



Siemens
Industry
Online
Support

APPLICATION EXAMPLE

Engineering guideline for WinCC Unified

SIMATIC WinCC Unified V18 / V19

SIEMENS

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.

Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.

By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.

The Siemens terms of use (<https://support.industry.siemens.com>) shall also apply.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial security measures that may be implemented, please visit

<https://www.siemens.com/industrialsecurity>.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under

<https://www.siemens.com/cert>.

Table of contents

1.	Introduction	6
1.1.	Overview.....	6
1.2.	Components used	7
1.3.	Explanation of the Symbols used	7
2.	Preliminary Information	8
2.1.	Reasons for using the Engineering Guideline.....	8
2.2.	Workflow for new Configurations	9
2.2.1.	Use Case: State dependent scripting: Show / Hide screen content	12
2.3.	Relevant sections for Screen change	14
3.	Best Practices of Screen Engineering	19
3.1.	Use of Screen Objects	19
3.1.1.	Screen Object Selection	21
3.1.1.1.	Use Case: Colored Square Box	22
3.1.1.2.	Use Case: Button without Implementation of the Press and Release Event.....	23
3.1.1.3.	Use Case: Simple Text Label V19	25
3.1.1.4.	Use Case: Colored Screen Background.....	26
3.1.1.5.	Use Case: Custom Styles V19	27
3.1.2.	Tidy up the Screen / Observe System Limits	30
3.1.3.	Usage of Screen Windows.....	31
3.1.4.	Screen Object Visibility	32
3.1.4.1.	Use Case: Visibility Dynamization of multiple Screen Objects by the same Condition	32
3.1.5.	Unified Controls	35
3.1.5.1.	Use Case: Different Settings in Runtime for Unified Controls	35
3.1.5.2.	Use Case: Alarm Control	36
3.1.5.3.	Use Case: Alarm Line Custom Solution	38
3.1.6.	Graphics and SVGs.....	41
3.1.6.1.	Use Case: Visualize Patterns and composed Objects.....	41
3.1.6.2.	Use Case: Composed Objects with Dynamization → dynamic SVG.....	42
3.2.	Use of Dynamizations	44
3.2.1.	Simple Tag Dynamization	44
3.2.2.	Script Dynamization	44
3.2.3.	Expressions	45

3.2.3.1.	Use Case : Dynamize a Property depending on Single Bits ^{V19 Upd2}	46
3.2.4.	Further Options	47
3.3.	Use of Faceplates	48
3.3.1.	Essential Insights into Faceplate Usage	48
3.3.1.1.	Use Case: Faceplate that is not always Visible in Runtime ^{V19}	48
3.3.1.2.	Use Case: Strings in the Property Interface	49
3.3.1.3.	Use Case : Dynamic Data Connection for hierarchical Faceplates	50
3.3.2.	Text Lists in Faceplates	51
3.3.2.1.	Use Case: Transmitting a Subset of the Text List	51
3.3.2.2.	Use Case: Static use of Text Lists ^{V19}	54
3.3.2.3.	Use Case: Dynamic use of Text Lists	55
3.4.	Use of scripts	59
3.4.1.	Script Triggers	59
3.4.1.1.	Use Case: Scripts using the same Trigger Tag	60
3.4.1.2.	Use Case: Trigger Scripts unrelated to Screen Object Properties	62
3.4.2.	Efficient Code	64
3.4.2.1.	Use Case: Write and Read multiple Tags	66
3.5.	Others.....	69
3.5.1.	Acquisition Cycle	69
3.5.1.1	Use Case: Writing a PLC Tag in Screen Loaded Event	70
3.5.2.	Use Case: PLC UDT Arrays with Multiplexing	73
4.	Analysis of an existing Project.....	75
4.1.	ShowScripts Add-In	78
4.1.1.	Download and Installation	78
4.1.2.	Usage of the ShowScripts Add-In	79
4.1.3.	Preparation of the Screen Overview File	81
4.1.4.	Analysis of the Screen Overview File	83
4.1.4.1.	Analysis of Screen Context.....	83
4.1.4.2.	Usage of Controls	85
4.1.4.3.	Usage of Dynamizations	86
4.1.5.	Analysis of the exported Scripts	88
4.2.	WinCC Unified JS Connector ^{V19}	91
4.3.	Runtime Analysis	93
4.3.1.	RTIL Trace Viewer	93
4.3.1.1.	Additional Flags.....	95
4.3.2.	Script Debugger	96
5.	Appendix.....	101

5.1.	Service and support	101
5.2.	Links and literature	102
5.3.	Change documentation	102

1. Introduction

1.1. Overview

SIMATIC WinCC Unified is the completely newly developed visualization system from Siemens in the automation environment. The SIMATIC WinCC Unified system consists of the SIMATIC WinCC Unified visualization software and the new SIMATIC HMI Unified Comfort Panels as well as the Unified Basic Panels.

The Unified Comfort Panels extend the product range of the SIMATIC Advanced Panel-based HMIs and are the successor devices of the SIMATIC HMI Comfort Panels. In addition to the new hardware, there are numerous innovative new features that have a lot of impact in the way you use things. To get a better overview of the steps that help you get the best out of the new features and behaviors in Unified Comfort panels and PC Runtime, this document will address several topics. We will discuss the advantages and guide you through the steps of the changed process.

The content of this document refers to the entire Unified HMI portfolio, unless there are further references to device-specific information.



1.2. Components used

This application example has been created with the following hard- and software components:

Component	Number	Article Number	Note
WinCC Unified Engineering V18	1	6AV215-...01-8A.5	Engineering in TIA Portal
WinCC Unified Engineering V19	1	6AV215-...02-3.A5	Engineering in TIA Portal
SIMATIC HMI Unified PC Runtime V18	1	6AV2154-..B01-8.A0	Runtime System
SIMATIC HMI Unified PC Runtime V19	1	6AV2154-..B02-3.A0	Runtime System
SIMATIC HMI Unified Comfort Panel MTP1200	1	6AV2128-3MB06-0AX1	Alternatively, any other SIMATIC HMI Unified Comfort Panel can be used.

You can purchase these components from the [Siemens Industry Mall](#).

NOTE




Please note that all functions shown or explained in the document only apply to WinCC Unified V18 and V19 including updates.

This application example consists of the following components:

Component	File name	Note
Manual	109827603_WinCC_Unified_engineering_guideline_DOC_V2_en.pdf	This document

1.3. Explanation of the Symbols used

The following table shows the used symbols that indicate for which version the described recommendation can be applied. If no symbol is used, the content has no restrictions on the version.

Symbol	Unified Version
	Only related to V18
	Only related to V19
	Only related to V19 Upd2

2. Preliminary Information

2.1. Reasons for using the Engineering Guideline

Unified Panel and PC-RT Devices offer the latest technologies, such as HTML5, JavaScript and scalable vector graphics (SVGs). This makes them more powerful and able to offer a wider range of functions and capabilities than the old Comfort Panels and WinCC RT Advanced systems, allowing you to get more out of your system. In order to use them efficient this document provides some recommendations for the engineering. The presented implementations can be helpful if you start a new project but also if you have an existing project already there.

When creating a new project, it is recommended to first work through the engineering guideline. It will provide the knowledge to use the resources of the devices the most efficient way.

But also, if there is already an existing Unified HMI project, this guideline contributes to further improve the implementation. There is also an explicit section on how to analyze a project to find the areas that can be improved easily.

If optimizations regarding Java Script code in general, the style guide for scripting and the HMI layout can be found in additional documentation: [StyleChecker](#), [HMI Template Suite](#), [Tips and Tricks for Scripting](#)

All tools can also be found in section [5.2. Links and literature](#).

2.2. Workflow for new Configurations

The moment you start creating completely new configurations or even just new screens is the perfect opportunity for you to utilize the new WinCC Unified function in the most efficient way. In this context the term view is used and stands for the entire display of an HMI and thus for a combination of several screens in a screen window arrangement. When creating new views, the first step is to collect all the content that should be displayed in advance. Depending on its reusability, you decide to categorize into individual content or content that will be used in multiple screens. The procedure of the most efficient handling is explained in the following scheme.

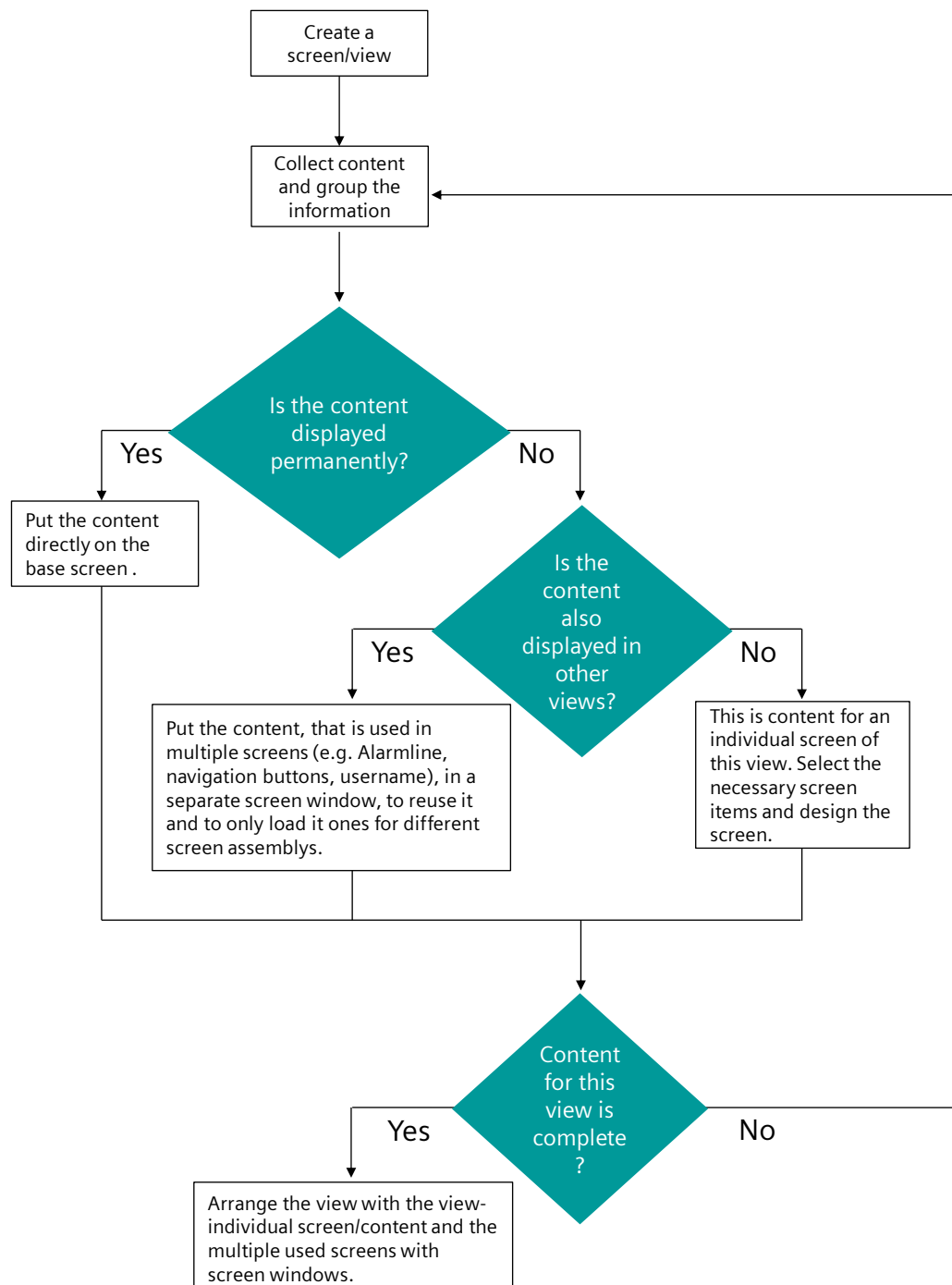


Figure 2-1 Flowchart: Creating a view with screens

The result of this configuration could be a screen layout like it is displayed in the following figure. All individual screen windows are highlighted with an orange frame.

For guidance on structuring your project, consider utilizing the HMI Template Suite. This template helps to create and organize the display area into distinct screen segments with versatile screen navigation (see [Figure 2-2](#)). It prioritizes performance and design aspects, ensuring an intuitive operational concept.

NOTE A detailed description about the setup and usage is available in the application example [HMI design with the HMI Template Suite](#)

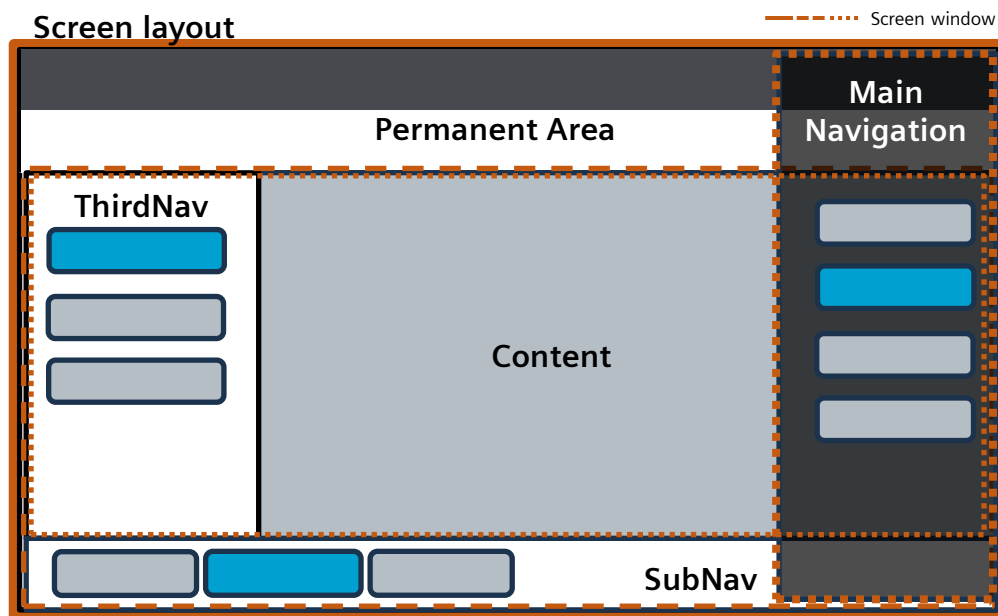


Figure 2-2 Example of a view using several screen windows

Once you have determined which elements are to be placed on which screen, you can continue working on the individual screens. Most of the screen items need dynamic properties or will use events to trigger system functions or scripts. To implement these dynamizations, the following scheme shows you which implementation might be best for your application. Depending on the desired functionality, select a suitable element from the toolbox and drag it onto the screen if it is not already there.

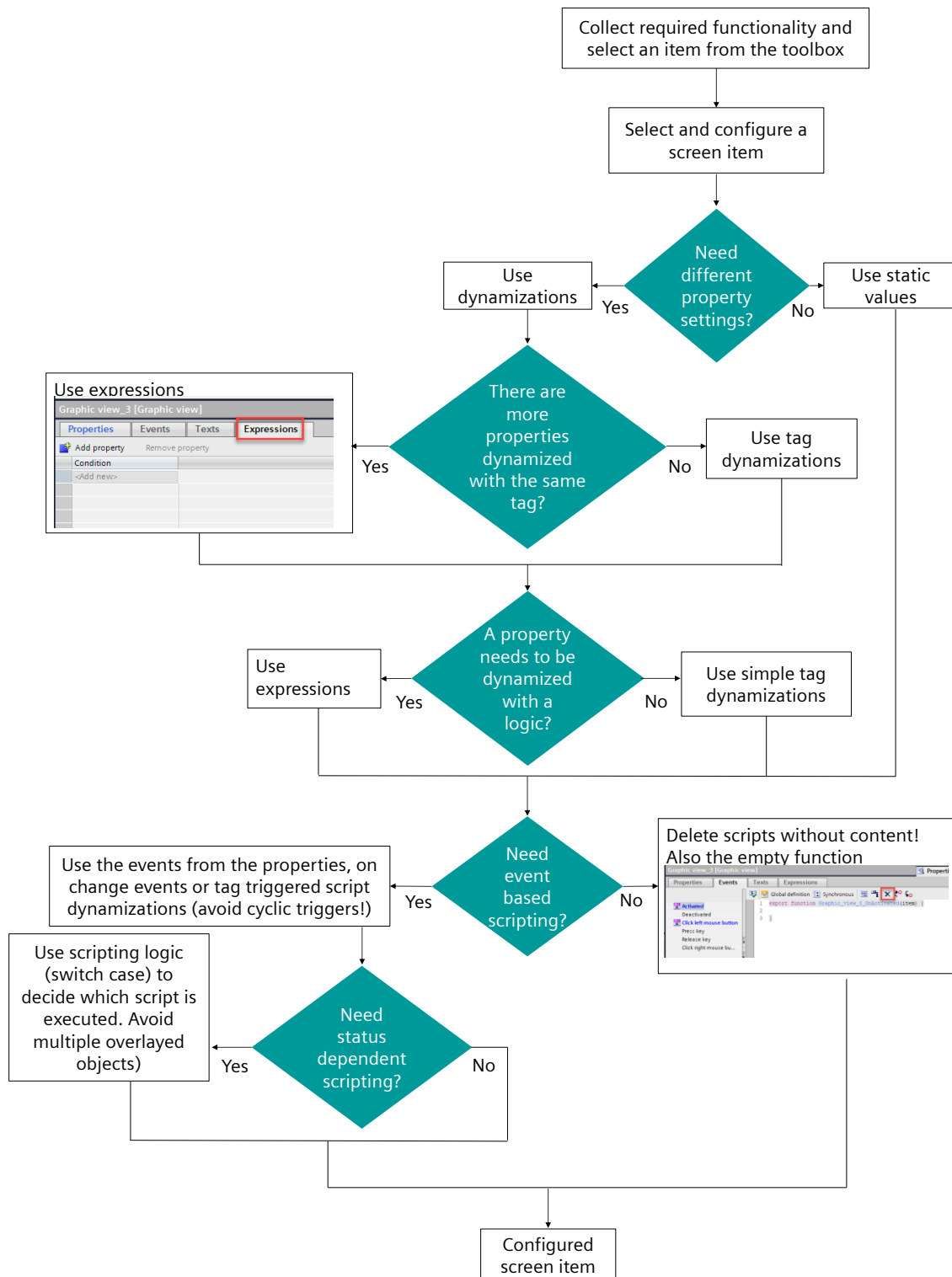


Figure 2-3 Configuring a new screen item flowchart

NOTE

In general, tag dynamization is the most recommended dynamization type. If a more complex logic is necessary, use the Expressions. Try to avoid script dynamizations. Nevertheless, if a scripting logic can replace multiple overlaid screen items, then use scripts. e.g., a button to show and/or hide other elements: Use one button and decide through scripting which method (show or hide) is executed (see chapter [2.2.1](#)).

