255, 0, 0	Medium

Then, approve the faceplate version, thereby concluding the editing process.

Result

You have created three faceplates, along with the associated User Data Types, and linked them with each other. You can now instance the "Motor" faceplate in a screen and, via its interface, link all information (tags) needed for the inner faceplates.

3 Engineering

3.1 The "Configuration string" data type

The "Configuration string" data type lets you pass ordinary text with an unlimited character length to a faceplate.

For example, you can use it to pass a filter parameter to an alarm view in a faceplate.

Example

In the following example, we would like to use a button to pass a predefined filter to an alarm view located in a faceplate. The filter is passed to the corresponding property in the form of a string and then evaluated by the alarm view.

The filter needs to display only alarms with priority 8 and with an ID higher than 28. The evaluated string is "Priority = 8 AND ID > 28" (without quotation marks).

Note Spelling is important in the passed string so that the alarm view can evaluate it correctly. To be on the safe side, you can create the filter beforehand directly in the alarm view and then copy the string there.

Faceplate preparation

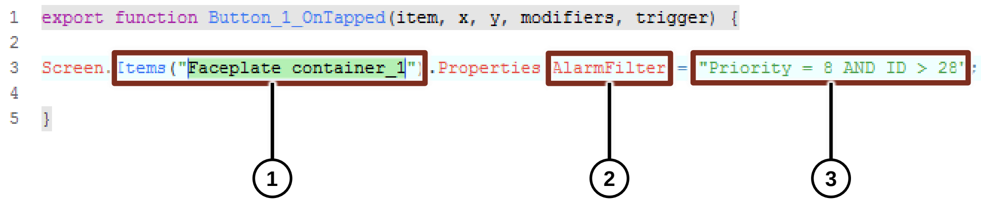
1. Create a new faceplate type that contains at least one alarm view.
2. Create a "properties interface" of data type "Configuration string".
3. For the "General > Filter" property of the alarm view, create a dynamization of type "properties interface" and assign the previously generated Configuration string interface.
4. Approve the faceplate version.

Screen preparations

1. Create an instance of the faceplate, created previously, in a screen.
2. Add a button to the screen.
3. Create a click event for the button, convert it to a script and enter the following code:

```
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {
Screen.Items("Faceplate container_1").Properties.AlarmFilter =
"Priority = 8 AND ID > 28";
}
```

Figure 3-1



```
export function Button_1_OnTapped(item, x, y, modifiers, trigger) {

Screen.Items("Faceplate container_1").Properties.AlarmFilter = "Priority = 8 AND ID > 28";

}
```

- Screen object (1)
- Property (2)
- Call parameters (3)

Result

When you click on the configured button during runtime, a filter will be sent to the alarm view and evaluated.

"Multilingual text" comparison

Use the "Multilingual text" data type to send texts (multilingual) to a faceplate. You can link it to the text property of any screen object that supports it. This lets you assign various texts to the screen objects of every instance of a faceplate and thereby avoid duplicated effort in configuring them individually.

All texts that you call with the "Multilingual text" interface are also listed in the project texts, where they can be edited.

The "Configuration string" data type is not suitable for passing to text properties. It should only be used for configuring/passing ordinary text parameters as described in the example above.

3.2 Translation of texts in connection with faceplates

If you wish to use faceplate texts in multiple languages, then it is not necessary to translate them and integrate them right in the configuration phase. You can later export your texts to an Excel file where you can translate them and finally re-import them to the faceplate. This process does not generate a new version of the faceplate.

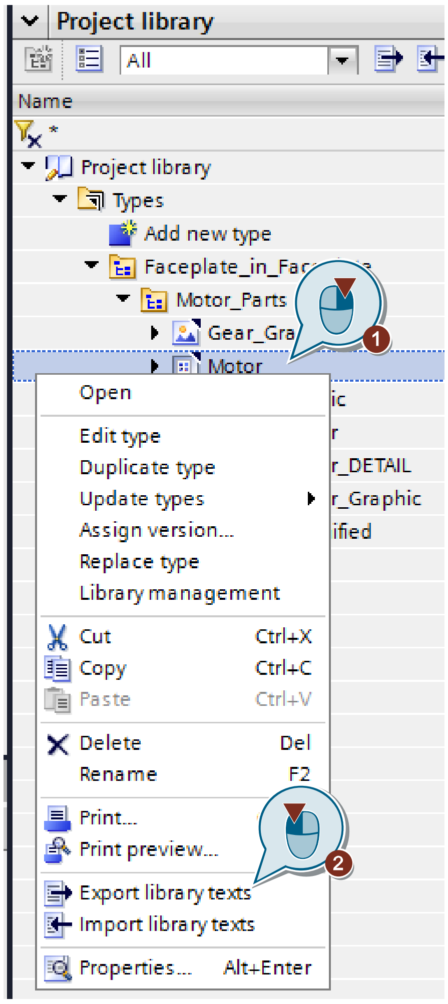
Note

Only the texts of the version that is set as the "Default" version will be exported. Set them to the desired version beforehand if necessary.

Example export

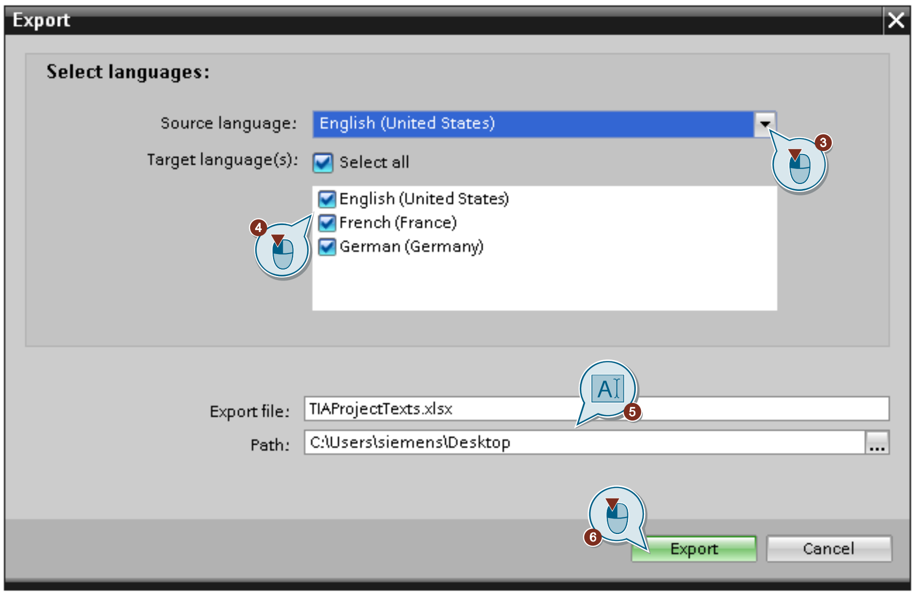
1. Ensure that the necessary languages are activated both under "Languages & resources > Project languages" and in the runtime settings under "Language & font".
2. Right-click to open the context menu of the corresponding faceplate type (the complete type must be selected, not a version) (1) and select "Export library texts" (2).

Figure 3-2



3. In the dialog that opens, select the source language (3) and the languages you wish to export (4).
4. Enter a name for the Excel file and select a save location (5).
5. Confirm the export by clicking "Export" (6).

Figure 3-3



Source language: English (United States)

Target language(s): ✓ Select all

✓ English (United States)
✓ French (France)
✓ German (Germany)

Export file: TIAProjectTexts.xlsx

Path: C:\Users\siemens\Desktop

6. Another dialog will confirm that the export was successful. You can close it by clicking "OK".

Result

You have successfully exported the texts of a faceplate to an Excel file. You can now use them for translations.

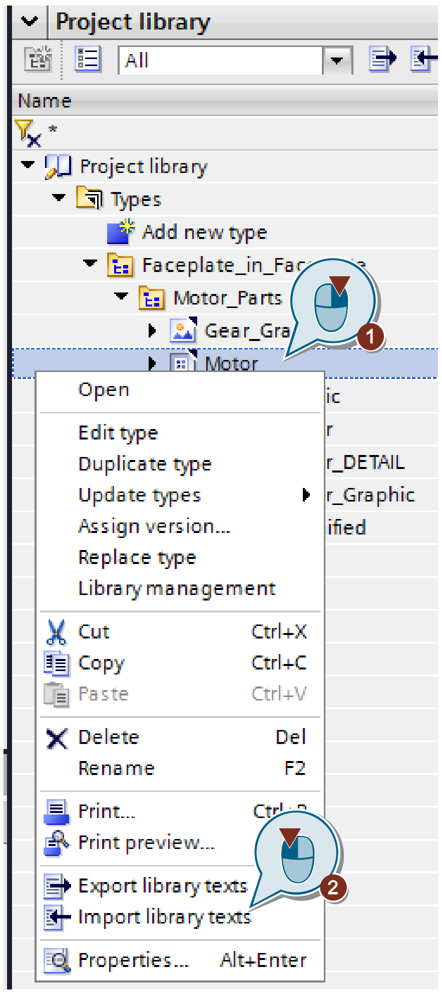
Example import

Requirements:

You have an Excel file for the corresponding faceplate and you have made changes (translations) in this file.

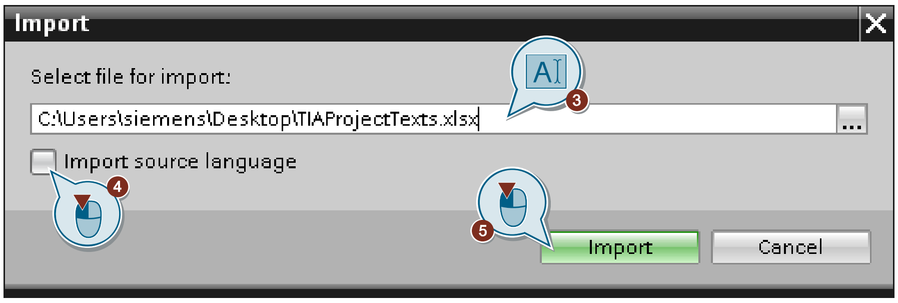
1. Right-click to open the context menu of the corresponding faceplate type (the complete type must be selected, not a version) (1) and select "Import library texts" (2).