The following section explains the practice of a scripting logic instead of multiple objects with a simple use case.

2.2.1. Use Case: State dependent scripting: Show / Hide screen content

Scripting in WinCC Unified now offers many more options for configuring screen elements and makes them much more flexible. As described in the following use case, a single button can trigger different actions depending on the current state.

Description

The concrete action or scripting behind an event can often depend on the current state. In this exemplary use case, there should be a button to show and hide a specific screen content (here: a rectangle), dependent on the current visibility of that screen content. One solution could be to use two different buttons, that implement once the show-action and once the hide-action and to change the visibility of these two buttons also with the current state. The second solution could be, having only one button, and decide by scripting (switch ... case), which action should be done.

Solution

Use a single button for this task and implement a scripting logic for the decision if the visibility of the content needs to be set to be visible or not. Directly use the same tag (here: "ContentVisible") for

- Indication of the current state
- the visibility dynamization of the content
- the button text dynamization

As this use case is very simple, the inverting of the state tag "ContentVisible" in the scripting logic could also be just done by a system function through the button click event. But to demonstrate how the implementation could look like for a more complex use case the screenshot shows the implementation with switch case.

To change the button text for the different states, it is dynamized with a text list and again the state tag.

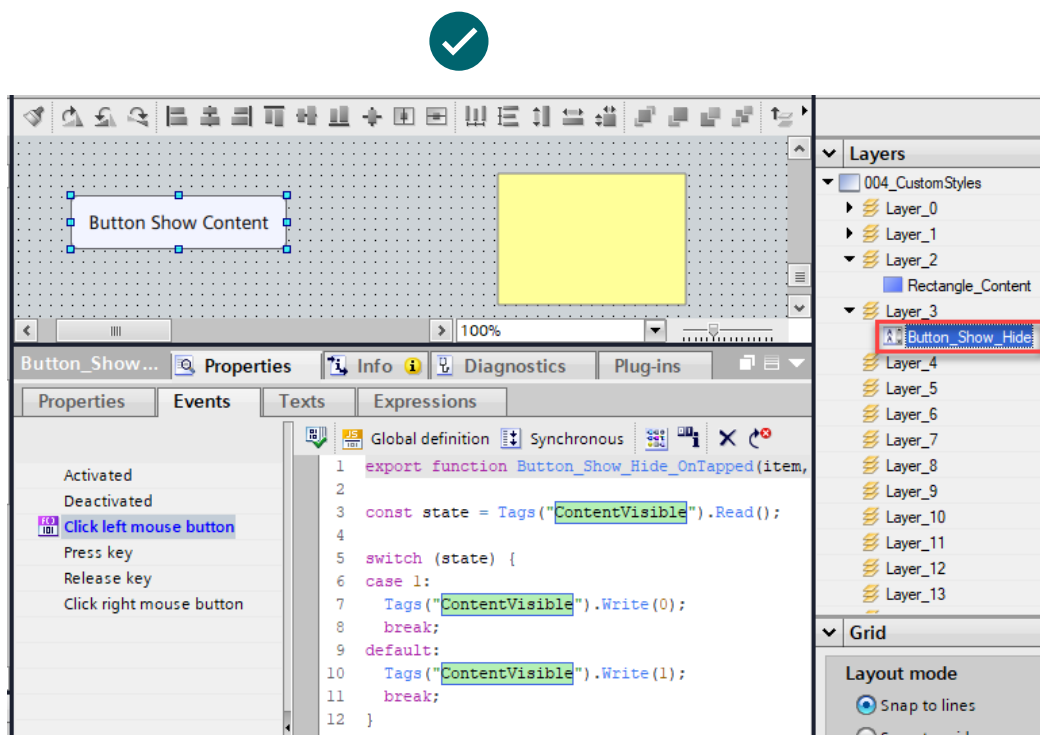


Figure 2-4 Show-Hide button use case scripting logic

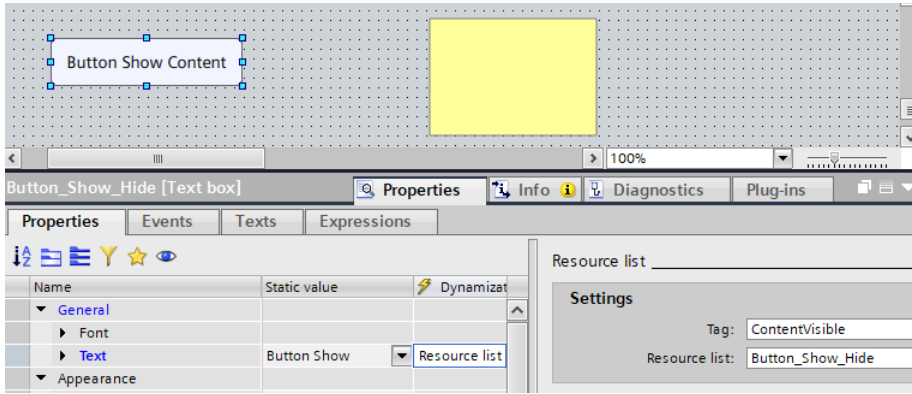


Figure 2-5 Button Show-Hide use case button text dynamization

In the following solution, that is NOT recommended, two overlaying buttons are used, one to set the visibility of the rectangle to true and a second one to set the visibility to false. To make the buttons accessible at the right time, their visibility is also dynamized with the visibility state of the rectangle. For such use cases with a state depended action, the previous shown scripting solution is better than having multiple overlaid screen item, in order to minimize the number of objects per screen.

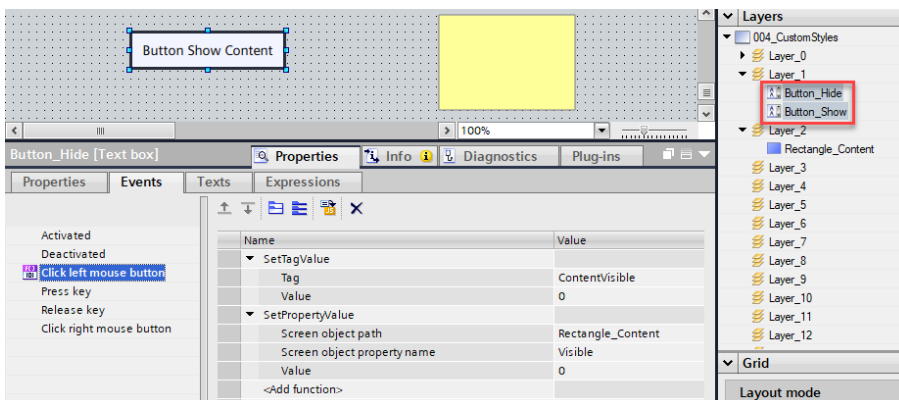


Figure 2-6 Two overlaying buttons for the show-hide-task

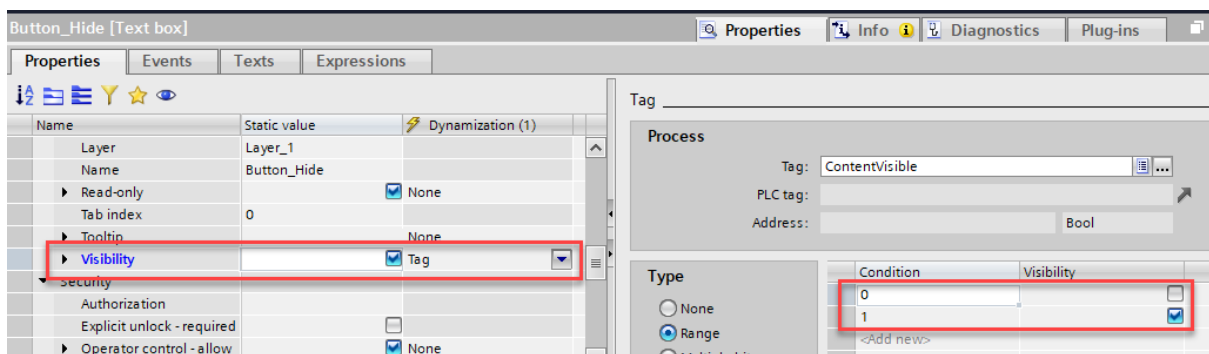


Figure 2-7 Visibility dynamization of the two buttons

The reason why the use case example applies text boxes as buttons, is explained in chapter [3.1.1.2. Use Case: Button without Implementation of the Press and Release Event.](#)

2.3. Relevant sections for Screen change

In Unified as well as in other visualization systems, screen changes play a central role in the representation of your various plant screens and in the user experience. A quick screen change gives the user a good feeling of efficiency in the projected elements, but a slow or long-lasting change makes them feel uncomfortable. With a few small tricks, you can prepare the screen changes even better.

To get the most out of your engineering steps, it is important that you understand how to realize the steps in the best way. Before going into too many details about the “right” implementation, it is also important to get familiar with the procedure of how a screen loads in runtime.

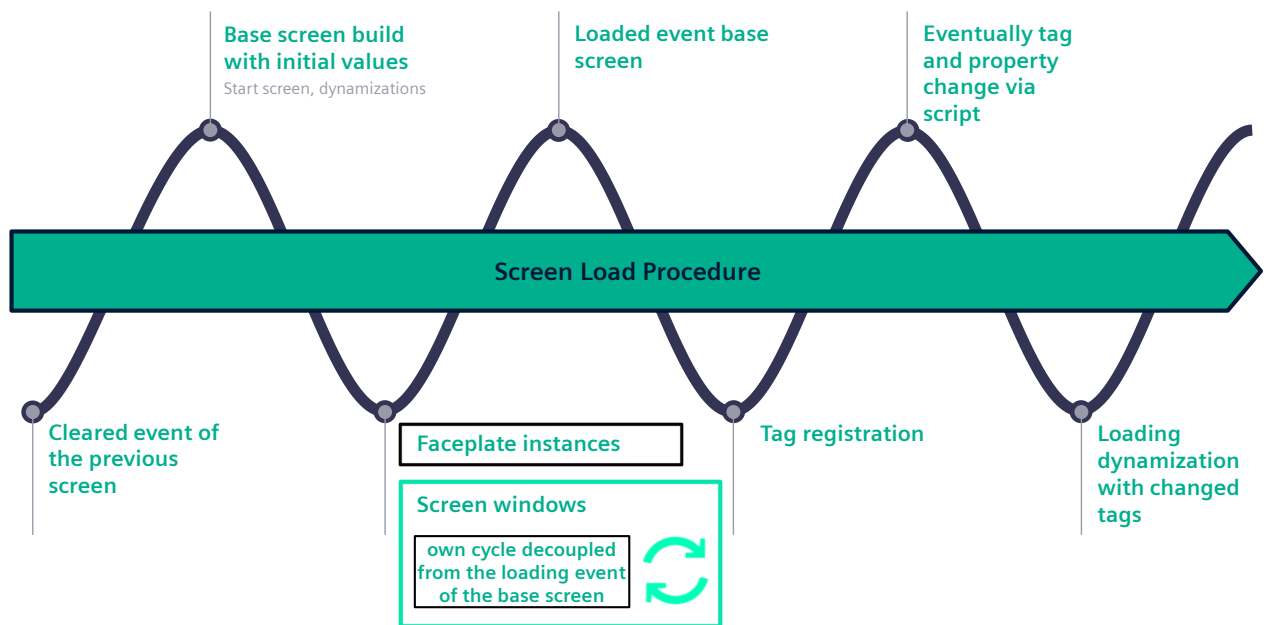


Figure 2-8 Screen load procedure

The screen load procedure of an individual screen is mostly separated in synchronous sections, meaning that one thing happens sequential after another. As the layout is often made by the composition of nested screens and Faceplates, it is important to know, that for each screen window and Faceplate instance there is an own decoupled cycle.

At the very beginning of a screen change, even before the new screen is loaded, the previous one gets cleared. After this step is finished, the base screen is built and all properties, even if they are dynamized, e.g., linked to tags, are loaded with the static value for an initial load of a screen. All nested Faceplates and screens in screen windows are loaded in an own decoupled cycle also first with static values.

Next the screen “loaded” event, that also can be customized in the engineering, is executed.

The tag registration happens after the initial load of the screen. When the tag registration is done, all dynamizations are considered and the screen item properties that have a linked tag in the dynamization area can change again from the static value to the value of the dynamized tag.

If tags, linked to any dynamization or any property are changed in this context again, also the screen item properties can change again. Through this order, the on-change event of a property can be executed twice during the screen load procedure. This second call of the change event is highlighted orange in the following figure ([Figure 2-9](#)).

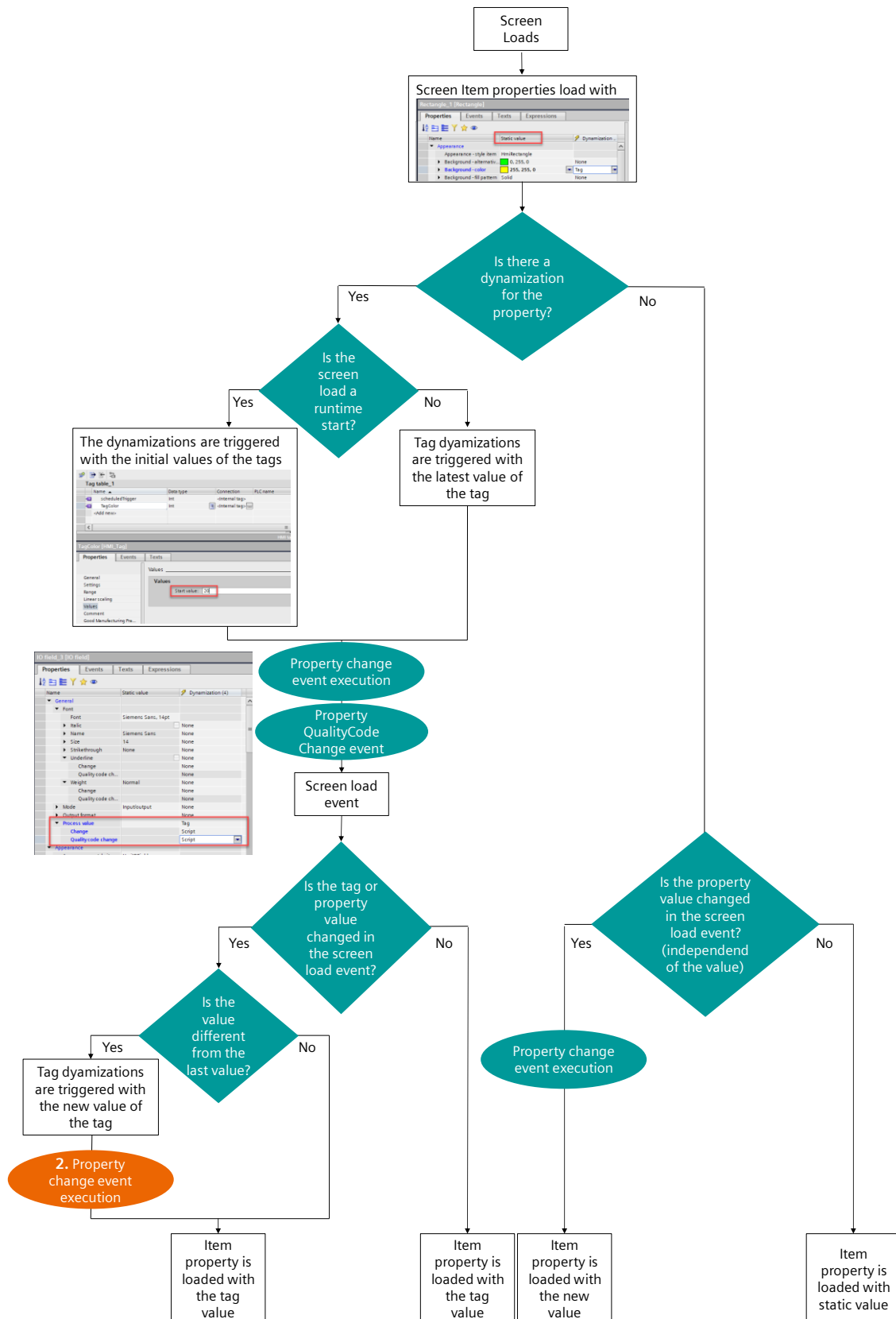


Figure 2-9 Call of the change event of screen item properties during screen load

The following chart shows the different behavior when a PLC Tag is used for dynamization. The important points are the PLC acquisition cycle time and the second execution of the quality code event. More information regarding acquisition cycle in general can be found in section: [Acquisition Cycle](#).