Figure 3-4



2. In the dialog that opens, select the path of the file you wish to import (3).
3. If necessary, tick the box for the "Import source language" option (4). This is only necessary if changes also need to be made to the source language.
4. Confirm the process by clicking on "Import" (5).

Figure 3-5



C:\Users\siemens\Desktop\TIAProjectTexts.xlsx

5. Another dialog will confirm that the import was successful. You can close it by clicking "OK".

Result

You have successfully imported the (multilingual) texts of a faceplate from an Excel file. You can now use them in the faceplate.

3.3 Changing visibility of the screen layers in faceplates

You can show or hide single screen layers of a faceplate. This can be helpful if certain screen objects are not always necessary or if they only need to be displayed in special cases.
Using screen layers and their visibility property requires you to dynamize every screen object individually.

Although you can configure the methods in every event, changing the visibility is only possible during runtime. Thus, you cannot call up a faceplate with already hidden screen layers.

Note As a possible workaround, you can configure the switchover in the "Established" event of the faceplate, which will cause the effect to become active right when the faceplate is called up.

In addition, this function is only available within the faceplate. If you wish to access it from outside the faceplate, you must configure the transfer of the necessary data/information via a tag or properties interface.

For a status query, use

```
Faceplate.Layers("Layer_X").Visible == true
```

or

```
Faceplate.Layers("Layer_X").Visible == false
```

If you wish to set a value, use

```
Faceplate.Layers("Layer_X").Visible = true
```

or

```
Faceplate.Layers("Layer_X").Visible = false
```

Note Replace the placeholder "X" in the expression "Layer_X" with the corresponding number in the screen layer.

You can link this function with any of the common logic functions as desired.

3.4 Replacing the version used in the project

You have the option of changing the faceplate version used in the project, for example to test the functioning of different versions.
You can go forward or backward in the versioning.

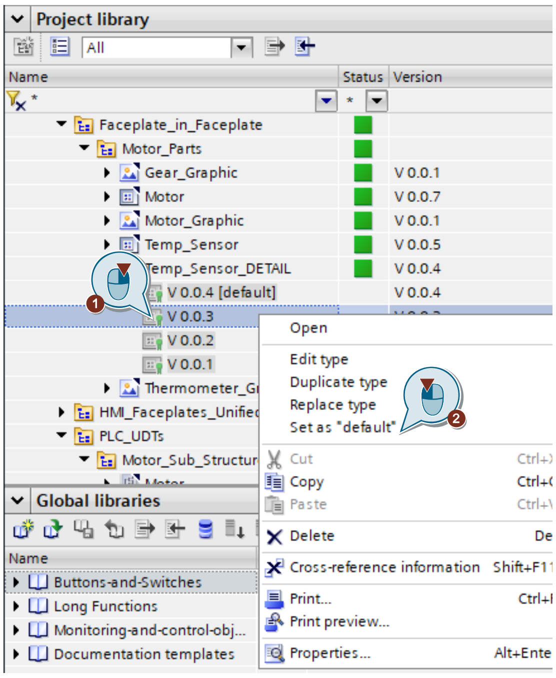
You can tell which version is currently being used because "[Default]" will come after the version number, for example "V 0.0.2 [Default]".

Example

In this example, you will change the default version of a faceplate from V 0.0.4 to V 0.0.3 and update the instance used in the project.

1. In the tree structure of the "Temp_Sensor_DETAIL" faceplate, select version V 0.0.3 and right-click to open the version's context menu (1).
2. Click on "Set as 'default'" (2). The "[Default]" suffix will move to this version.

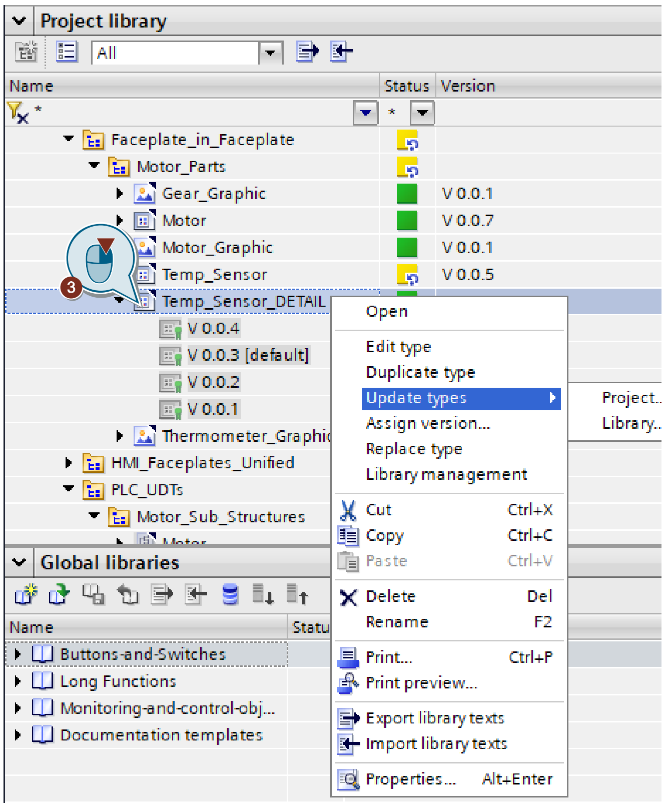
Figure 3-6



Name	Status	Version
Faceplate_in_Faceplate	🟩	
Motor_Parts	🟩	
Gear_Graphic	🟩	V 0.0.1
Motor	🟩	V 0.0.7
Motor_Graphic	🟩	V 0.0.1
Temp_Sensor	🟩	V 0.0.5
Temp_Sensor_DETAIL	🟩	V 0.0.4
V 0.0.4 [default]		V 0.0.4
V 0.0.3		
V 0.0.2		
V 0.0.1		
Thermometer_Gi		
HMI_Faceplates_Unified		
PLC_UDTs		
Motor_Sub_Structur		

3. Right-click to open the context menu of the faceplate type (3) and select "Update types > Project..." (4).

Figure 3-7

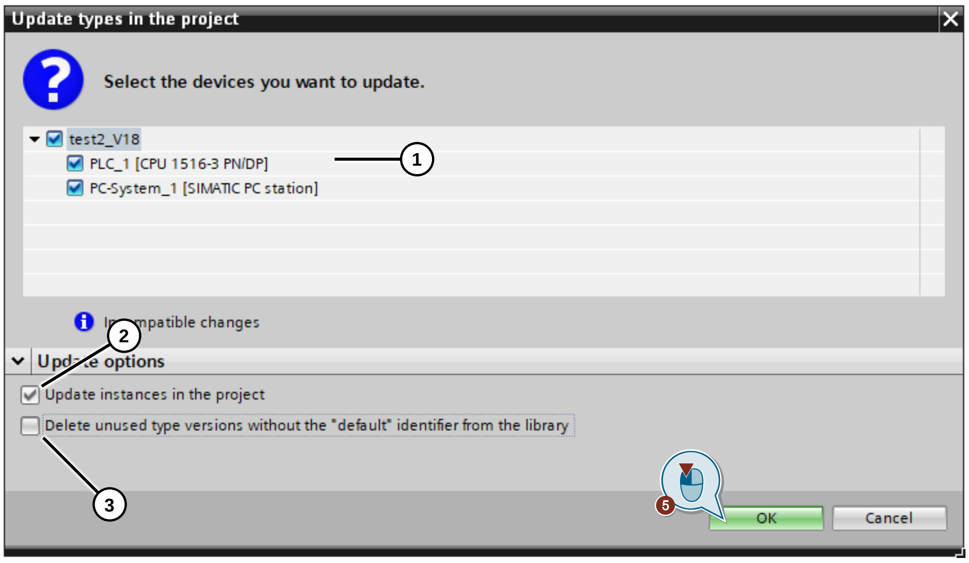


4. A dialog window ("Update types in the project") will open. Leave the default settings and confirm the selection by clicking "OK" (5).

Note

The dialog window lets you set (if possible) the devices that the changes will be pushed to (1), whether the instances used in the devices will be refreshed (2), and whether any other versions of the faceplate not marked as "Default" should be deleted from the library (3).

Figure 3-8


Update types in the project
Select the devices you want to update.
test2_V18
PLC_1 [CPU 1516-3 PN/DP]
PC-System_1 [SIMATIC PC station]
Incompatible changes
Update options
Update instances in the project
Delete unused type versions without the "default" identifier from the library
OK
Cancel
1
2
3
5

Result

You have changed the faceplate version used in the project from V 0.0.4 to V 0.0.3.

3.5 Sharing faceplates via global libraries

To use already existing faceplates with other users or in multiple projects, thus avoiding duplication of effort from creating them again, you can use global libraries to share faceplates.

You can copy existing faceplates from the "Types" folder in the project library and paste them into the folder of the same name in a global library, then save it. For example, if you save the global library on a network drive, all users with access to this network drive can retrieve the library and its contents at any time and copy it to the respective local project library as needed.

You can also save the global library locally and share it with others in other ways; or you can access it with another project if you need to, then reuse its contents (e.g. faceplates).

3.6 Faceplates in pop-up form

Besides their static configuration, faceplates can also be configured in a faceplate container as a pop-up window. This causes faceplates to lose their local link; they can be placed anywhere in the runtime, for example by dragging and dropping, or they can stay in the foreground on screen changes or closed if you do not need to see them anymore.

An appearance as a pop-up window can be configured either from a screen or from another faceplate. To do this, configure an event on a screen object and use the code snippet "OpenFaceplateInPopup".

These code snippets vary in a couple respects depending on whether you use "OpenFaceplateInPopup" in a screen or in a faceplate.

Overview

In this chapter, you will learn:

- How to open a faceplate as a pop-up and interconnect it with a script → chapter 3.6.1
- How to configure a faceplate in a screen and modify the interconnection in the runtime → chapter 3.6.2
- How to open a faceplate from a faceplate → chapter 3.6.3

Example

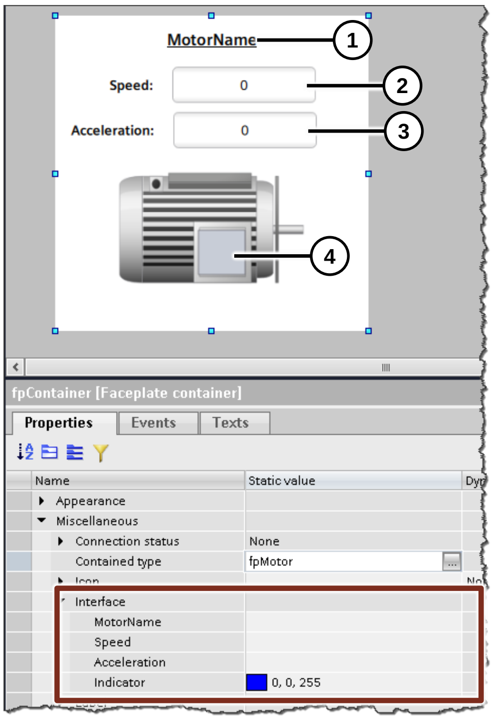
Note

The elements (faceplates, screens, tags, etc.) used in the following examples are used only for illustration and are NOT part of the example project.

For a better understanding, the interconnection will be explained below using the example of the "fpMotor" faceplate. It has:

- three tag interfaces ("MotorName" (1), "Speed" (2) and "Acceleration" (3)) and
- one properties interface ("Indicator" (4)).

Figure 3-9

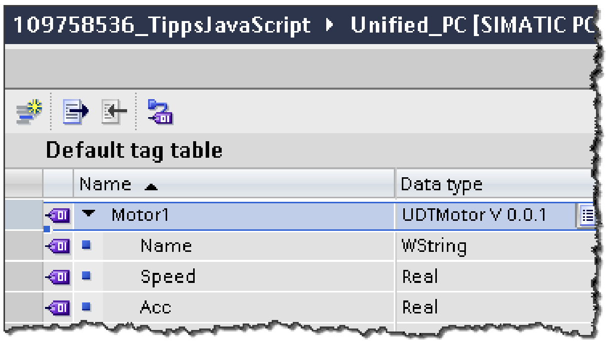


fpContainer [Faceplate container]

Name	Static value	Dy
Appearance		
Miscellaneous		
Connection status	None	
Contained type	fpMotor	
Icon		No
Interface		
MotorName		
Speed		
Acceleration		
Indicator	🟦 0, 0, 255	

This faceplate should be launched as a pop-up and interconnected with the UDT "UDTMotor".

Figure 3-10, "UDTMotor"



109758536_TippsJavaScript ▸ Unified_PC [SIMATIC PC

Default tag table

Name	Data type
▾ Motor1	UDTMotor V 0.0.1
Name	WString
Speed	Real
Acc	Real

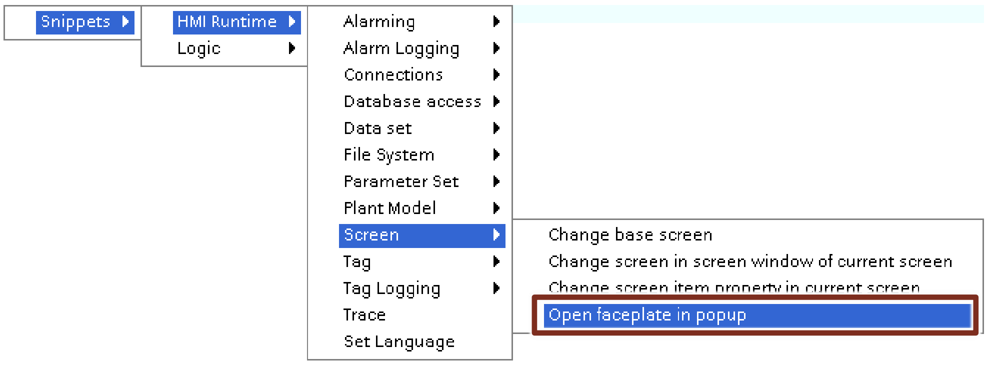
3.6.1 Open faceplate as a pop-up

In this example, we want to open the faceplate "fpMotor" as a pop-up via a button and; the interface must be interconnected accordingly.

Code snippet

The snippet "Open faceplate in popup" is available for launching faceplates.

Figure 3-11

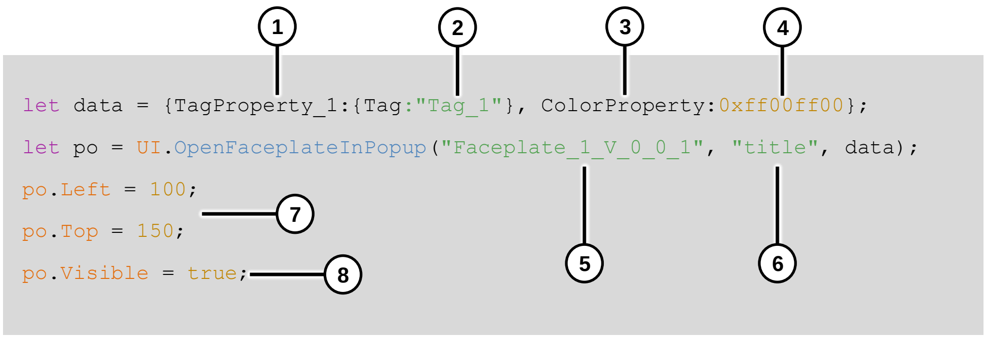


```
let data = {TagProperty_1:{Tag:"Tag_1"}, ColorProperty:0xff00ff00};
let po = UI.OpenFaceplateInPopup("Faceplate type_1", "title", data);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

The interface of the faceplate is composed of the following parts:

1. the interface tag name of the faceplate
2. the tag name of the HMI tag to be passed
3. the name of the interface property
4. the property value being passed
5. the faceplate name
6. the title of the pop-up window
7. the position of the pop-up window
8. the visibility of the pop-up window

Figure 3-12



```
let data = {TagProperty_1:{Tag:"Tag_1"}, ColorProperty:0xff00ff00};
let po = UI.OpenFaceplateInPopup("Faceplate_1_V_0_0_1", "title", data);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

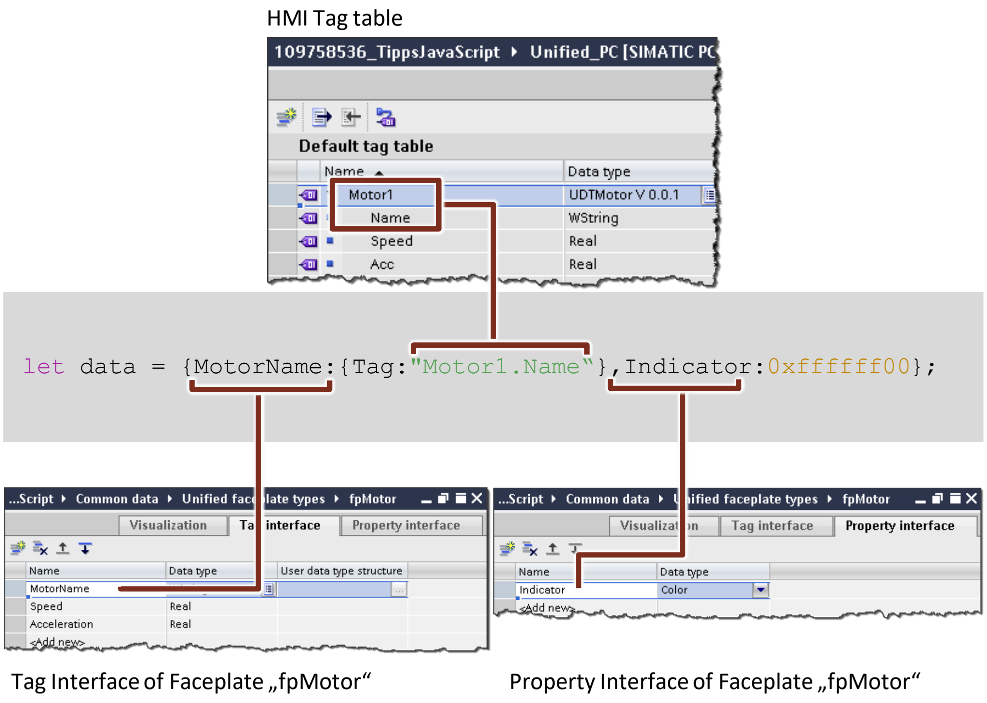
Note

You can find further information about the interface parameters and the optional parameters "parentScreen" and "Visibility" in the manual under the method "OpenFaceplateInPopup":

https://support.industry.siemens.com/cs/de/en/view/109813308/159974440971

The following interconnection comes from the "fpMotor" example and the "UDTMotor" that we wish to interconnect:

Figure 3-13


HMI Tag table
109758536_TippsJavaScript ▸ Unified_PC [SIMATIC PC
Default tag table
...Script ▸ Common data ▸ Unified faceplate types ▸ fpMotor
Visualization
Tag interface
Property interface
Tag Interface of Faceplate „fpMotor“
Property Interface of Faceplate „fpMotor“

Name	Data type
Motor1	UDTMotor V 0.0.1
Name	WString
Speed	Real
Acc	Real

```
let data = {MotorName:{Tag:"Motor1.Name"},Indicator:0xffffff00};
```

Name	Data type	User data type structure
MotorName		
Speed	Real	
Acceleration	Real	
<Add new>		

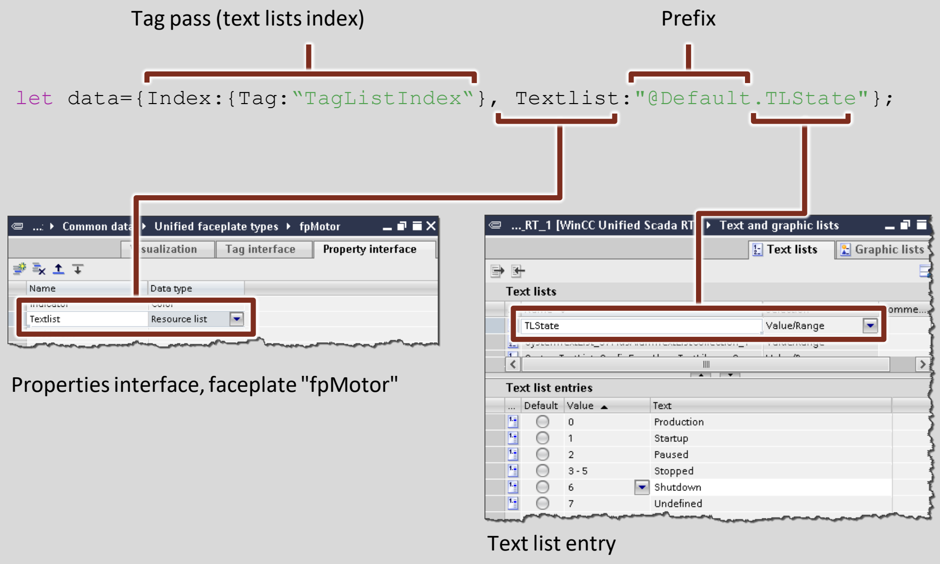
Name	Data type
Indicator	Color
<Add new>	

Note

If you pass a text list via the tag interface, you will also need another tag pass in order to pass the value (index) of the text list.