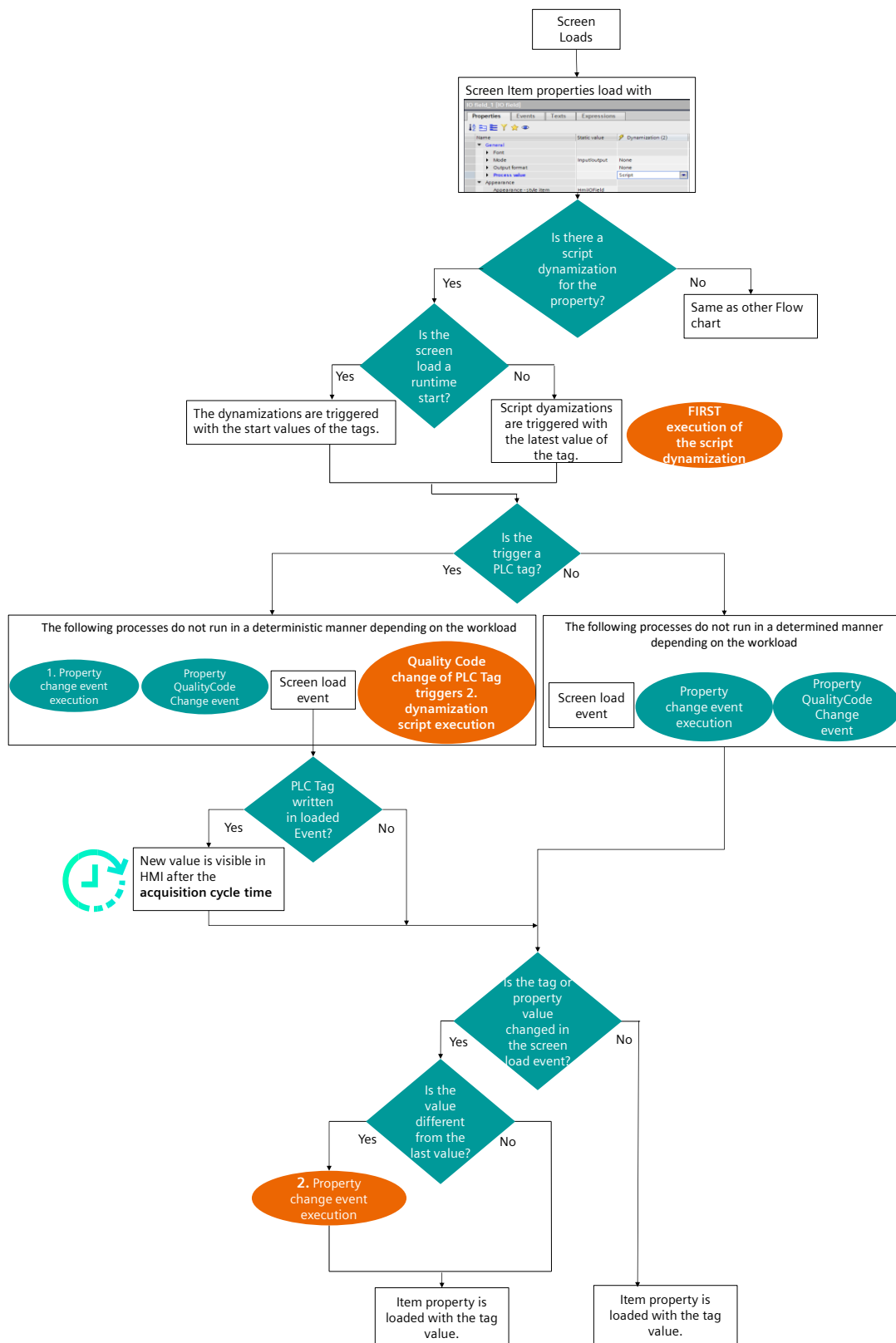


Figure 2-10 Dynamization with PLC Tag

Another important aspect are the execution contexts. A context is an independent Runtime environment where a specific script or task is performed. The different contexts work parallel in Runtime; therefore, you can handle several scripts and tasks at the same time.

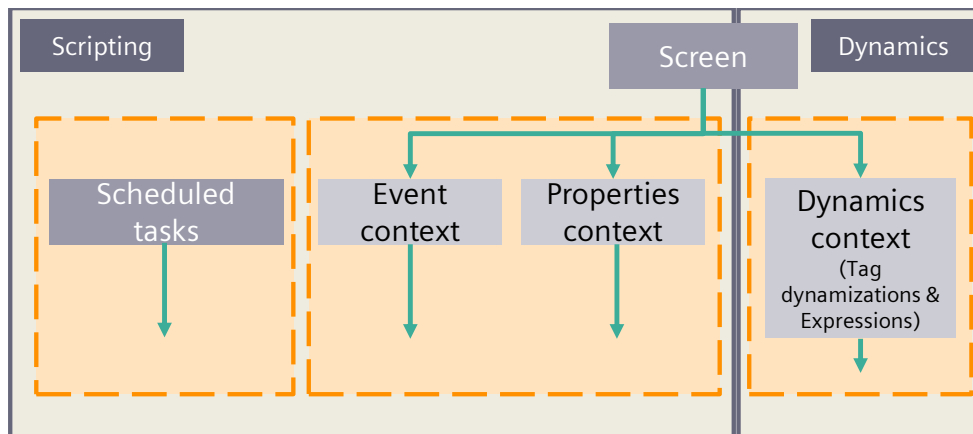


Figure 2-11 Script and dynamics execution contexts

As you can see in the figure above, the main separation is made into scripting and dynamics. This already indicates why the tag dynamization and Expressions are preferred to the script dynamization as they work in parallel to all processed scripts.

For the scripting the separation is between the scheduled tasks context and screen context. Each screen has its own scripting context for scripts and system functions. The namespace (global definition area) is unique for each scripting context. The scheduled tasks and also scripts in the screen context can be triggered cyclically, tag triggered, or alarm triggered. Both is important for the load procedure, the decision if the script is implemented in the scheduled task or screen context but also which trigger type is used (see chapter 3.4.1).

With this information it can already be defined what configurations are relevant for screen load.

Script	Is relevant	More information
Screen loaded event	✓	Figure 2-8 Screen load procedure
Script in the global definition area of screens	✓	
Scheduled tasks	-	Figure 2-11 Script and dynamics execution contexts
Script modules (referenced)	✓	
Scripts in "Click left mouse button" events	-	
Property on change event scripts	✓	Figure 2-9 Call of the change event of screen item properties during screen load
Dynamization scripts	✓	

Table 2-1 Overview of screen load relevant scripts

Object / Implementation	Is relevant	More information
Faceplate interface	✓	3.3 Use of Faceplates
Expressions	✓	
Dynamizations	✓	

Object / Implementation	Is relevant	More information
Number of used HMI tags	✓	See system limits in Unified manual 19
Number of screen elements (items, Faceplate-instances, screen windows,...)	✓	See system limits in Unified manual 19
Type of screen element	✓	3.1.1 Screen Object Selection

Table 2-2 Overview of screen load relevant objects

As you can see, there are a lot of things that affect the procedure of screen change at the runtime device. To give you all the detailed information on the single aspects, we're focusing even more on the following areas. You will also receive a description and best practice how to implement the features in the most efficient way. If a project needs to be analyzed there is a guideline provided in chapter [4](#). But first this document describes the best practices for screen engineering in more detail.



Figure 2-12 Best Practices Topics Overview

3. Best Practices of Screen Engineering

In this section lots of recommendations for the implementation of an HMI are described. Therefore, we will first show you on which aspects you can have an influence and which possibilities you have. Then we will give you illustrated details with the help of an exemplary use case. We split the sections in the use of screen objects, dynamizations, Faceplates, scripts and others.

NOTE

To support the comprehension of these best practices, the general screen load procedure is described in the chapter [2.3](#).

3.1. Use of Screen Objects

In the following figure, the available items of the Unified Toolbox for a PC-RT can be seen (The content for a Unified panel can differ). In general, there are five relevant sections that separate them: The Basic objects, Elements, Controls, My controls and the Dynamic widgets.

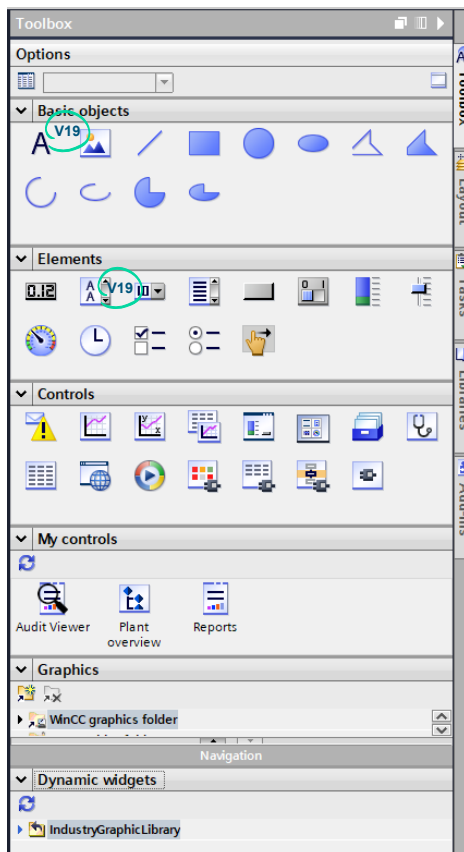


Figure 3-1 Unified Toolbox V19 for PC-RT

In general, all recommendations of this section about screen objects follow one rule.

**Screen object usage**

Minimize the number of objects on the screen and select the ones with the smallest necessary footprint



Before we go on, it is important to mention that each object has a footprint. With footprint the rendering effort is described. A small footprint is associated with a small number of properties and the low complexity of the screen item, e.g., a rectangle. Objects with a higher footprint often contain a lot of properties as the controls do have. Besides the fixed properties, also the customization through dynamizations can increase the rendering effort of a screen item and therefore influence the loading procedure of the whole screen.

3.1.1. Screen Object Selection

First, when placing a new object on the screen, select the one with the smallest footprint and just the necessary functionality. In general, you can say that the complexity raises from Basic objects to Controls, as the order in the toolbox. The following illustration shows all screen objects in the order of the required rendering effort. Please note that the rendering effort shown in this illustration has no time equivalent and only displays the relation. The actual rendering effort depends also on the configuration (dynamizations and connected data).

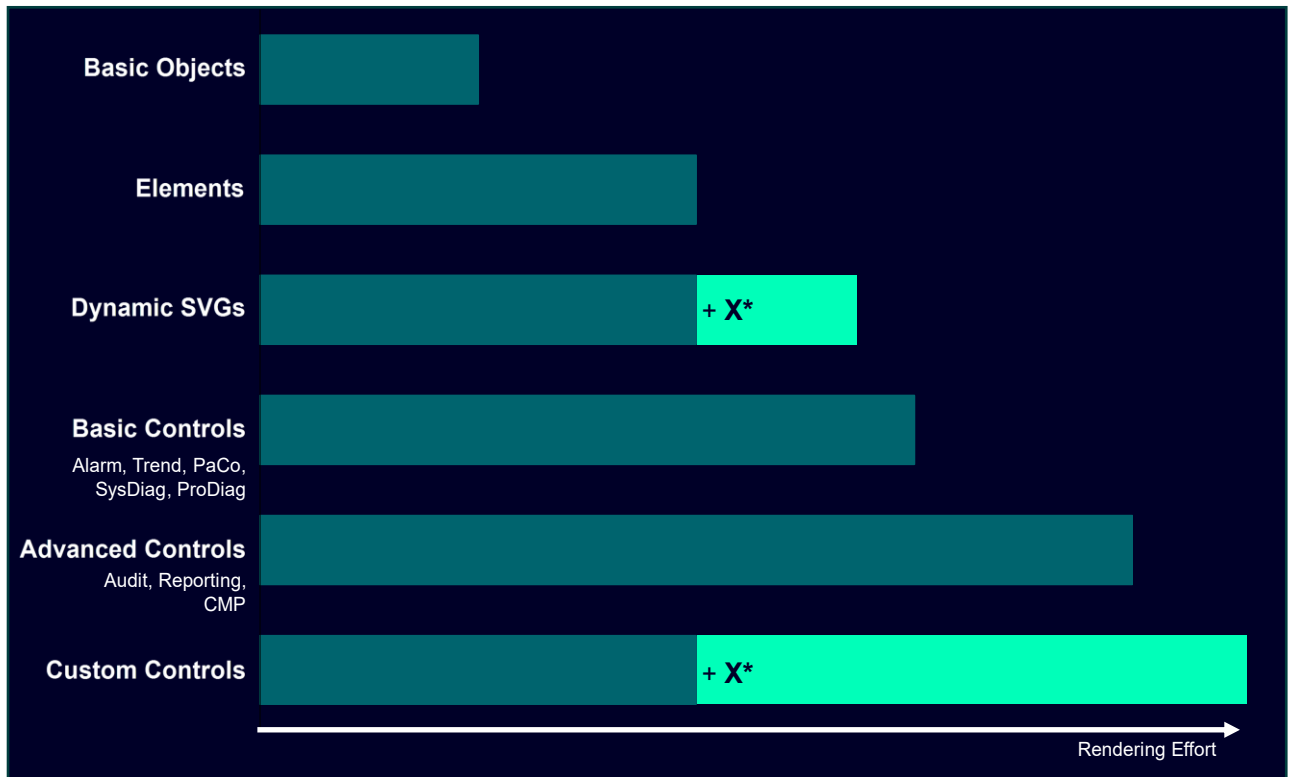


Figure 3-2 Relative comparison of the screen items' rendering effort

In the groups themselves there are differences as well and the most important thing to understand is that each screen object has an individual rendering effort (footprint). Before we go on, make sure that you know the toolbox elements and even the adaptations in V19. The text and text box adaption is shown in the figure below.

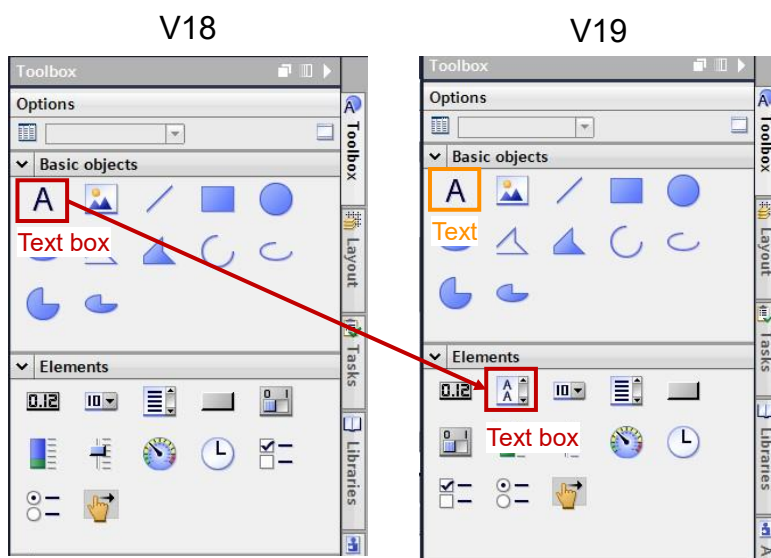



Figure 3-3 Toolbox V18 vs Toolbox V19

NOTE  The Text box from the basic objects has moved to the elements section for V19. The same looking item in the basic objects of the V19 toolbox is now “only” a text, comparable to a simple label. So, if the engineering system is a V18 version, the rendering effort of the Text box is also higher than for the basic objects and comparable with the items from the elements section

Sometimes, there is more than one possibility to display a functionality with different objects. Even if two objects look the same when configured on screen, they can have different influence on loading time of the whole screen. In the following some examples are shown.

Even if the following examples do not cover a complete real use case, keep these points in mind when configuring a new screen.

3.1.1.1. Use Case: Colored Square Box

Description

This use case is about the configuration of a colored square box as you can see in the figure below. Possible solutions can be a rectangle or a Text box without text assignments because they can look the same on a screen.

Solution

Instead of using a Text box, the rectangle element is the better choice due to its smaller rendering effort.

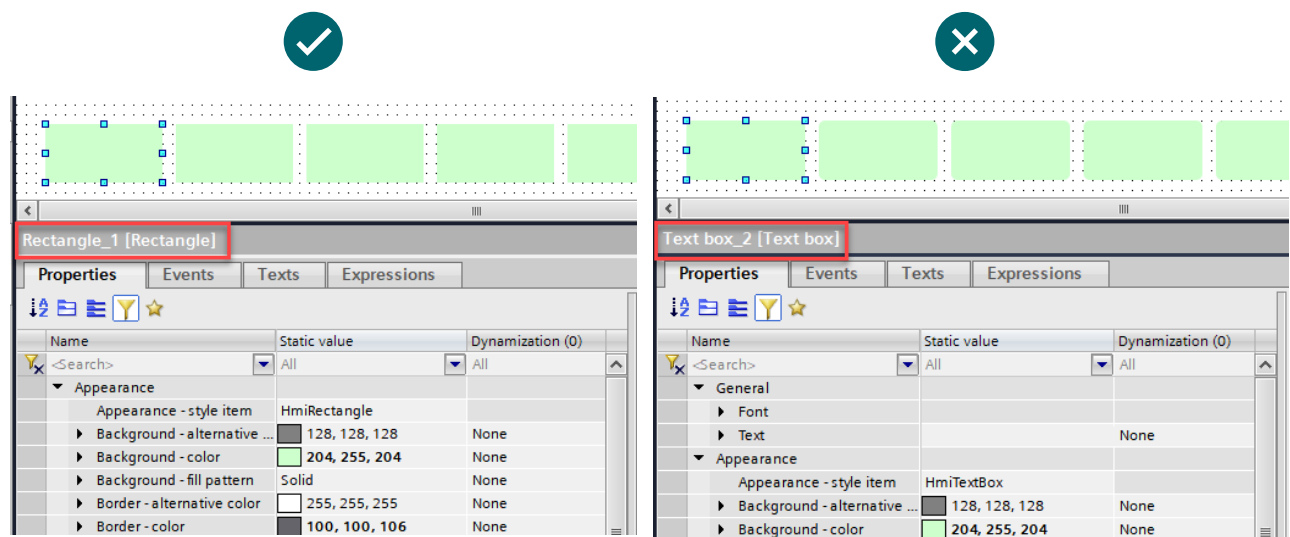


Figure 3-4 Rectangle vs. Text box



Screen object selection

Always use the screen object with the smallest footprint and the necessary functionality



3.1.1.2. Use Case: Button without Implementation of the Press and Release Event

Description

This use case is about the configuration of a Button that only uses the “Click left mouse button” event and without implementation of the press or release event. Possible solutions can be a Text box, a Text (V19) or a Button.

Solution

In terms of rendering effort, the text has the smallest footprint, followed by the text box and the button. Therefore, use a Text box if you need a background color, otherwise a Text if they fulfil your requirements. Also, these items have on mouse click events. Only use a button when the press and release events are necessary.

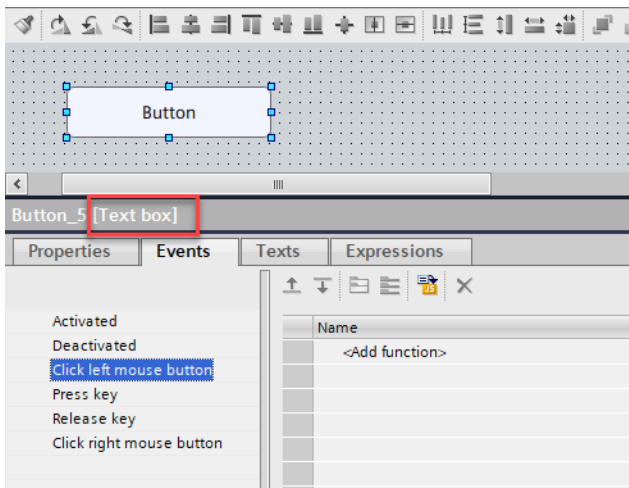


Figure 3-5 Text box as a button

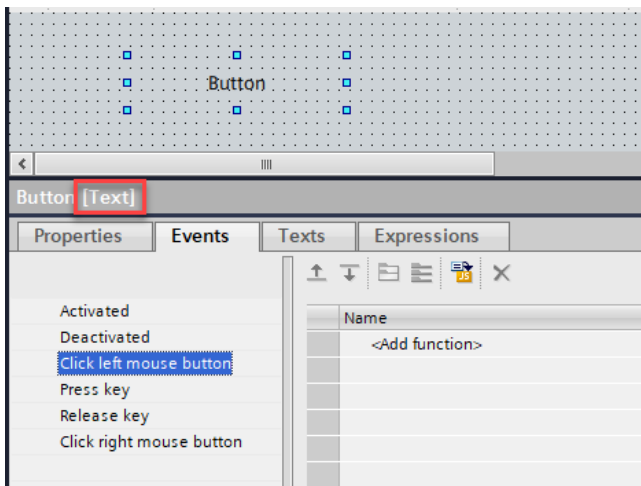


Figure 3-6 Text as a button

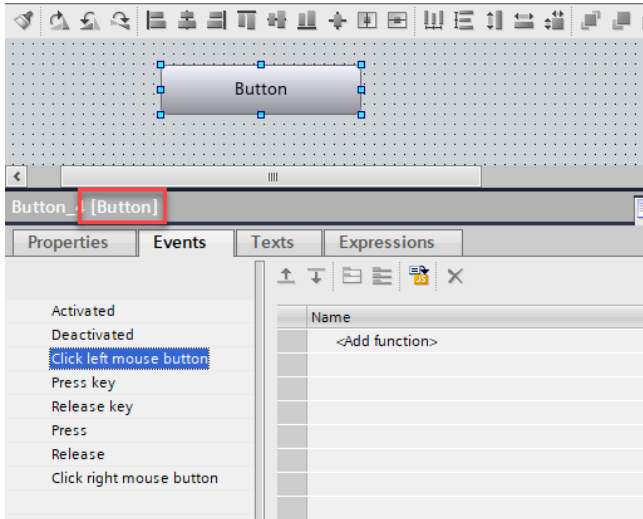


Figure 3-7 Button as a button

3.1.1.3. Use Case: Simple Text Label V19

Description

For text element of the screen a Text box or a Text can be used.

Solution

Whenever possible, especially for simple labels, use a Text instead of a Text box. The following table shows the different feature sets of a Text compared to a Text box. Compared to the text box, the reduced functional scope of the text causes the lower food print and rendering effort at the runtime device.

Property	Text	Text box
Colors	Foreground	Foreground, background, border, alternative colors
Border	X	Color, Width
Format – Spacing	X	✓
Format – Text trimming	X	✓
Format – Text break	X	✓
Miscellaneous – Connection quality	X	✓
Miscellaneous – Read only	X	✓
Parameter Field	✓	✓
Editable / Input Text (in RT)	X	✓
Copy and paste (in RT)	X	✓

Table 3-1 Comparison of a Text and Text box screen item

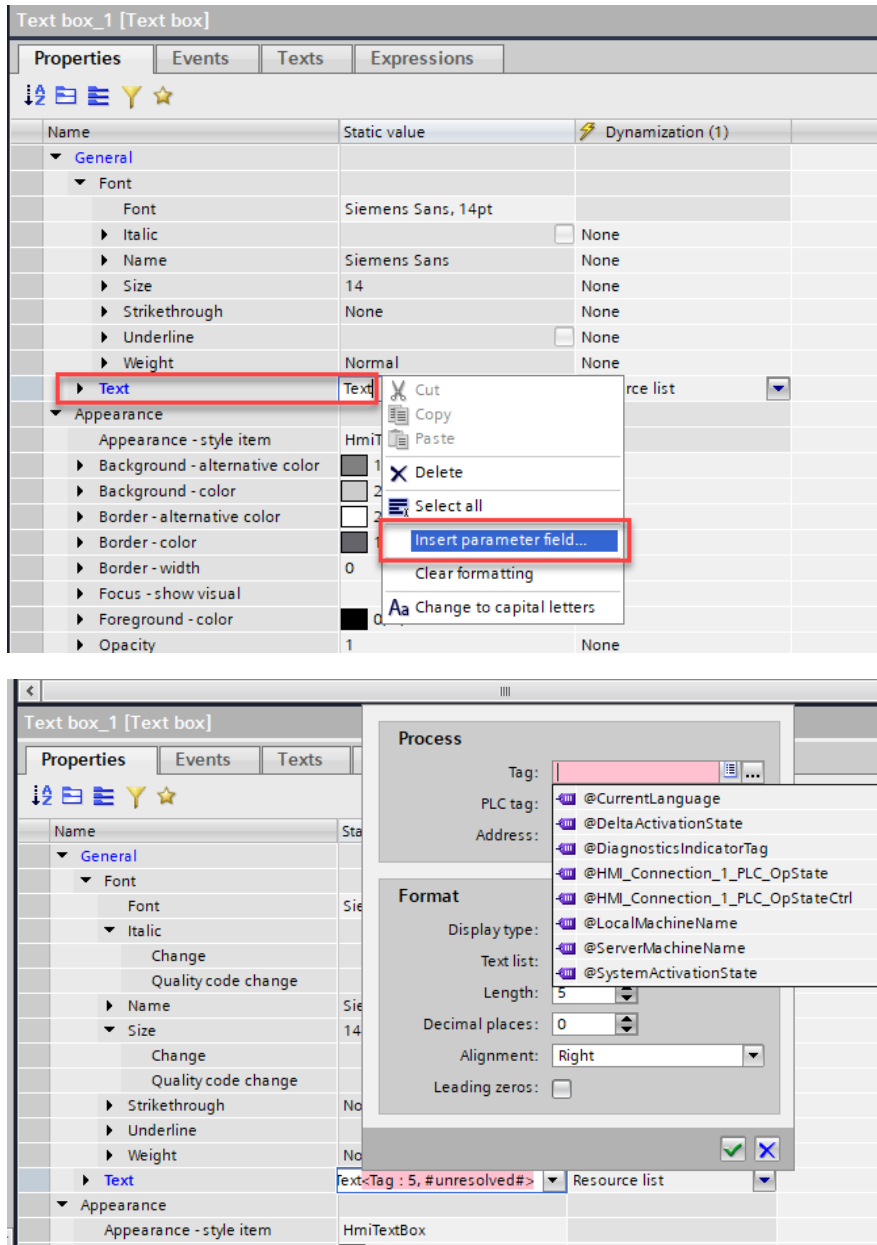


Figure 3-8 Text property parameter field

3.1.1.4. Use Case: Colored Screen Background

Description

In some cases, the default background color of a screen needs to be adapted.

Solution

Instead of adding an additional rectangle to the screen and using its background color as the screen background, change the color of the screen element directly. This is also relevant for Faceplates.

Screen_1 [Screen]			
Properties			
Name	Static value	Dynamization (0)	
▼ Appearance			
▶ Background - alternative c...	235, 235, 235	None	
▶ Background - color	192, 192, 192	None	
▶ Background - fill pattern	Solid	None	
▼ Format			
▶ Alignment - horizontal	Left	None	

Figure 3-9 Background color configuration of a screen element

3.1.1.5. Use Case: Custom Styles V19

Description

It is common that a corporate layout also comes with a corporate color palette and designs. Through the object properties the screen items can be adapted to this layout and design. But also, the overall appearance of objects like buttons can be part of the corporate design or even the visualization of more complex objects like sliders.

Solution

It is always preferable to use the Unified screen objects with their intended function. The reproduction of user-defined designs of an item with the help of several screen items should be avoided, as this leads to a higher number of screen objects and usually to additional unused functions of these auxiliary items. Also, the use of Faceplates as a standardization method for corporate designed objects create an unnecessary overload. Use the [SIMATIC WinCC Unified Corporate Designer](#) to create these designed objects and also colors as part of your own style and use this style in your project.

Inside the corporate designer a new style can be created already based on existing styles. Then you can either change the style of existing objects or also create completely new objects. These styles can also be linked to color palettes and fonts.

When the style design is finished, the style needs to be exported and the generated file must be copied to the project directory (Userfiles > Styles). After these steps the style can be selected in the runtime settings.

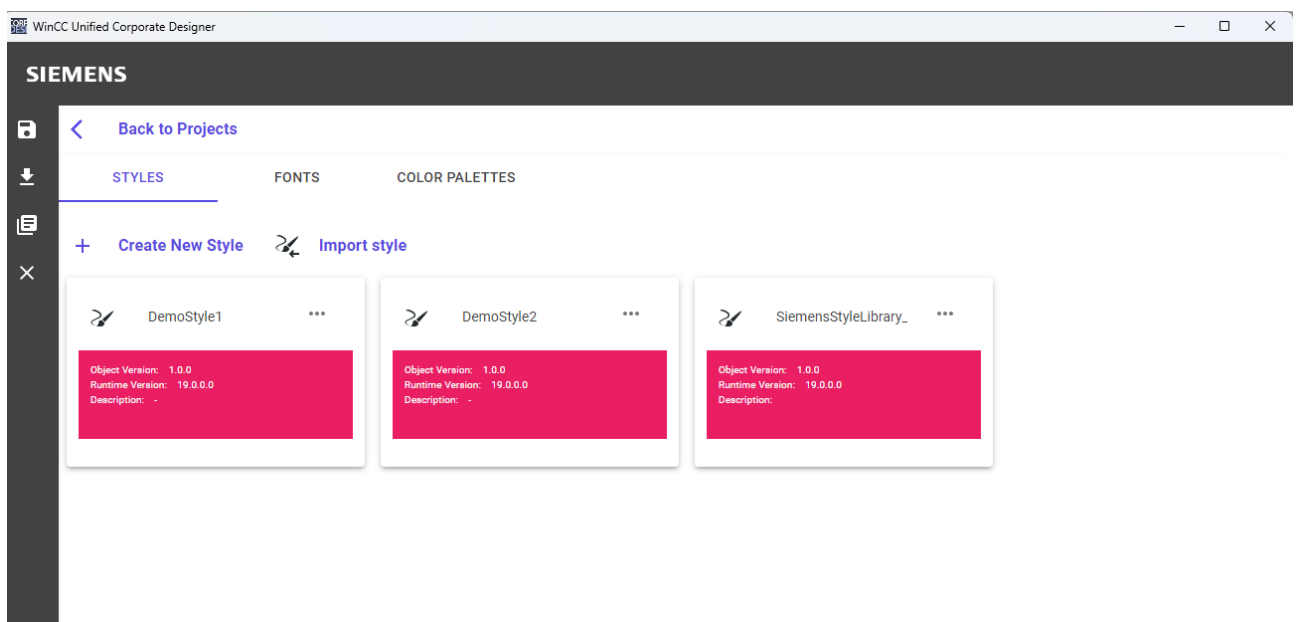


Figure 3-10 Unified Corporate Designer Project View

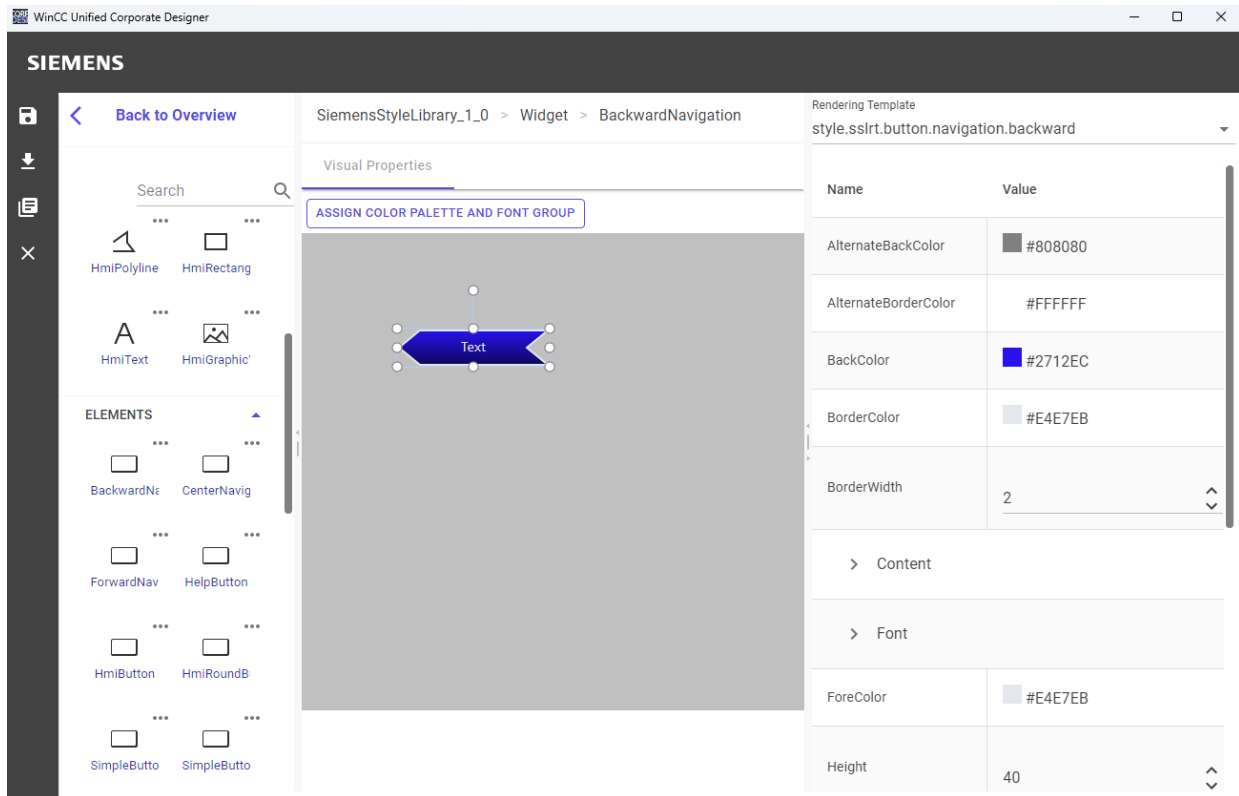


Figure 3-11 Unified Corporate Designer: Edit View

The style that is selected in the runtime settings of the device, can still be changed on demand in runtime. Also, with the new custom styles.