The call parameters for a text list itself are composed of the following information:

- Name of the properties interface (here: "Textlist")
- Prefix ("@Default.") + name of the text list (here: "TLState")


Tag pass (text lists index)
Prefix
… Common data › Unified faceplate types › fpMotor
sualization
Tag interface
Property interface
Name
Data type
Textlist
Resource list
Properties interface, faceplate "fpMotor"
…_RT_1 [WinCC Unified Scada RT] › Text and graphic lists
Text lists
Graphic lists
Text lists
TLState
Value/Range
Text list entries
Text list entry

```
let data={Index:{Tag:"TagListIndex"}, Textlist:"@Default.TLState"};
```

Default	Value	Text
○	0	Production
○	1	Startup
○	2	Paused
○	3 - 5	Stopped
○	6	Shutdown
○	7	Undefined

Passing graphic lists is currently not supported.

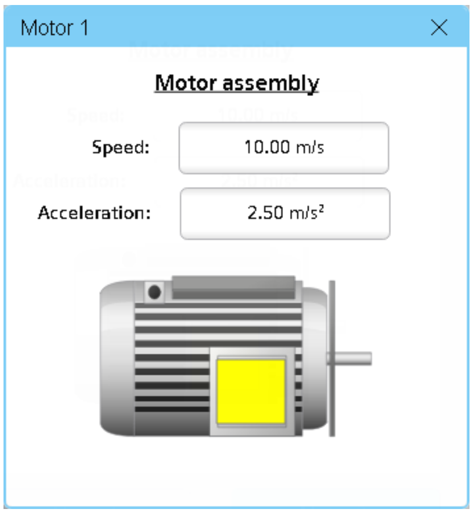
Result

The following overview shows the fully configured code snippet.

```
let data={MotorName:{Tag:"Motor1.Name"},Speed:{Tag:"Motor1.Speed"},
Acceleration:{Tag:"Motor1.Acc"},Indicator:0xffffff00};
let po = UI.OpenFaceplateInPopup("fpMotor", "Motor 1", data);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

Once the button has been pressed in the runtime, the faceplate will appear as a pop-up with the data that were passed.

Figure 3-14



Motor assembly

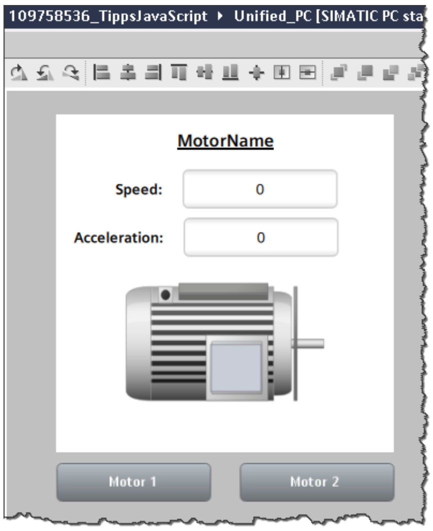
Speed: 10.00 m/s

Acceleration: 2.50 m/s²

3.6.2 Modify faceplate interconnection in the screen

In the second use case, we want to configure the faceplate "fpMotor" in the screen. Using two buttons, we want to change the interface in the runtime from UDT Motor 1 to UDT Motor 2.

Figure 3-15


109758536_TippsJavaScript ▸ Unified_PC [SIMATIC PC sta

MotorName

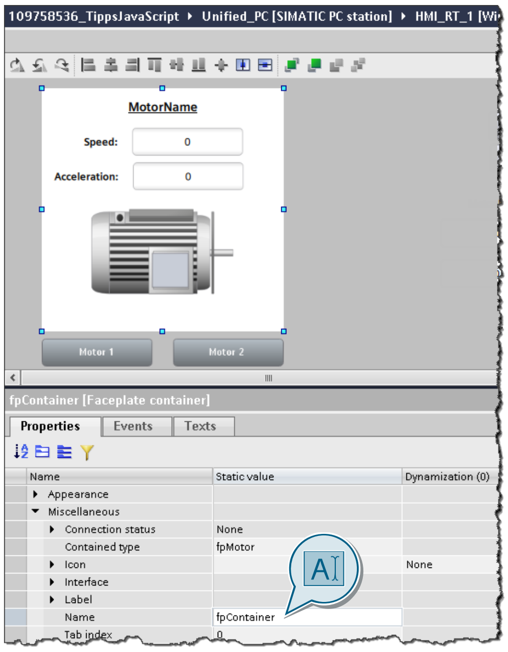
Speed: 0

Acceleration: 0

Configuration

1. Place the faceplate "fpMotor" in the screen by dragging and dropping.
2. Rename the automatically generated faceplate container to "fpContainer".
3. Insert two buttons labeled "Motor 1" and "Motor 2".

Figure 3-16



109758536_TippsJavaScript ▸ Unified_PC [SIMATIC PC station] ▸ HMI_RT_1 [Wi

MotorName

Speed: 0

Acceleration: 0

Motor 1 | Motor 2

fpContainer [Faceplate container]

Properties | Events | Texts

Name	Static value	Dynamization (0)
▸ Appearance		
▾ Miscellaneous		
▸ Connection status	None	
Contained type	fpMotor	
▸ Icon		None
▸ Interface		
▸ Label		
Name	fpContainer	
Tab index	0	

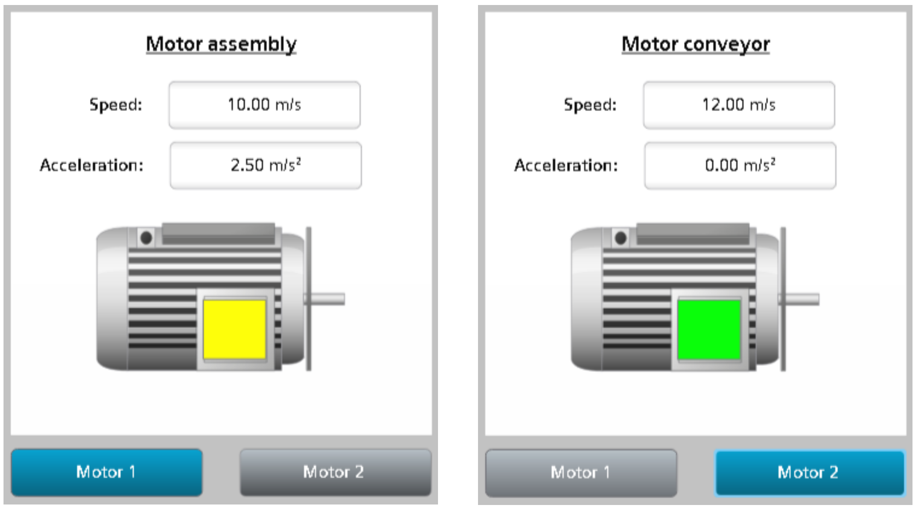
4. Add the following script to the "Click left mouse button" event on the button for Motor 1.

```
Screen.Items("fpContainer").Properties.MotorName.Tag = "Motor1.Name";
Screen.Items("fpContainer").Properties.Speed.Tag = "Motor1.Speed";
Screen.Items("fpContainer").Properties.Acceleration.Tag = "Motor1.Acc";
Screen.Items("fpContainer").Properties.Indicator = 0xff00ff08;
```

5. Repeat step 4 on the button for Motor 2, replacing "Motor1" with "Motor2".

Result in the runtime

Figure 3-17



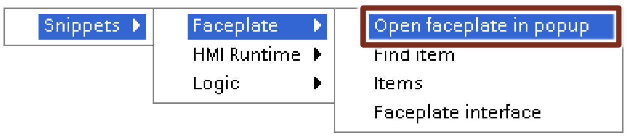
3.6.3 Open a faceplate from a faceplate

In WinCC Unified you also have the ability to open a second faceplate from an already opened faceplate.

Code snippet

Within a faceplate, you also have the option of opening a second faceplate by using the snippet "Open faceplate in popup".

Figure 3-18



```
let po = Faceplate.OpenFaceplateInPopup("Faceplate type_1", "title",
true, false);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

Note If you launch or open a second faceplate (e.g. "Faceplate_2") from the original faceplate (e.g. "Faceplate_1"), the interconnected interface tags will be automatically transferred to ("inherited by") the faceplate being opened (e.g. "Faceplate_2"). No additional configuration steps are necessary for this.

3.6.4 Close a faceplate

One way to close a faceplate is the "Faceplate.Close()" method.

To do this, create a button within the corresponding faceplate.

Add an event to the button (e.g. "Click left mouse button"), convert the event to a script and insert the following code:

```
Faceplate.Close();
```

The method automatically distinguishes between two cases:

- If the faceplate is a (static) faceplate container in a screen, then the faceplate will be changed to hidden.
- If the faceplate is a pop-up, it will be closed.

Note

The Faceplate.Close() method only works within a faceplate. External access is therefore not possible. For this reason, it must be pre-configured in the faceplate editor.

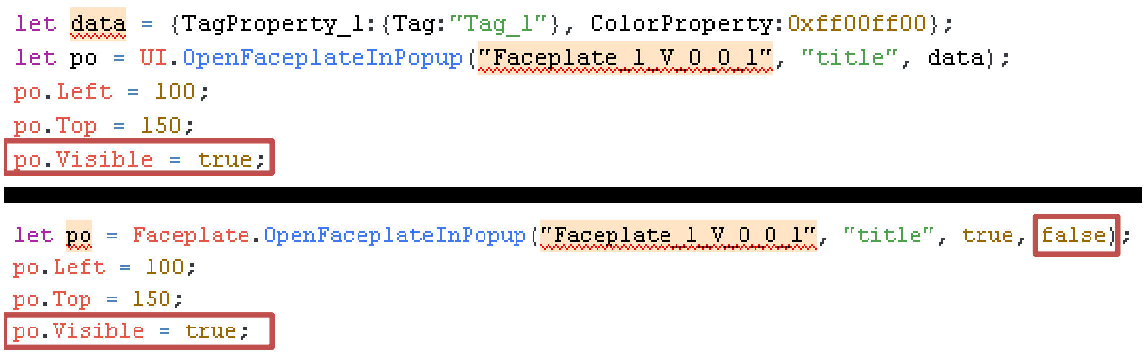
3.6.5 The Invisible parameter

The parameter "Invisible" is at the end of the "OpenFaceplateInPopup" method; it is optional. The parameter is present both in "UI.OpenFaceplateInPopup(…)" as well as in "Faceplate.OpenFaceplateInPopup". The default value of the parameter when inserting the snippet is "false" (invisible = false -> pop-up is visible). The pop-up opens hidden if you change the value to "true".

Note

When you insert the snippet "OpenFaceplateInPopup" the call "po.Visible = true;" is added at the same time.
This directly contradicts the Invisible parameter (Visible <> Invisible) and always takes precedence. Regardless of the value you choose for the Invisible parameter, the state specified in "po.Visible" is always applied.
If you wish to avoid this and use only the Invisible parameter, then you must comment out or delete the line with the corresponding call.

Figure 3-19



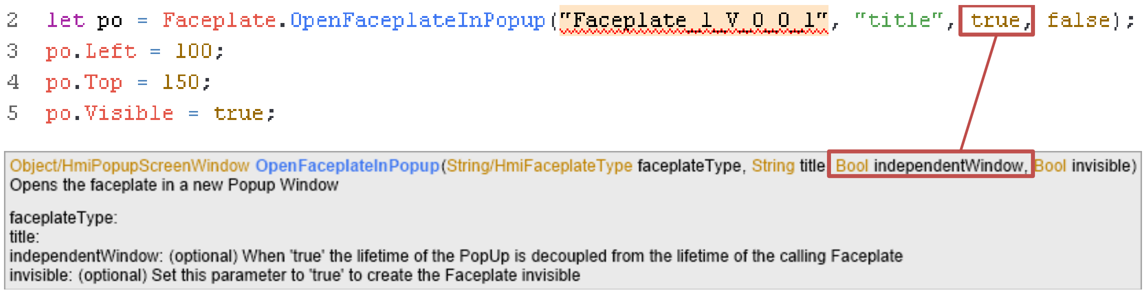
```
let data = {TagProperty_1:{Tag:"Tag_1"}, ColorProperty:0xff00ff00};
let po = UI.OpenFaceplateInPopup("Faceplate_1_V_0_0_1", "title", data);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

```
let po = Faceplate.OpenFaceplateInPopup("Faceplate_1_V_0_0_1", "title", true, false);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

3.6.6 The "IndependentWindow" parameter

The "OpenFaceplateInPopup" method contains the Boolean parameter "IndependentWindow".

Figure 3-20



```
let po = Faceplate.OpenFaceplateInPopup("Faceplate_1_V_0_0_1", "title", true, false);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

```
Object/HmiPopupScreenWindow OpenFaceplateInPopup(String/HmiFaceplateType faceplateType, String title, Bool independentWindow, Bool invisible)
Opens the faceplate in a new Popup Window