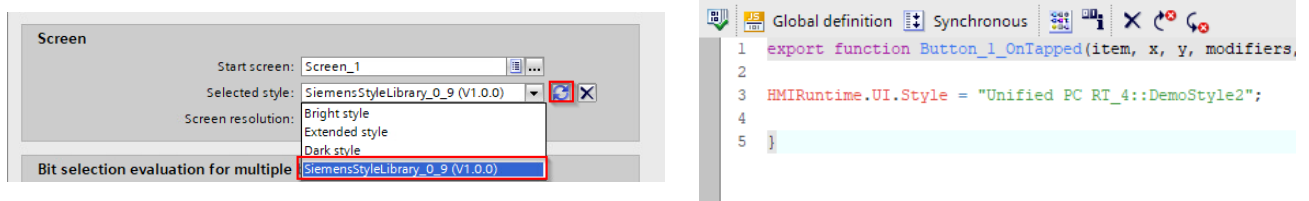


Figure 3-12 Style selection for the Unified Runtime

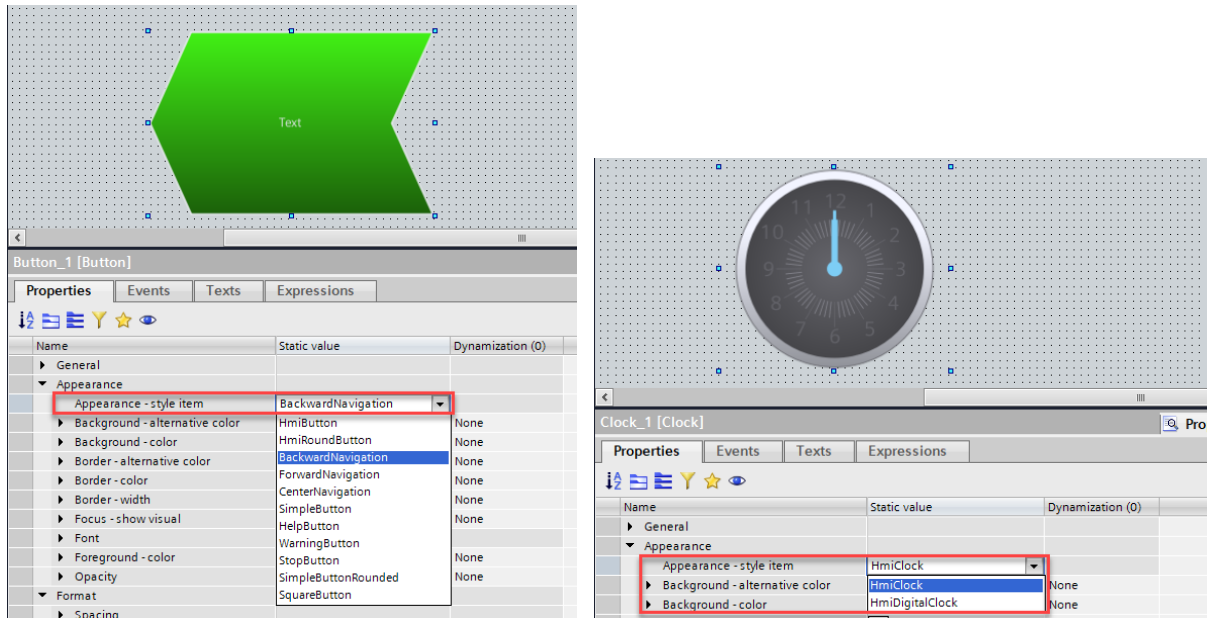


Figure 3-13 Change the style item of system and customized screen objects in engineering



Custom Style

Define a custom style instead of using Faceplates for single object configurations, using the currently available capabilities of the WinCC Unified Corporate Designer.



3.1.2. Tidy up the Screen / Observe System Limits

As you've already seen, there are elements that have more rendering effort and elements that require less. In any case, every element on a screen affects the loading time and there are also system limits defined that must be observed. To achieve the shortest screen loading time, especially shortly before a project is used as productive, all objects that were generated only for the implementing phase for debugging or other supportive reasons, should be deleted again.

- Delete unused objects that are in the background or not visible
- Delete unused objects that are out of the range of the screen. There is also a task card in the layout section available, that indicates the objects out of range.

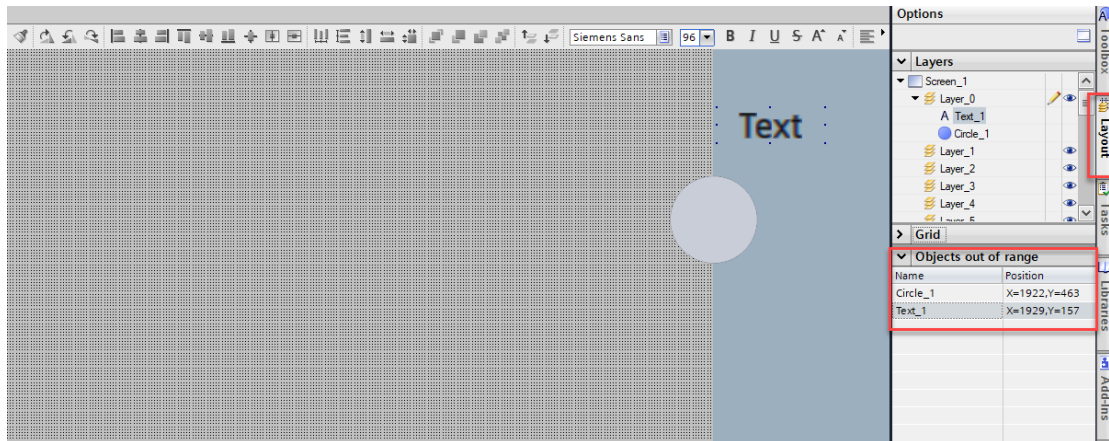


Figure 3-14 Task card objects out of range



System limits

Delete unused and invisible objects to achieve a lower loading effort



3.1.3. Usage of Screen Windows

Screen windows are often used to create a screen layout and to separate individual content from content that is apparent for more views and does not need to be loaded for each screen change (e.g., Header). Also, for pop up screens they are applied. To clarify when to use a screen window as a pop und when to configure it already in runtime, follow the flow chart below. The recommendation is only in terms of a good engineering. Keep in mind that screen windows in general have a different behavior as by system function OpenScreenInPopUp generated popup screen windows regarding zoom, scroll, position, screen layer and live time. So, the specific use case can require explicit one of the implementations.

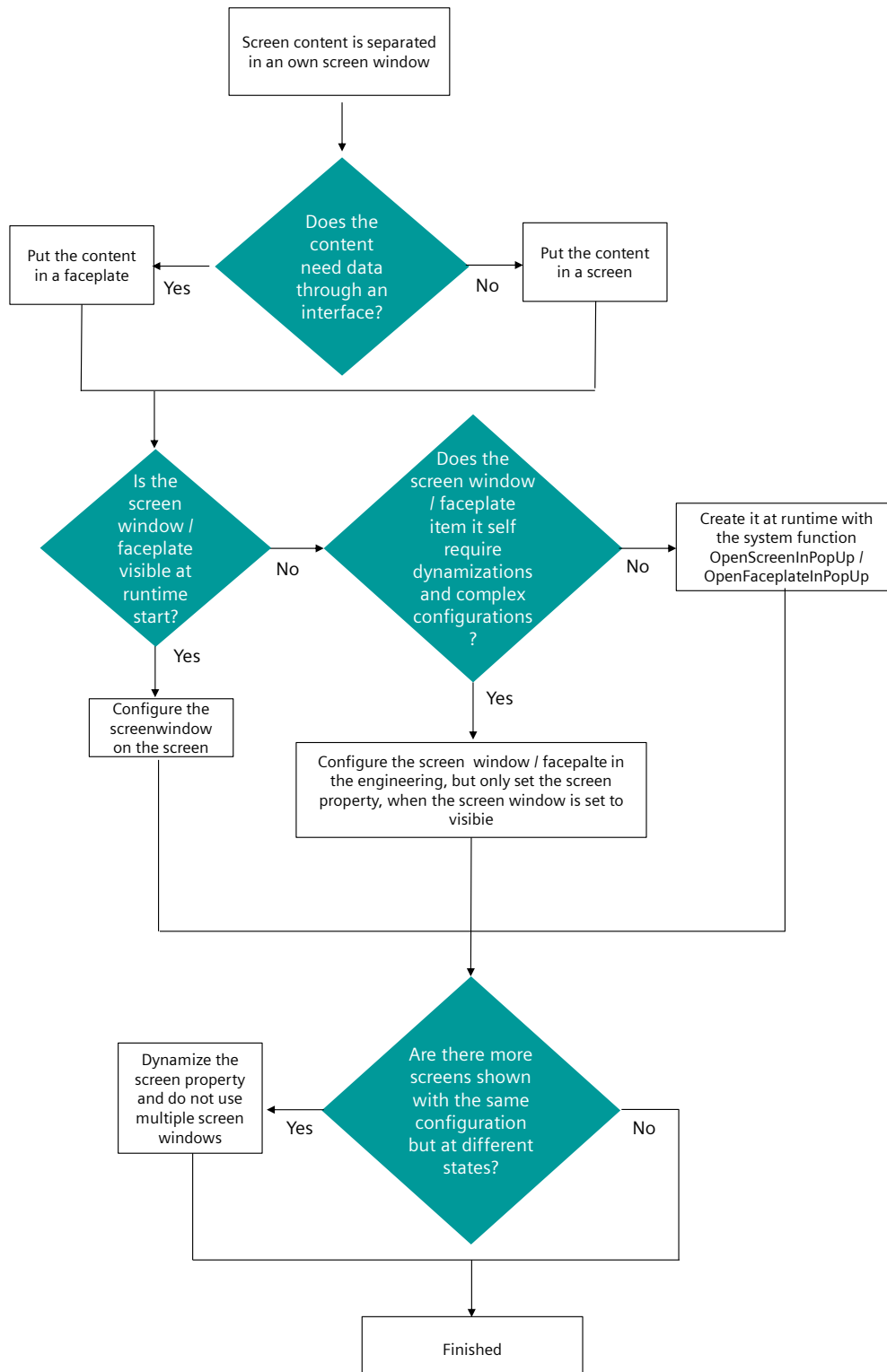


Figure 3-15 Configuration of screen windows

3.1.4. Screen Object Visibility

If the visibility of a screen item is dynamized, set the static value to false, especially when it can be assumed that the used dynamization, e.g., tag dynamization causes an invisible object during screen load.

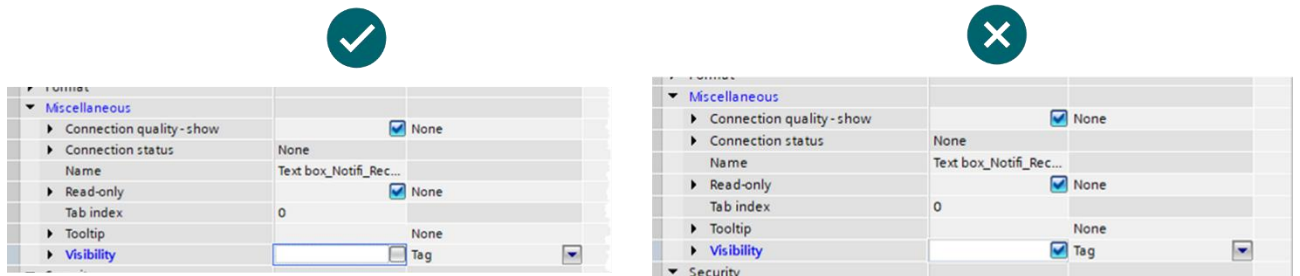


Figure 3-16 Static value for Visibility if the item is assumed to be invisible at screen load

3.1.4.1. Use Case: Visibility Dynamization of multiple Screen Objects by the same Condition

Description

The visibility of more than one screen items needs to be changed in runtime by the same condition.

Solution 1

If the visibility of multiple objects is changed depending on the same condition, define one tag to dynamize the visibility of all objects together.

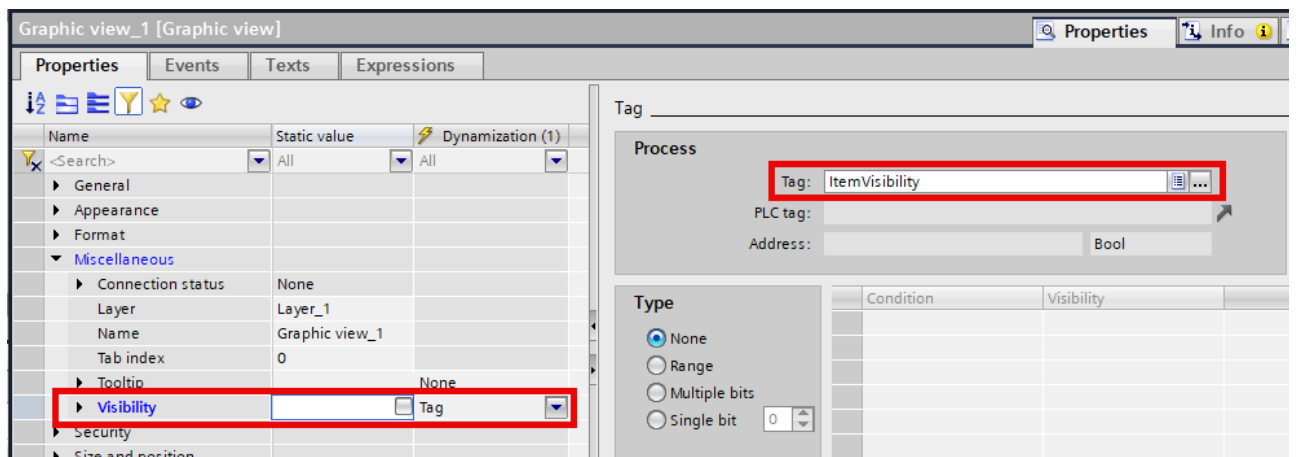


Figure 3-17 Runtime visibility of a screen object

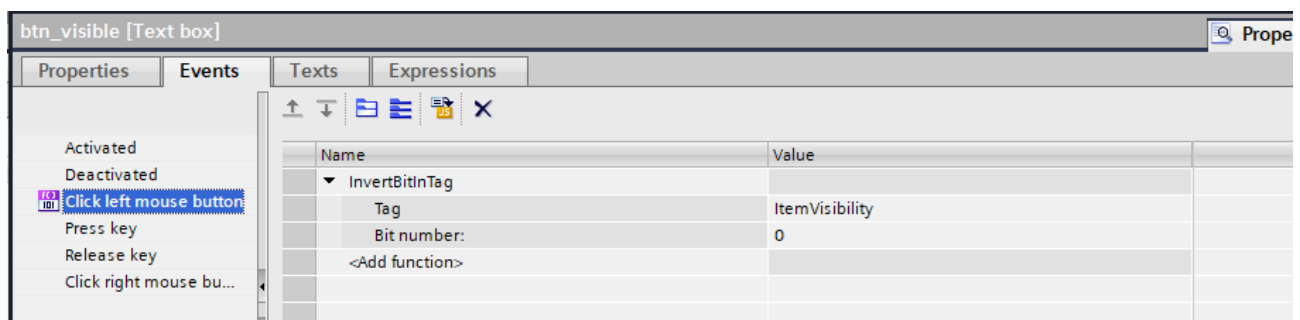


Figure 3-18 Toggle object visibility

Solution 2

If only the visibility is changed depending on the same condition, move all objects in one layer and dynamize the runtime visibility of the layer. The visibility of the layer can be changed also in runtime by scripting.

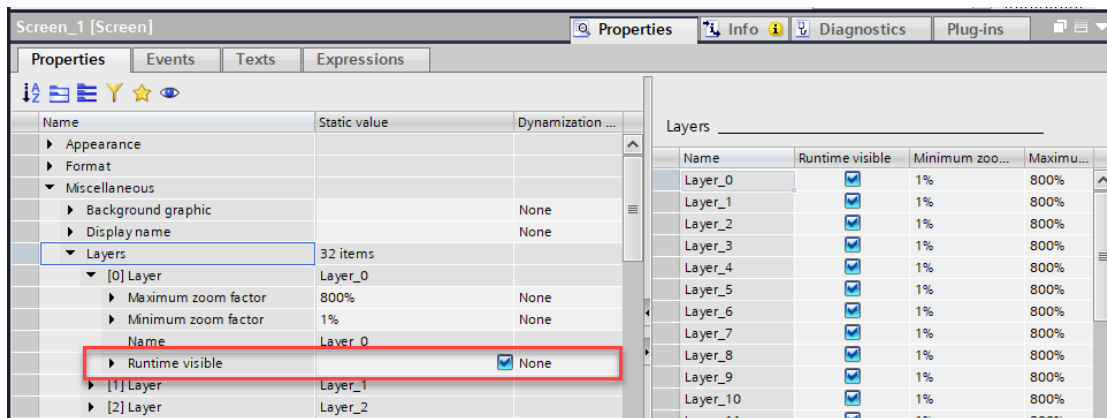


Figure 3-19 Runtime visibility of a screen layer

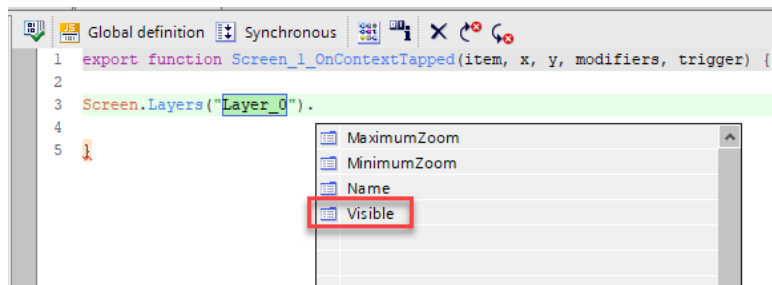


Figure 3-20 Access layer visibility in runtime

Solution 3

If there are more properties that have the same behavior or condition (e.g., background color, authorization...), group the objects and configure the property change for the whole group at once.

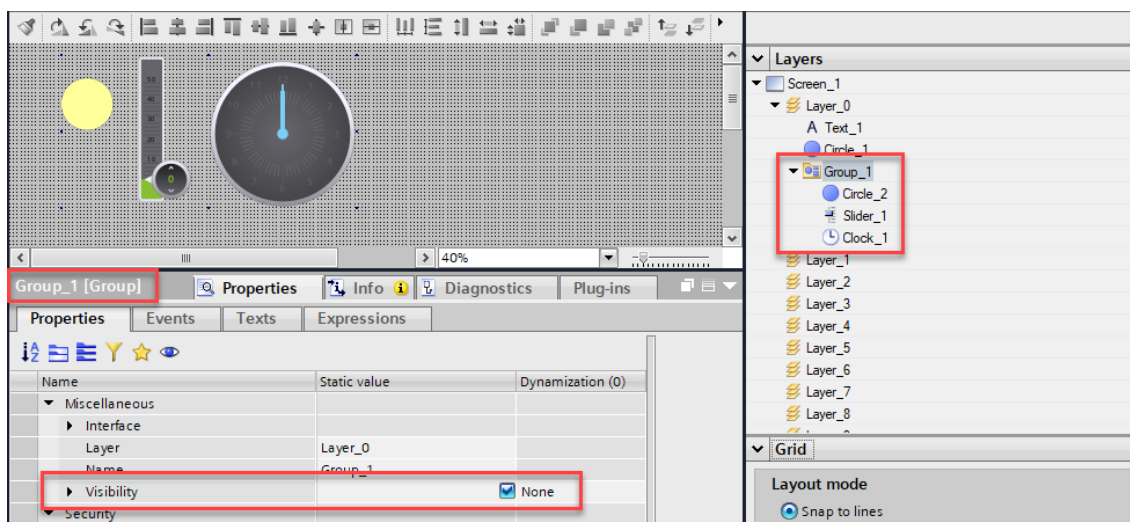


Figure 3-21 Visibility dynamization of a group of screen items



Visibility of screen items

If the visibility of more than one item is dynamized by the same condition, put them together in a group or layer and dynamize its visibility.



3.1.5. Unified Controls

Unified controls make it much easier to implement common use cases such as alarming, reporting and trend control. Nevertheless, there are also aspects in this area that should be considered during use.

3.1.5.1. Use Case: Different Settings in Runtime for Unified Controls

Description

There are a lot of configurations that can be done in the engineering for the Unified Controls. Sometimes a Control is needed in Runtime with different settings. Due to the very limited possibility of dynamizing the controls in the former WinCC Basic, Comfort and Advanced projects, it was common practice to place several statically parameterized controls in the screens of those projects and to make them visible alternately at runtime.

WinCC Unified provides very extensively configurable and dynamizable controls, so that the requirement to make various settings available at runtime can be met by implementing and controlling a single control.

Solution

Change the properties in runtime and do not configure multiple controls or even change the screen to realize different settings. Therefore, use the system function "SetPropertyValue" or the object model "Screen.Items(« Itemname »)". The name of the properties can be easily copied in the engineering from the properties tab.

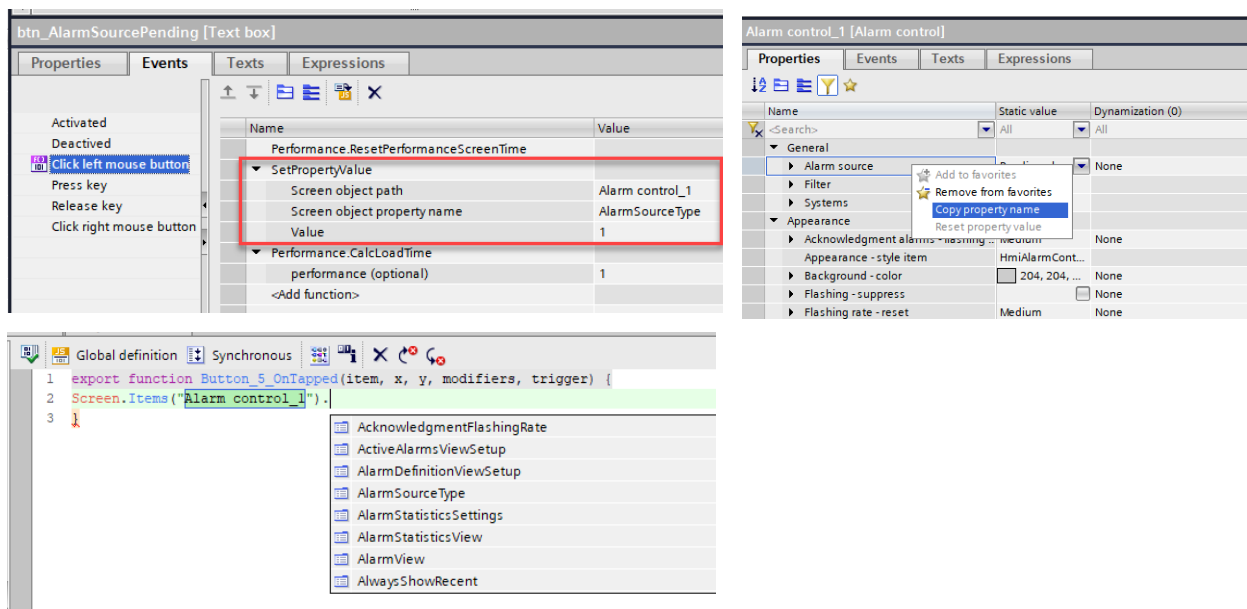


Figure 3-22 Accessing Unified control properties

All accessible properties can be found in the WinCC Unified object model directly in the TIA help or on the [SIMATIC HMI WinCC Unified V19 - Programming reference](#) on SIOS.



Unified Controls

Reconfigure the control in Runtime, when different settings are needed, instead of having multiple controls.



3.1.5.2. Use Case: Alarm Control

Description

For accessing alarms in general an alarm view is provided. This control comes with a lot of functionality as well in the engineering to configure but also in runtime. Additionally, there are also a lot of system function, to access the same functionality by scripting, without the Unified control.

Solution

As we in general recommend designing a screen layout with screen windows (see chapter 3.1.3 and Figure 2-2), there are screens that change the content and screens that remain. The alarm control belongs to the screen items with a huge footprint (see chapter 3.1.1 and Figure 3 2) therefore it is recommended to put it in screen windows, that are not reloaded that often like e.g., the header. Since these screen windows like the header are not that big, you can configure the alarm view as an alarm line with the following steps:

1. Use the simplified appearance style

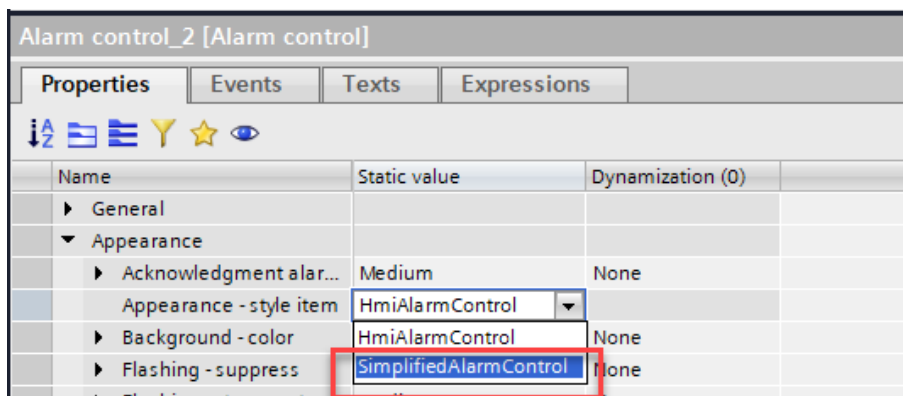


Figure 3-23 Simplified alarm control style item

2. Set the window settings to “always on top”

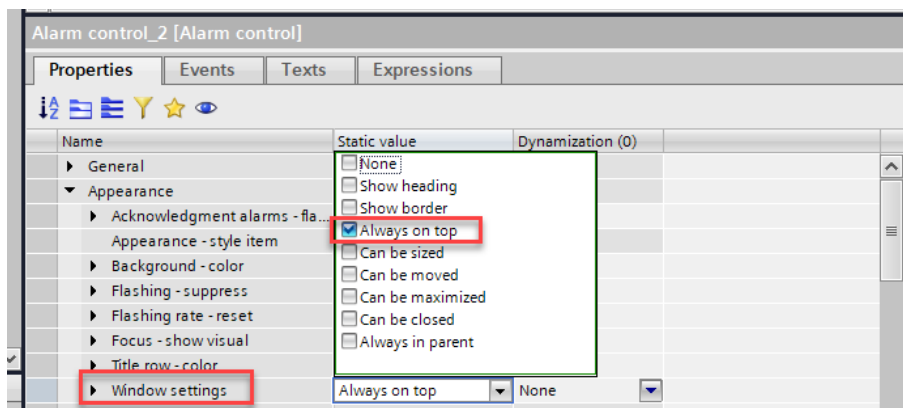


Figure 3-24 Alarm control window settings

3. Set sorting direction to ascending

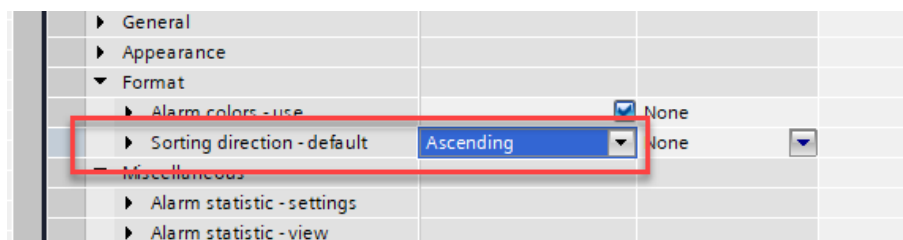


Figure 3-25 Alarm control sorting direction

4. Go to Miscellaneous > Alarm view > Header-settings and change column header to "None" and row header to "None"

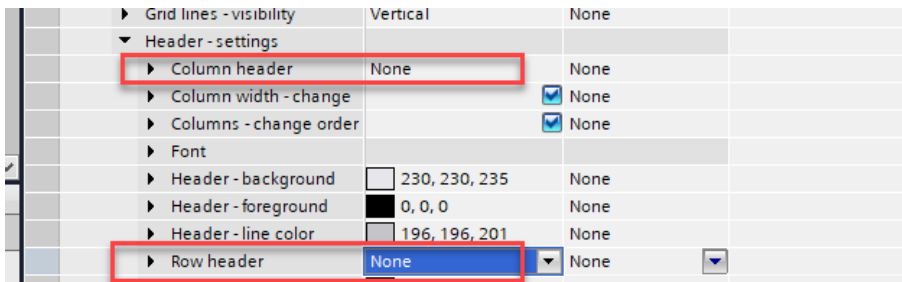


Figure 3-26 Alarm control column and row header

5. Go to Miscellaneous > Alarm view and collapse the scroll bars

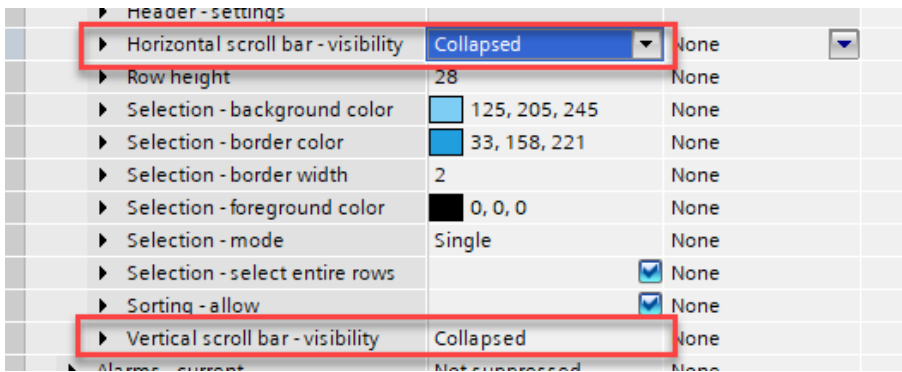


Figure 3-27 Alarm control horizontal and vertical scroll bars

6. Adapt the size to one or only a view lines

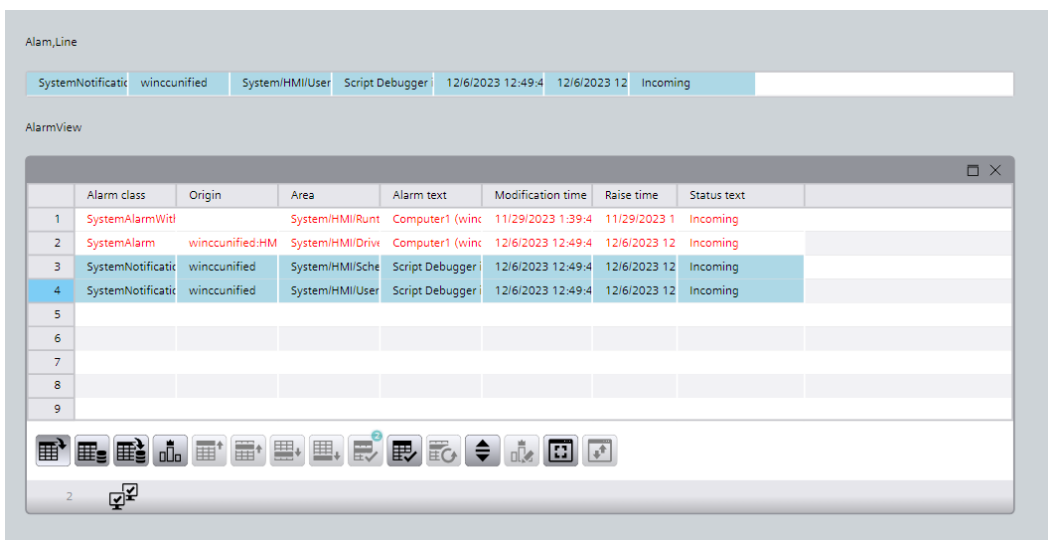


Figure 3-28 Alarm line vs Alarm View

If the alarm view with the whole functionality is necessary, show it as a pop up on demand.

3.1.5.3. Use Case: Alarm Line Custom Solution

Description

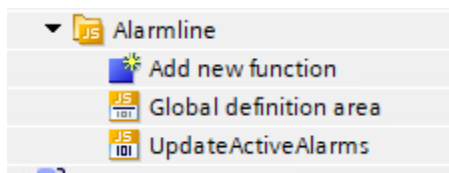
For accessing alarms in general an alarm view is provided. As mentioned before ([Use Case: Alarm Control](#)), when this information needs to be displayed in a permanent visible screen window, i.e. the header, the simplified appearance style is a good solution. But this is not recommended if the alarms need to be displayed in a screen that is getting reloaded often.

Solution

Using an Alarm Line to display only the latest triggered alarms (i.e. only 3 out of 8 pending alarms) can be a good solution for avoiding using an alarm control when you do not need all its functionalities. This is an open-source scripting solution provided free of charge via [Github](#) and is a good option if you only need to display the alarm name, raise time, status and address of the latest alarms. You can implement the Alarm Line custom solution with the following steps:

1. Create a new Global Module in your TIA Portal project and copy the code from [Alarmline.js](#) (lines 1-171) to the Global definition area. Here, the Alarm Manager class is defined, and it has several methods and properties (initializes the instance of the Alarm Manager with the given options, starts the alarm subscription, sets the sorting order and stops the subscription)
2. Add the new function "UpdateActiveAlarms" and copy the corresponding code from [Alarmline.js](#) (lines 174-201). This function will update the alarm tags based on the provided array of alarms. It creates a tag set and updates the values based on the alarms.