faceplateType:
title:
independentWindow: (optional) When 'true' the lifetime of the PopUp is decoupled from the lifetime of the calling Faceplate
invisible: (optional) Set this parameter to 'true' to create the Faceplate invisible
```

Independent of other conditions, this parameter lets you influence the position and "lifespan" of a pop-up faceplate.

In this chapter you will find a list of the possible configurations and their results.

Note

The parameter "IndependentWindow" is only available when you call the "OpenFaceplateInPopup" method in another faceplate (Faceplate.OpenFaceplateInPopup(…)). If using it in a screen (UI.OpenFaceplateInPopup(...)), the method structure is different and does not contain this parameter.

Dependency on the originating faceplate

- IndependentWindow = true
  - The pop-up faceplate is not closed when you close the dependent faceplate.
- IndependentWindow = false
  - When you close the originating faceplate, the pop-up faceplate closes with it.

Behavior on screen changes

- IndependentWindow = true
  - The pop-up faceplate remains open (foregrounded) during a screen change.
- IndependentWindow = false
  - The pop-up faceplate is closed (terminated) during a screen change.

Coordinate origins

The coordinate origin (0/0) is the position from which the values for "top" and "left" are measured (in pixels).

Depending on the configuration, the coordinate origin is located in the upper left corner

- of the main screen
- of the screen window
- of the faceplate
  (if this is a pop-up faceplate, the origin refers to the faceplate's screen contents - any header here is not included).

3.6.7 Parent function

A so-called "parent" describes a screen object that is directly superordinate to the object being referenced, such as

- the screen in which a screen object is located.
- the screen window in which a screen is located.
- the faceplate in which a screen object is located.
- the screen in which a faceplate is located or launched from.
- the faceplate from which a pop-up faceplate is launched

where the object named first is always the parent object.

Access to a parent can be configured with scripting; it facilitates reading of data and values from a parent object. Depending on the object, these could be, for example, the size, position, color, name or other such data. Information read in this manner can then be reused in the lower-level screen object (the one making the request).

Since configuration steps with "parents" differs from case to case and since there are various ways of implementing this, we will not go into greater detail at this time.

More detailed information on this topic can be found in the TIA Portal Help under the keyword "Parent".

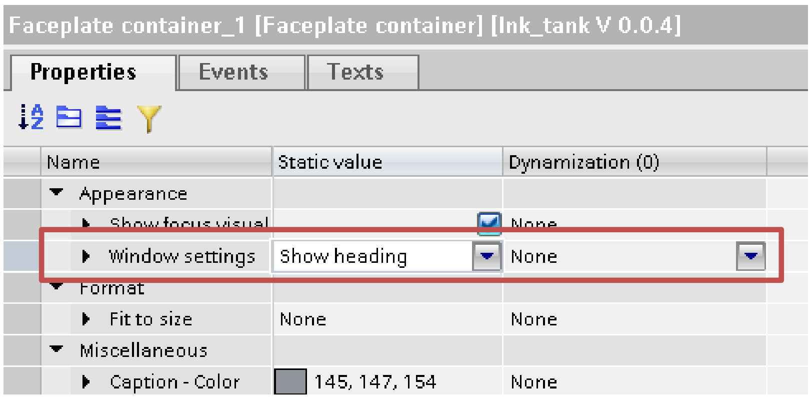
3.6.8 Header

General remarks

When configuring a faceplate, you can define whether the faceplate should appear with a so-called "header" (Faceplate container > Properties > Appearance > Show heading).

The header can be the title of the faceplate, for example, and contain a "Close" button. The appearance and height of the header change depending on the style you are using. The width always matches that of the faceplate container (the faceplate instance).

Figure 3-21


Faceplate container_1 [Faceplate container] [Ink_tank V 0.0.4]

Name	Static value	Dynamization (0)
Appearance		
Show focus visual	✓	None
Window settings	Show heading	None
Format		
Fit to size	None	None
Miscellaneous		
Caption - Color	▨ 145, 147, 154	None

Appearance behavior when using a header

When you activate the header in a faceplate instance, this will cause new behaviors when you change the appearance with the "Fit to size" property:

- Fit to size > None

  Hiding the header causes the original size of the faceplate container to be preserved. Showing the headers shrinks the area for displaying screen contents. This also hides the scrollbars.

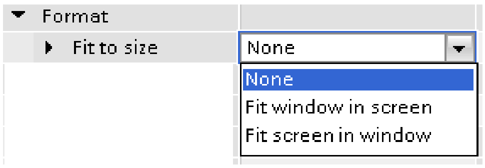
- Fit to size > Fit screen in window

  The screen content of the faceplate is scaled down at the same ratio (height and width) such that the header and full screen content is displayed - while the size of the faceplate container remains the same. A side effect of this is a larger "empty screen" in the faceplate instance.

- Fit to size > Fit window in screen

  The faceplate container will be automatically enlarged so that the header and the full screen content are displayed in their original sizes.

Figure 3-22


Format
Fit to size: None

Effect of the header on the "IndependentWindow" parameter

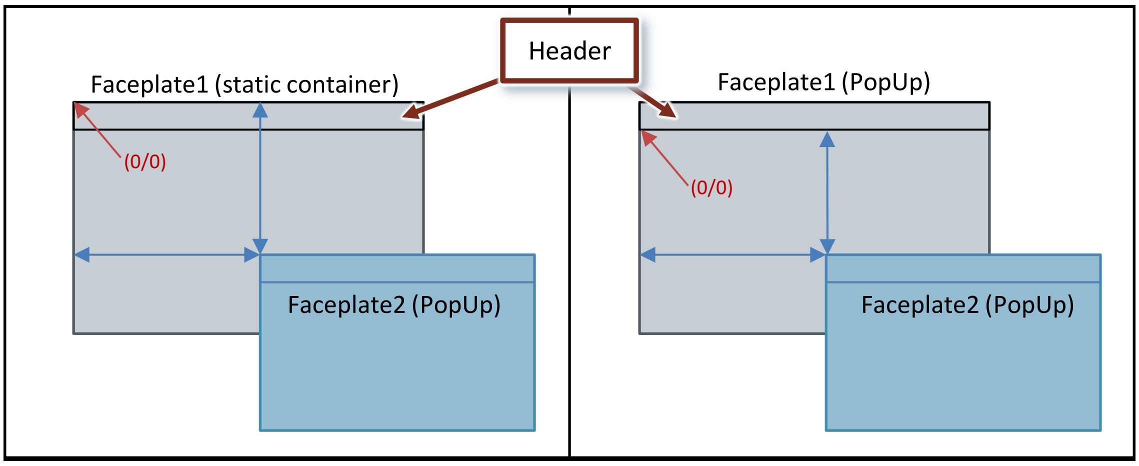
Using a header affects the behavior of the "IndependentWindow" parameter.

If you launch a pop-up faceplate from another faceplate and you use the "IndependentWindow" parameter, then the coordinate origin of the pop-up will change.

If you are using a header in a faceplate which

- is located in a (static) faceplate container within a screen, then when the parent pop-up faceplate is launched, the upper left corner of the faceplate container will be selected as the coordinate origin (the possible existence of a header is ignored).
- was opened as a pop-up faceplate (from another screen), then when the parent pop-up faceplate is launched, the upper left corner of the screen content will be selected as the coordinate origin. This is implemented in this way because a header can have a variable height (as a function of the chosen style). In this case, you may need to calculate in the specific height of the header to obtain the correct values for position and size.