3.



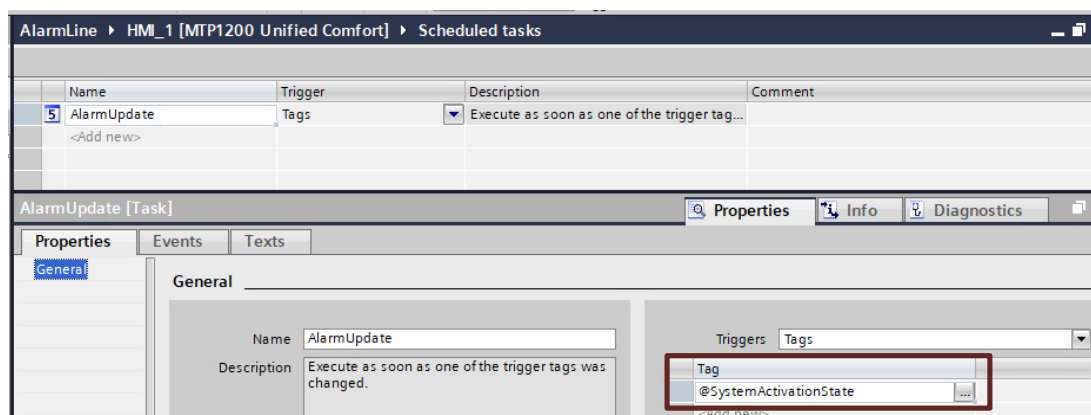
4.

Figure 3-29 Global modules of the custom alarm line solution

5.

6. Create a new Scheduled task, i.e. "AlarmUpdate", and select the tag "@SystemActivationState" as trigger. This task will update the alarms, it starts the alarm subscription based on the system activation state.

7.



8.

Figure 3-30 Scheduled Task of the custom alarm line solution

9. Use the code from [ScheduledTasks.js](#) in the Event > Update of your newly created task. In the Global definition, copy code from lines 1 to 9 and, for the event itself, use lines 11 to the end.
10. For the visualization of the alarms, you can configure it with basic objects and elements from the toolbox (see [Demoproject](#))
11. Create the required HMI tags for displaying the Alarm Line values and dynamizing properties. The number of tags needed will vary depending on the maximum number of alarms you want to display at the same time.

Custom_AlarmLine ▶ HMI_1 [MTP1200 Unified Comfort] ▶ HMI tags ▶ AlarmLine [24]

AlarmLine				
Name	Data type	Connection	PLC name	
Alarm1_Address	WString	<Internal tag>		
Alarm1_AlarmText	WString	<Internal tag>		
Alarm1_BackColor	UDInt	<Internal tag>		
Alarm1_DateTimeRaised	DateTime	<Internal tag>		
Alarm1_Flashing	Bool	<Internal tag>		
Alarm1_MachineUnitAssyPart	WString	<Internal tag>		
Alarm1_Status	WString	<Internal tag>		
Alarm1_TextColor	UDInt	<Internal tag>		
Alarm2_Address	WString	<Internal tag>		
Alarm2_AlarmText	WString	<Internal tag>		
Alarm2_BackColor	UDInt	<Internal tag>		
Alarm2_DateTimeRaised	DateTime	<Internal tag>		
Alarm2_Flashing	Bool	<Internal tag>		
Alarm2_MachineUnitAssyPart	WString	<Internal tag>		
Alarm2_Status	WString	<Internal tag>		
Alarm2_TextColor	UDInt	<Internal tag>		
Alarm3_Address	WString	<Internal tag>		
Alarm3_AlarmText	WString	<Internal tag>		
Alarm3_BackColor	UDInt	<Internal tag>		
Alarm3_DateTimeRaised	DateTime	<Internal tag>		
Alarm3_Flashing	Bool	<Internal tag>		
Alarm3_MachineUnitAssyPart	WString	<Internal tag>		
Alarm3_Status	WString	<Internal tag>		
Alarm3_TextColor	UDInt	<Internal tag>		
<Add new>				

12.

Figure 3-31 Custom Alarm Line Tags

13. With this approach, there are some parameters that can be modified to better suit your requirements:

- **Max. number of alarms** : the maximum number of alarms displayed at once can be modified in the Global definition of the Alarmline global module, in line 164. Take into consideration that, if more alarms are to be displayed, more HMI tags and basic elements are needed for displaying them.

Custom_AlarmLine ▶ HMI_1 [MTP1200 Unified Comfort] ▶ Scripts ▶ Alarmline ▶ Global definition area

```

145 return () => {
146     if (timerFlag === null) { // If there is no timer currently running
147         this.#sendUpdate(); // Execute the send update
148         timerFlag = HMIRuntime.Timers.SetTimeout(() => { // Set a timer to clear the timerFlag after the speci
149             timerFlag = null; // Clear the timerFlag to allow the main function to be executed again
150             if (this.#hasChanged) { // if a change occurred in the meantime, it must be sent to the visualizatio
151                 this.#hasChanged = false;
152                 this.#throttledSendUpdate();
153             }
154         }, delay);
155     }
156 };
157 }
158 }
159
160 /*
161 Global definition area of module "Alarmline"
162 */
163 //Number of configured Alarmline tags. When maxAlarmLineAlarms is increased also tagSetTags needs to extend
164 const maxAlarmLineAlarms = 3;
165 const tagSetTags = ['Alarm1_DateTimeRaised', 'Alarm1_AlarmText', 'Alarm1_MachineUnitAssyPart', 'Alarm1_Status'

```

Figure 3-32 Custom Alarm Line: MaxAlarmLineAlarms

- **Alarm language**: in the scheduled task event code, the language used for the alarms can be set by changing the value (decimal language ID) in line 7. In this example, "1033" refers to English (here you can find [further language IDs](#)). You can also use the value "127", which is the default language configured in your HMI.

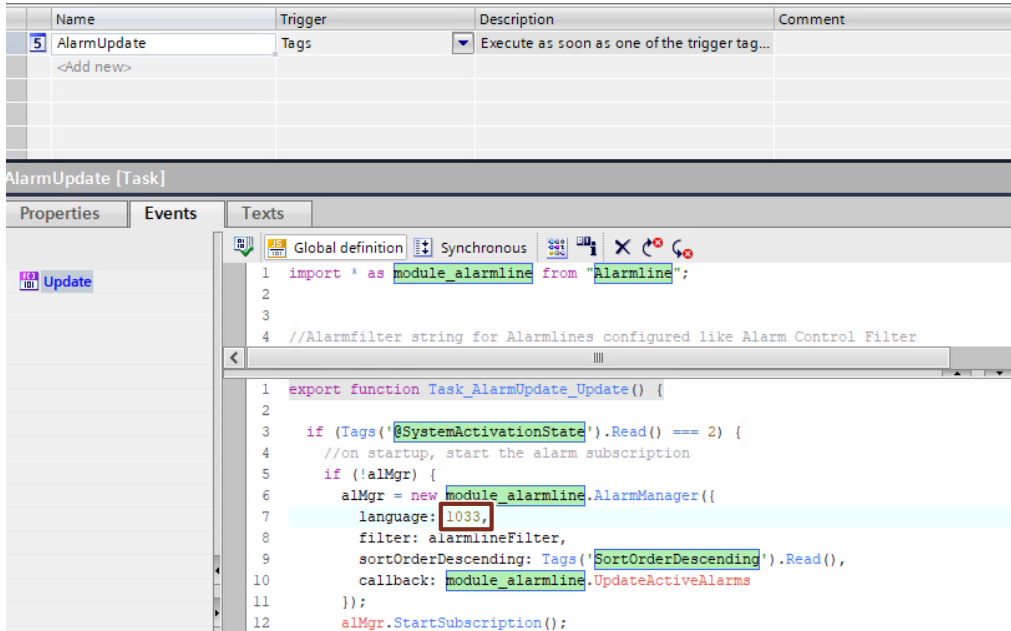


Figure 3-33 Custom Alarm Line Language

- **Alarm filter** : in the global definition of the scheduled task, the filter for the alarms is configured. This can also be modified so that it fits your needs.
- **Update interval** (optional): in the Scheduled task Event code, you can add the parameter “delayInMilliseconds” to the Alarm Manager() call. The suggested time is 250ms or more, since below this value there is no real benefit. Notice that, when setting it to a higher number, it does not mean that you will not be updated in time. This parameter works as follows: imagine that we have set 10.000ms as delay, whenever there is a new alarm raised it will be sent directly, as soon as possible, to the visualization. If after 2s another alarm is coming, it will not be sent to the visualization directly, but it will be visible after 8s (10s delay – 2s = 8s). This will allow us to see the alarms in time and also prevent overflowing of the system. This is an optional parameter, so you can choose the delay you prefer or not use it at all.

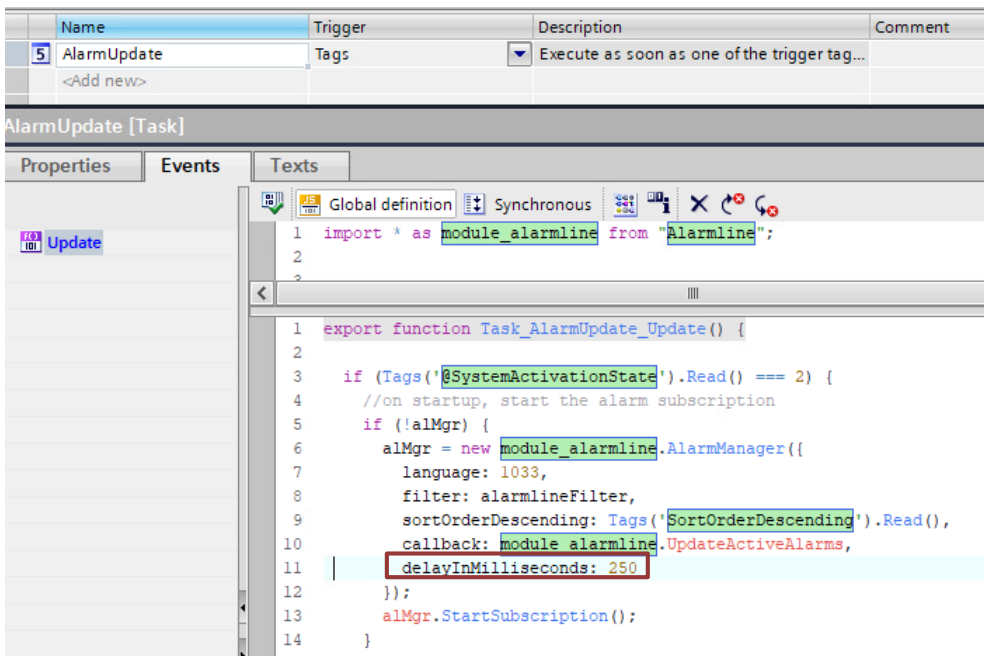


Figure 3-34 Custom Alarm Line delay in Milliseconds

3.1.6. Graphics and SVGs

Additionally, to all the textual information, an HMI usually also contains images. These can be small icons, bigger images, or animated graphics / dynamic SVGs. Depending on the use case there are also recommendations on how to use images.

Image format:

- **JPG** (joint photographic experts group) are recommended for UCPs as the format reduces the size of large files. So there is also a loss of quality through the compression, but the files can also be decoded faster. When a graphic is needed in a high resolution, a JPG can also be used in a high quality. JPG does not automatically mean less quality.
- **PNG** (portable network graphic) is a raster graphics format with lossless data compression. Therefore it is recommended when transparency is required or when exact pixel accuracy is important.
- **SVG** (scalable vector graphics) is used for displaying two-dimensional graphics, diagrams and illustrations on websites. As a vector format, SVG images can be scaled to different sizes without affecting the resolution. Therefore, this format is recommended when high quality zooming is required or dynamic SVGs are used. In other cases, it is better to convert the file in the size used to a PNG or, if no transparency is required, to a JPG, whereby the largest size used for multiple use should be selected.

NOTE

The formats can be divided into two groups:

- **Raster graphics format**, that relies on pre-rendered bitmaps and is therefore less demanding on system resources and recommended as a **runtime format** (JPG, PNG)
- **Descriptive graphics format**, that uses code to specify how the graphic should appear. It will only be rendered during Runtime, which makes it easily adaptable in size (e.g. when zooming), and well suited as a primary **library format** (SVG)



Graphics

Regardless of the graphic format, the file size should not be larger than the maximum size /resolution that is needed for this HMI



Graphics Format

The recommended order for selecting the graphic format is

1. JPG
2. PNG
3. SVG

If there are further requirements for the graphic, still the SVG can be the most suitable. But select it carefully!



3.1.6.1. Use Case: Visualize Patterns and composed Objects

Description

Some objects or patterns can be created by multiple objects. Furthermore, these combined objects might be reused multiple times on the screen.

Solution

Do not create pattern by the use of single items but combine them to one image. This reduces the number of objects on the screen. Even if the objects are basic like rectangles, the combination to one object reduces the rendering effort. When dynamizations are necessary create a custom dynamic SVG (see next use case). This also reduces the number of object containers on the screen and helps to comply with the system limits.

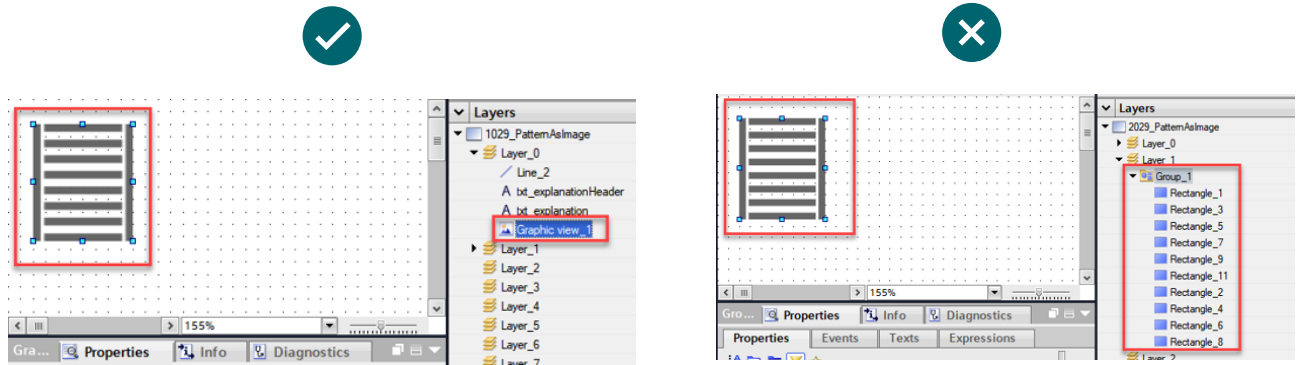


Figure 3-35 Pattern as SVG example



Combined objects

Reduce the number of elements on the screen, by creating SVGs and dynamic SVGs of combined objects



Also use graphics in their original resolution and only as large as it is necessary.

3.1.6.2. Use Case: Composed Objects with Dynamization → dynamic SVG

Description

Some illustrations with a dynamic visualization can be either created by the composition of multiple dynamized screen objects or directly by a dynamic SVG.

Solution

As already described in the previous use case, to minimize the number of screen items per screen, composed objects should be combined to one object container. There are dynamic SVGs available for a wide range of standard components in the IndustryGraphicLibrary. Before building your own component with several screen items, check if there is already a solution in the graphic library. Through the interface some properties can be configured and changed in runtime dynamically. More information about the usage of the dynamic widgets can be found in the [Unified V19 Manual](#).



Figure 3-36 Industry Graphic Library

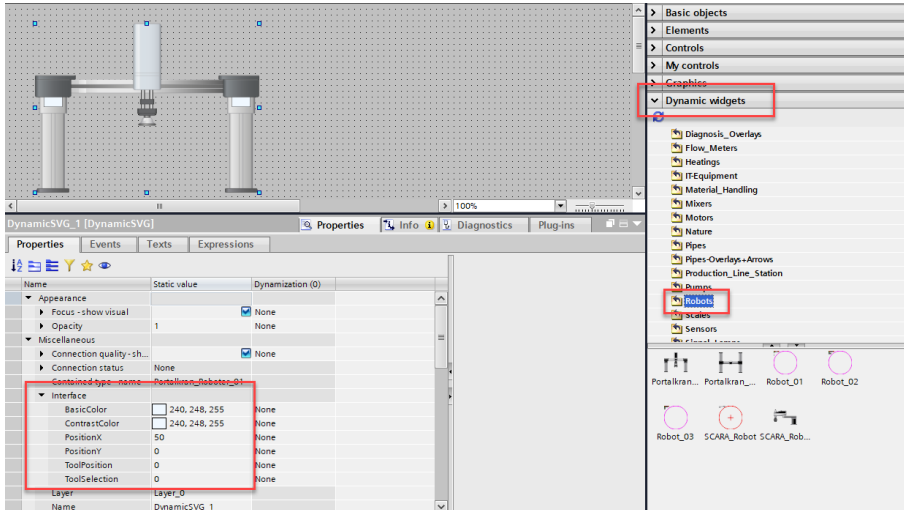


Figure 3-37 Robot dynamic SVG from Siemens Graphic Library

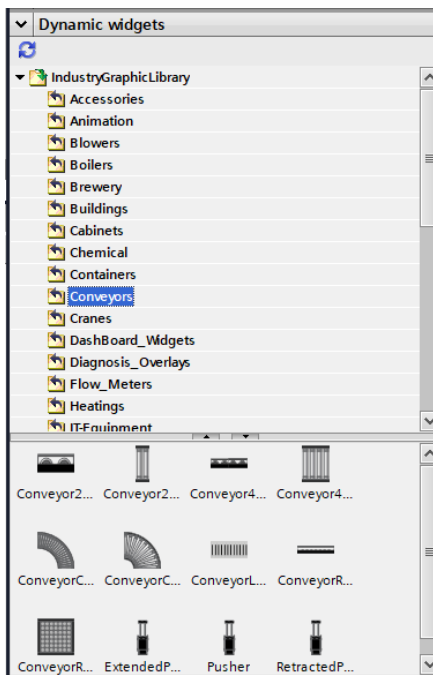


Figure 3-38 Dynamic Widgets Conveyor examples

3.2. Use of Dynamizations

In this document different types of dynamizations are mentioned.