Figure 3-23



3.7 Examples with the "IndependentWindow" parameter

Note In the following examples we assume that Faceplate1 and Faceplate2 are configured as pop-up faceplates, where Faceplate1 is launched from a screen and Faceplate2 is launched from Faceplate1. In addition, both faceplates have a header.

Table 3-1

No.	Case	Brief description	Chapter
1.	Faceplate2 needs to display details about a machine. The information must remain visible when the operator carries out actions on the rest of the visualization (for example, screen changes). The position of the faceplate is defined in absolute terms. The faceplate must be explicitly closed.	Position of Faceplate2 is absolute in the (main) screen. Faceplate2 remains open during a screen change or when Faceplate1 is closed.	3.7.1.1- Absolute and persistent (screen, large)
2.	Faceplate2 needs to display details about a motor. In this case, it needs to open next to the view of the associated machine (Faceplate1). Faceplate2 is positioned relative to Faceplate1. It should close automatically when Faceplate1 is closed or during a screen change. Suitable for large screens / monitors.	Position of Faceplate2 relative to Faceplate1. Faceplate2 is closed upon screen change or when Faceplate1 is closed.	3.7.1.2 - Relative and temporary (screen, large)
3.	Faceplate2 needs to display details about a motor. In this case, it needs to open next to the view of the associated machine (Faceplate1). Faceplate2 is positioned relative to Faceplate1. The information must remain visible when the operator carries out actions on the rest of the visualization (for example, screen changes). The position of Faceplate2 is defined by the parent function (relative to Faceplate1). In this way, for example, the position can be controlled by a script and modified automatically. This is especially useful in small screens / Panels. The faceplate must be explicitly closed.	Position of Faceplate2 is determined via parent object. Faceplate2 remains open during a screen change or when Faceplate1 is closed.	3.7.1.3 - Relative and persistent (screen, small)
4.	Like #2, but the position of Faceplate2 is defined by the parent function (relative to Faceplate1). In this way, for example, the position can be controlled by a script and modified automatically. This is especially useful in small screens / Panels.	Position of Faceplate2 is determined via parent object. Faceplate2 is closed upon screen change or when Faceplate1 is closed.	3.7.1.4 - Relative and temporary (screen, small)
5.	Like #1, but Faceplate1 is launched in a screen within a screen window. The position of both faceplates is defined with an absolute reference to the main screen, regardless of the screen window.	The position of both faceplates is independent of the screen window. Faceplate2 remains open during a screen change or when Faceplate1 is closed.	3.7.1.1- Absolute and persistent (main screen)
6.	Like #2, but Faceplate1 is launched in a screen within a screen window. The position of Faceplate1 is defined absolutely to the main screen.	The position of Faceplate2 is relative to Faceplate1. Faceplate1 is independent of screen window. Faceplate2 is closed upon screen change or when Faceplate1 is closed.	3.7.2.2 - Relative and temporary (main screen)
7.	Like #1, but the position of Faceplate1 is defined via its parent object (screen in screen window). The position of Faceplate2 is defined absolutely to the main screen.	The position of Faceplate2 is independent of the screen window. Position of Faceplate1 is relative to the screen window. Faceplate2 remains open during a screen change or when Faceplate1 is closed.	3.7.2.3- Absolute and persistent (screen window)

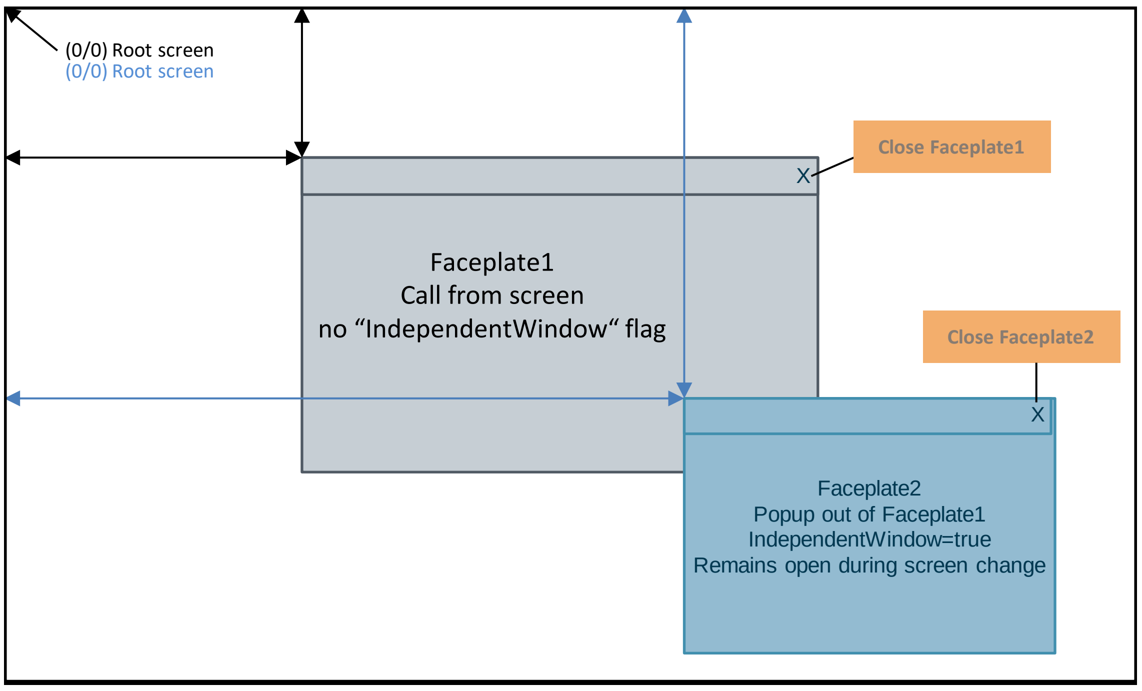
No.	Case	Brief description	Chapter
8.	Like #2, but the position of Faceplate1 is defined via its parent object (screen in screen window).	Position of Faceplate2 is relative to Faceplate1, position of Faceplate1 is relative to screen window, Faceplate2 is closed upon screen change or when Faceplate1 is closed.	3.7.2.4 - Relative and temporary (screen window)

3.7.1 Launching in (main) screen

3.7.1.1 Absolute and persistent (screen, large)

- The Faceplate1 making the call is launched in a screen.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "true".
- The coordinate origin (0/0) of both faceplates is the same as that of the (main) screen.
- Faceplate1 and Faceplate2 can be closed independently of one another.
- Faceplate2 remains open (in the foreground) during a screen change.

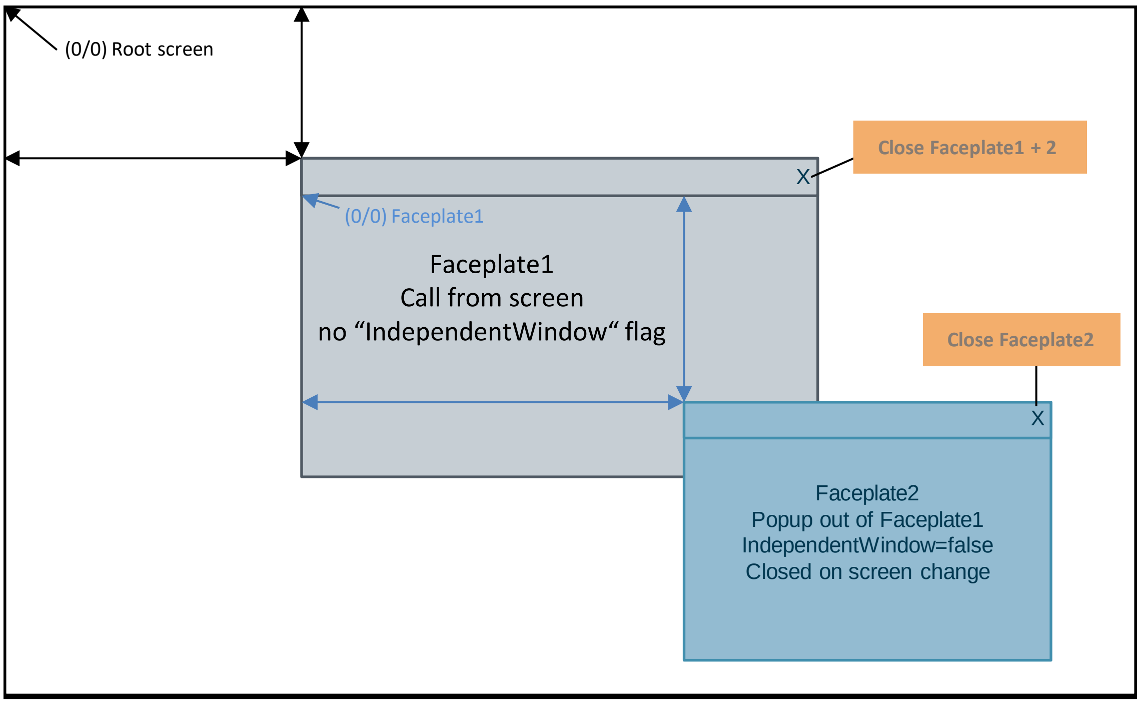
Figure 3-24



3.7.1.2 Relative and temporary (screen, large)

- The Faceplate1 making the call is launched in a screen.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "false".
- The coordinate origin (0/0) of Faceplate1 matches that of the (main) screen.
- The coordinate origin (0/0) of Faceplate2 matches that of the screen content of Faceplate1.
- Closing Faceplate1 also closes Faceplate2.
- Faceplate2 is closed upon a screen change.

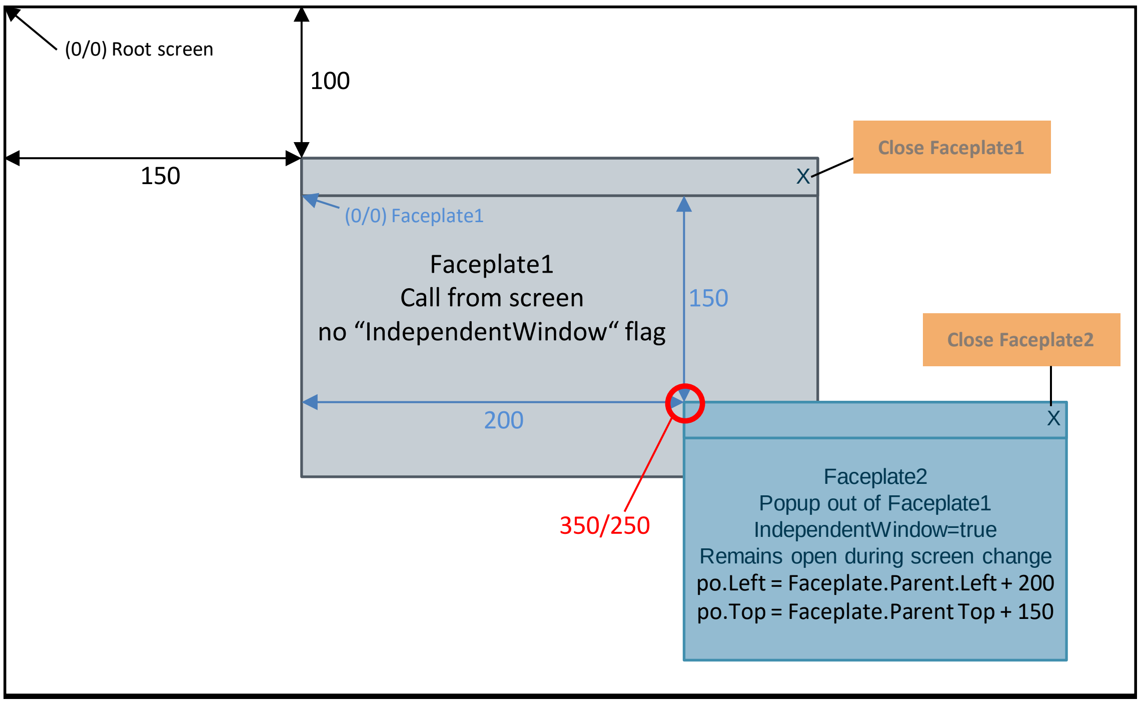
Figure 3-25



3.7.1.3 Relative and persistent (screen, small)

- The Faceplate1 making the call has no "IndependentWindow" parameter.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "true".
- The coordinate origin (0/0) of Faceplate1 matches that of the (main) screen.
- The coordinate origin (0/0) of Faceplate2 matches that of the screen content of Faceplate1.
- Faceplate2 accesses its parent object (Faceplate1).
  - ⇨ The position of Faceplate1 is ascertained and its X and Y (upper left) coordinates are added to those of Faceplate2. The position of Faceplate2 is thus defined as a function of the position of Faceplate1.
- Faceplate1 and Faceplate2 can be closed independently of one another.
- Faceplate2 remains open (in the foreground) during a screen change.

Figure 3-26

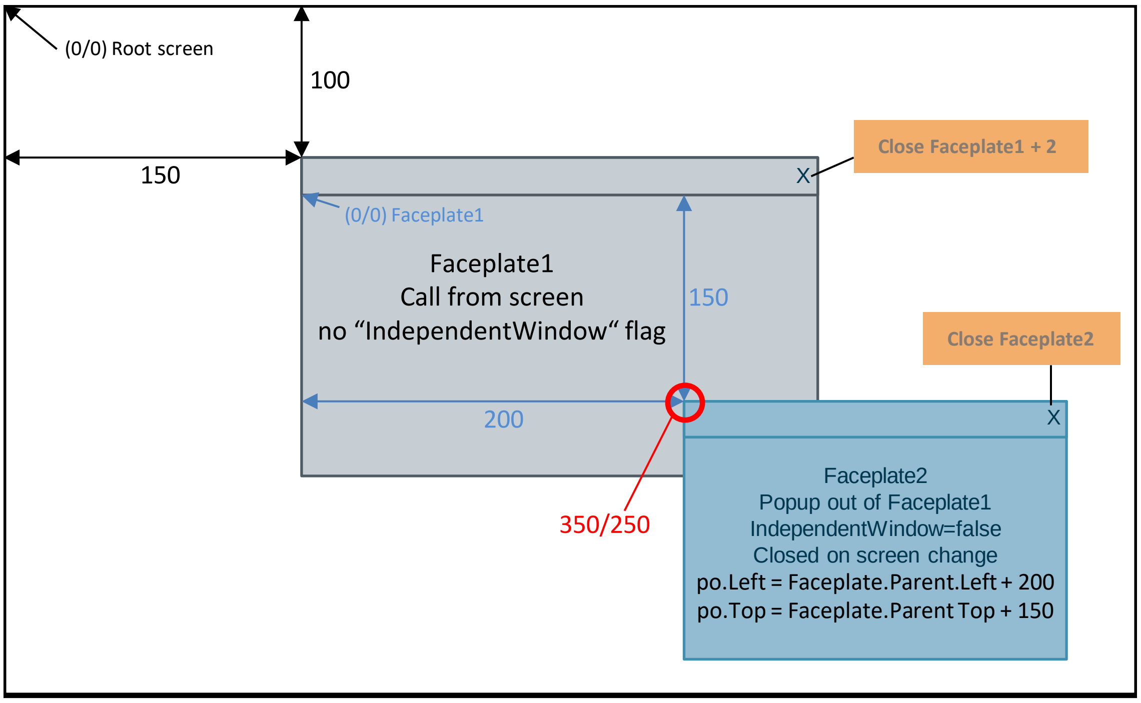


3.7.1.4 Relative and temporary (screen, small)

- The Faceplate1 making the call is launched in a screen.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "false".
- The coordinate origin (0/0) of Faceplate1 matches that of the (main) screen.
- The coordinate origin (0/0) of Faceplate2 matches that of the screen content of Faceplate1.
- Faceplate2 accesses its parent object (Faceplate1).
  ⇨ The position of Faceplate2 is defined relative to the position of Faceplate1.
- Closing Faceplate1 also closes Faceplate2.
- Faceplate2 is closed upon a screen change (results from "IndependentWindow=false", independent of parent relation).

Note Behavior only applies as of TIA Portal V17 Update 4

Figure 3-27

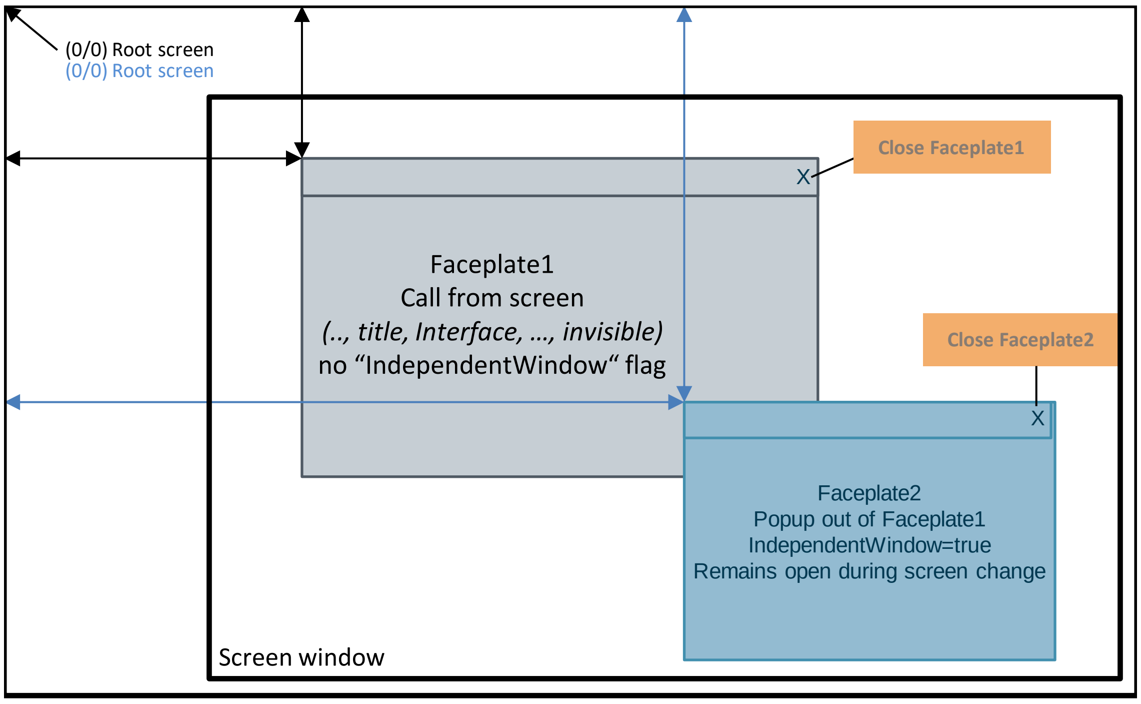


3.7.2 Launch in screen window

3.7.2.1 Absolute and persistent (main screen)

- The Faceplate1 making the call is launched in a screen within a screen window.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "true".
- The coordinate origin (0/0) of both faceplates is the same as that of the (main) screen.
- Faceplate1 and Faceplate2 can be closed independently of one another.
- Faceplate2 remains open (in the foreground) during a screen change.

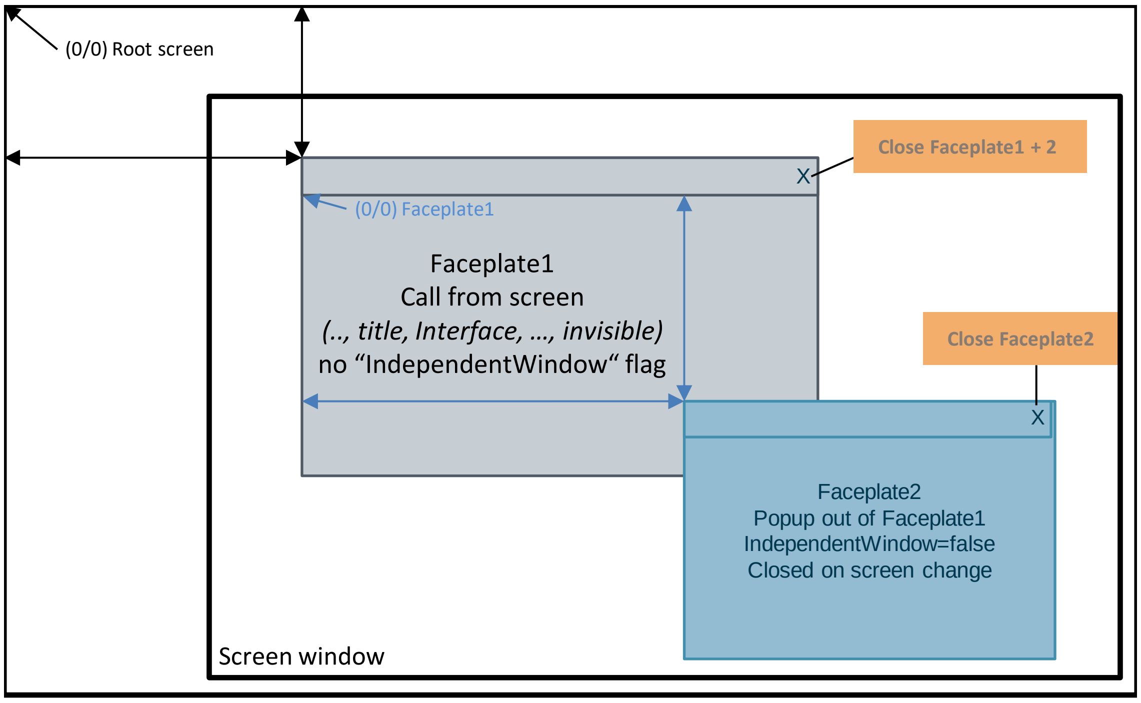
Figure 3-28



3.7.2.2 Relative and temporary (main screen)

- The Faceplate1 making the call is launched in a screen within a screen window.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "false".
- The coordinate origin (0/0) of Faceplate1 matches that of the (main) screen.
- The coordinate origin (0/0) of Faceplate2 matches that of the screen content of Faceplate1.
- Closing Faceplate1 also closes Faceplate2.
- Faceplate2 is closed upon a screen change.

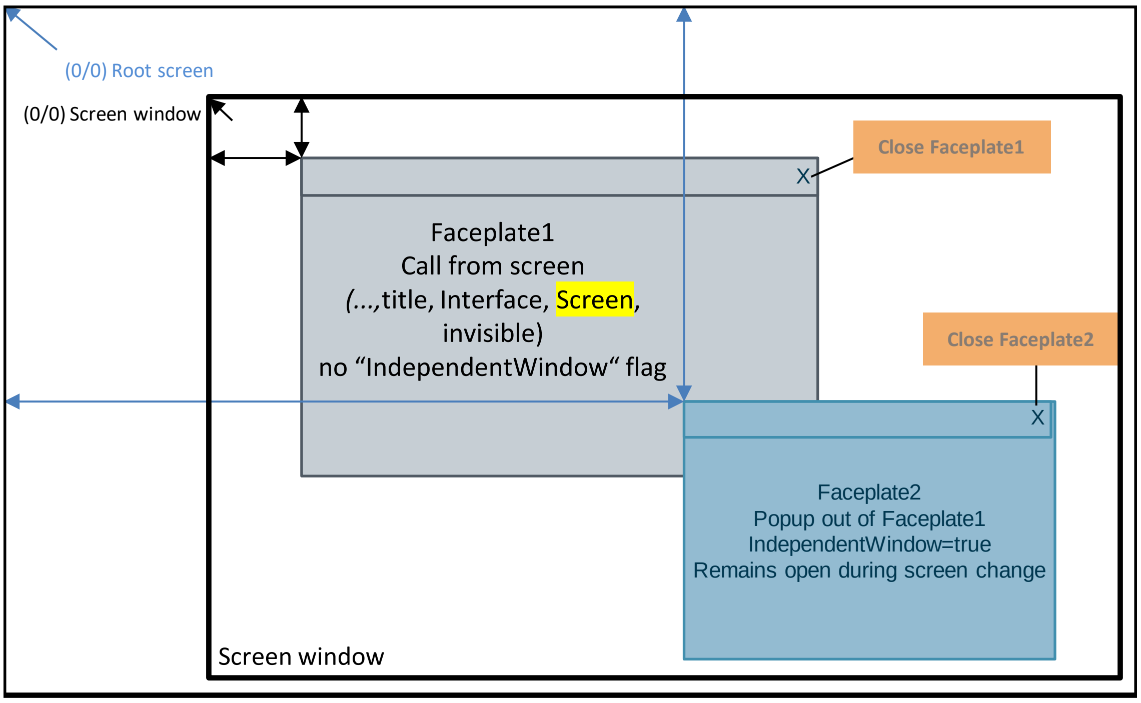
Figure 3-29



3.7.2.3 Absolute and persistent (screen window)

- The Faceplate1 making the call is launched in a screen within a screen window.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- Faceplate1 accesses its parent object (screen window).
  - ⇨ The coordinate origin (0/0) of Faceplate1 matches that of the screen window.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "true".
- The coordinate origin (0/0) of Faceplate2 matches that of the (main) screen.
- Faceplate1 and Faceplate2 can be closed independently of one another.
- Faceplate2 remains open (in the foreground) during a screen change.

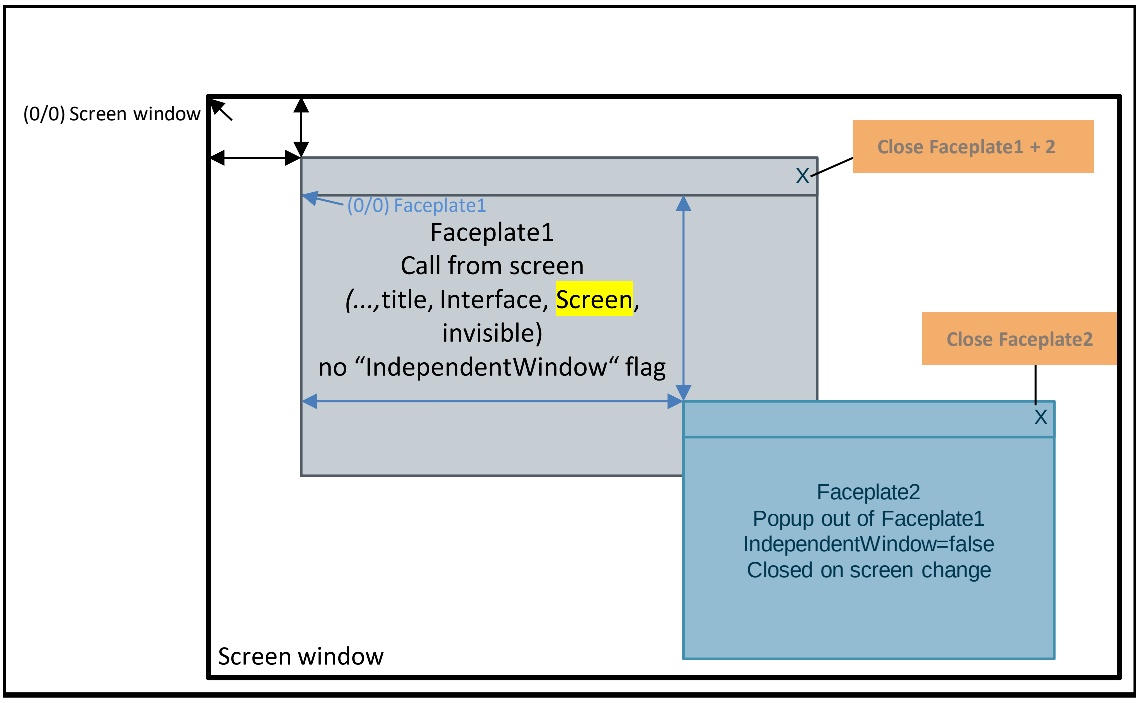
Figure 3-30



3.7.2.4 Relative and temporary (screen window)

- The Faceplate1 making the call is launched in a screen within a screen window.
- The Faceplate1 making the call has no "IndependentWindow" parameter.
- Faceplate1 accesses its parent object (screen window).
  - ⇨ The coordinate origin (0/0) of Faceplate1 matches that of the screen window.
- The pop-up Faceplate2 is launched from Faceplate1.
- The "IndependentWindow" parameter of Faceplate2 is "false".
- The coordinate origin (0/0) of Faceplate2 matches that of the screen content of Faceplate1.
- Closing Faceplate1 also closes Faceplate2.
- Faceplate2 is closed upon a screen change.

Figure 3-31



4 Appendix

4.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:

support.industry.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts.

Please send queries to Technical Support via Web form:

support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:

siemens.com/sitrain

Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:

support.industry.siemens.com/cs/sc

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for iOS and Android:

support.industry.siemens.com/cs/ww/en/sc/2067

4.2 Industry Mall

The Siemens Industry Mall is the platform on which the entire siemens Industry product portfolio is accessible. From the selection of products to the order and the delivery tracking, the Industry Mall enables the complete purchasing processing – directly and independently of time and location:
mall.industry.siemens.com

4.3 Links and literature

Table 4-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the article page of the application example https://support.industry.siemens.com/cs/ww/en/view/109812366
\3\	SIMATIC HMI WinCC Unified Getting Started https://support.industry.siemens.com/cs/ww/en/view/109801175
\4\	Manual: SIMATIC HMI WinCC Unified Engineering V18 https://support.industry.siemens.com/cs/ww/en/view/109813308
\5\	SIMATIC WinCC Unified - Tips and Tricks for Scripting (JavaScript) https://support.industry.siemens.com/cs/ww/en/view/109758536
\6\	SITRAIN system course: WinCC Unified & Unified Comfort Panels https://support.industry.siemens.com/cs/ww/en/view/109773211
\7\	SITRAIN advanced course: SIMATIC WinCC Unified for PC systems https://support.industry.siemens.com/cs/ww/en/view/109781323
\8\	Multiuser Engineering with TIA Portal Project Server https://support.industry.siemens.com/cs/ww/en/view/109740141
\9\	The TIA Portal Tutorial Center (videos) https://support.industry.siemens.com/cs/ww/en/view/106656707

4.4 Change documentation

Table 4-2

Version	Date	Change
V1.0	01/2023	First version