3.2.1. Simple Tag Dynamization

A tag dynamization is shown in the following figure. It can have types of configurations. You can take “None” to directly transfer the tag value to the property value. With the “Range” type, a condition can be specified for several tag value ranges and linked to a property value. The “Multiple bits” type makes it possible to change the property depending on various single bit positions of the dynamization tag. And finally, the “Single bit” type is limited to one bit of the dynamization tag, to specify conditions for the property.

Every time the tag value changes, the property value is adapted.

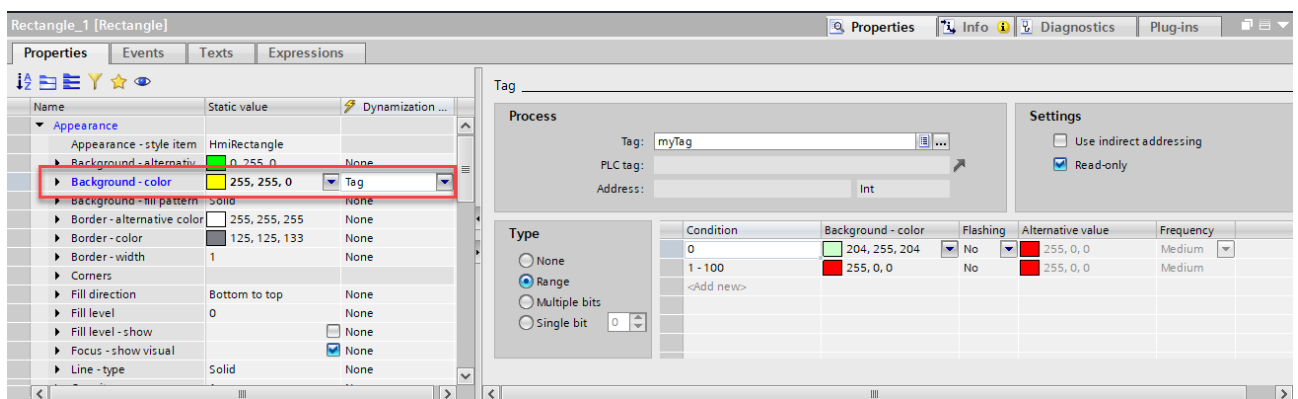


Figure 3-39 Tag dynamization

In this context it is important to mention, that there is also a “Change” event. Through this event, an additional script can be executed if the property is changed at runtime, e.g., if the tag value of the dynamization tag is changed. That happens when this event is triggered during the screen load process can be seen in detail in [Figure 2-9](#).

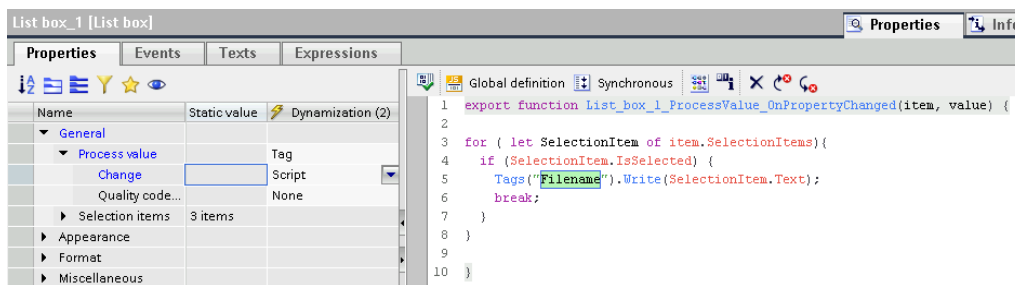


Figure 3-40 Tag dynamization – change event

3.2.2. Script Dynamization

The second dynamization is a script dynamization. The return value of this script defines the property value after the script is executed. To execute the script a trigger needs to be specified. Information about triggers is given in section [3.4.1. Script Triggers](#).

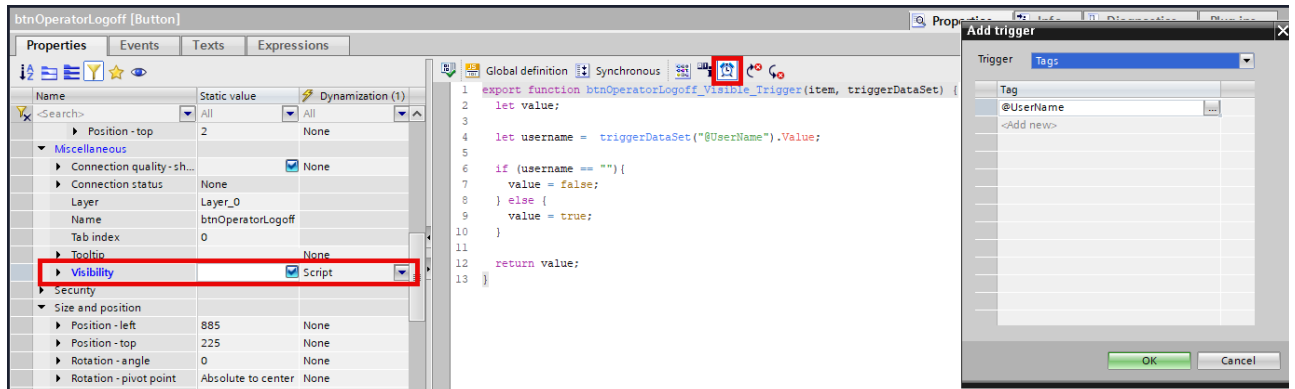


Figure 3-41 Script dynamization

3.2.3. Expressions

The Expressions are another way to dynamize a screen item property. To configure them, there is an additional tab available. With an Expression a condition based on multiple tags can be defined and multiple properties can be changed by the conditions.

- The Expression is evaluated when one of the tag values changes.
- Properties change as soon as the result of the Expression changes.
- If none of the conditions returns TRUE, the default value is assumed.
- If multiple conditions are defined, the first condition in the list that returns TRUE is applied.
- Expressions that cannot be evaluated are skipped, e.g. because of a syntax error or a tag that cannot be accessed.

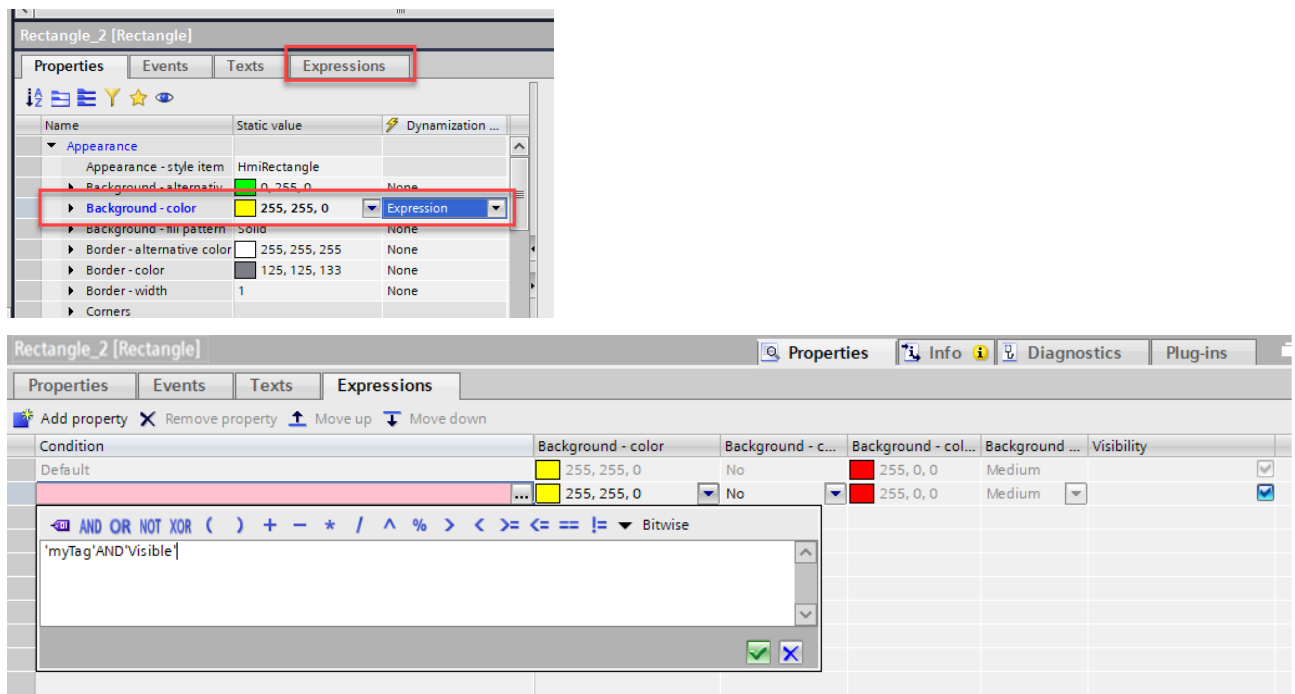


Figure 3-42 Expression as a dynamization

Be aware of the order in which the Expressions are defined. They are executed from top to bottom and once a condition is true, the others are not executed anymore. In the following example, the order must be as follows. If it is defined in reverse order, the AND condition is never evaluated, as the OR condition is always fulfilled first.

Condition	Background - color	Background - color -> Fla..	Background - color -> Alt..	Br
Default	255, 255, 0	No	255, 0, 0	M
'TagA'AND'TagB'	255, 255, 0	No	255, 0, 0	Mk
'TagA'OR'TagB'	0, 255, 0	No	255, 0, 0	M

Figure 3-43 Expression execution order

3.2.3.1. Use Case : Dynamize a Property depending on Single Bits V19 Upd2

Definition

Some Use Cases require to dynamize a property depending on single bits in a word. Previously, this had to be done with scripts like the following example shows. A different color value is returned depending on bits.

```

1 export function Rectangle_3_BackColor_Trigger(item, triggerDataSet) {
2   RMIruntime.Tags.SysFor.ShiftAndMask("Word2", "TargetWord", 4, 3);
3   switch (Tags("TargetWord").Read()) {
4     case 1:
5       return 0xFF5ED1AD; //green
6     case 2:
7       return 0xFFEACE21; //yellow
8     case 3:
9       return 0xFFDE3858; //red
10    default:
11      return 0xFFB58EC5; //grey
12  }
13 }
    
```

Figure 3-44 Script to dynamize property depending on certain bits using ShiftAndMask()

Solution

Use bitwise operations in Expressions V19
Upd2 to replace the script dynamization on the property. The SR8()-Operator is used to shift the bits to the right and the AND8()-Operator is used to mask.

Name	Static value	Dynamization (1)
Appearance - style item	HmiRectangle	
Background - alternati...	0, 255, 0	None
Background - color	181, 190, 197	Expression
Background - fill pattern	Solid	None
Border - alternative color	255, 255, 255	None
Border - color	125, 125, 133	None
Border - width	0	None
Corners		
Fill direction	Bottom to top	None

Figure 3-45 Expression Property

Condition	Background - color	Background - color -> Fla..	Background - color -> Alt..	Background - color -> Fr...
Default	181, 190, 197	No	255, 0, 0	Medium
AND8(SRB("Word2",4),3) == 1	94, 209, 173	No	255, 0, 0	Medium
AND8(SRB("Word2",4),3) == 2	234, 206, 33	No	255, 0, 0	Medium
AND8(SRB("Word2",4),3) == 3	222, 56, 88	No	255, 0, 0	Medium
<Add new>				

Figure 3-46 Expression with Shift and Mask operators

3.2.4. Further Options

Depending on the property it is also possible to use a resource list or flashing as a dynamization. The flashing option is available for color properties. The resource list for text properties.

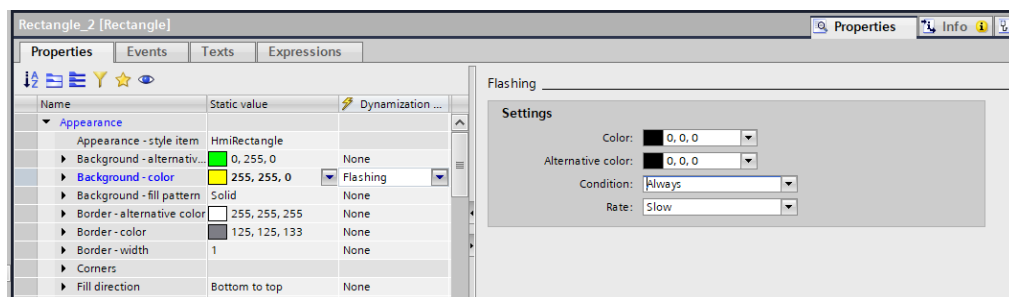


Figure 3-47 Flashing dynamization for color properties

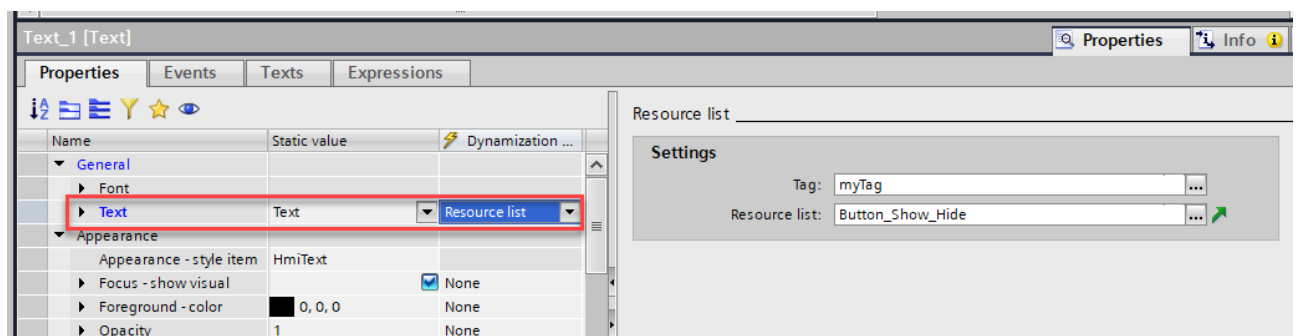


Figure 3-48 Resource list dynamization for text properties

In general, it can be said that Expressions and tag dynamizations are the preferred solution for property changes. But if a script can be used to avoid multiple screen objects, scripts are recommended (see chapter [2.2.1](#)).



Dynamizations

Use, if possible, tag dynamization. When more complex logic is required use Expressions.
Try to avoid script dynamizations!



Information about the different trigger types (cyclic, tag, alarm, event-driven,...) are documented in detail in [SIMATIC WinCC Unified – Tips and Tricks for Scripting \(Java Script\)](#).

3.3. Use of Faceplates

Faceplates are used to bring standardization to your projects. If you want to use them, make sure you do configurations with multiple screen objects. The use of a Faceplate for individual objects, such as a single rectangle, is not recommended for performance reasons, since the additional overhead due to the interface. Use the WinCC Unified Corporate Designer

for this use case (see chapter [3.1.1.5. Use Case: Custom Styles](#) ^{V19}).

3.3.1. Essential Insights into Faceplate Usage

When using Faceplates, it's essential to pay attention to general project considerations to achieve optimal performance. Firstly, assess the compatibility of Faceplates with versions prior to V19 and explore reimplementations possibilities with newer features. Additionally, ensure the following:

- Use the Faceplate in the highest possible version (same as device version)
- Use Faceplates screen objects with sophisticated and extensive logical configuration
- Chose appropriate datatypes in the Faceplate interface
- Keep in mind the system limits [19](#) of a screen when creating a Faceplate. Sample calculation for a Unified Comfort Panel 7-12 inch:

Number of objects per screen for a UCP 7-12": 800

20 Faceplates on the Screen

20 Additional Objects on the Screen

→ 800 - 20 Screen Objects - 20 Faceplates = 760 Objects that are remaining for the Faceplates

→ 760 / 20 = 38 Objects that can be used per Faceplate in this use case

3.3.1.1. Use Case: Faceplate that is not always Visible in Runtime ^{V19}

Description

A Faceplate, which content is only necessary on demand, can be shown as a popup or can already be configured on screen in the engineering but set to invisible-.

Solution

Activate the suspendable flag for these Faceplate instances. With this setting the cyclic script or scripts triggered by tag changed are not executed when the Faceplate is not visible. Not visible refers also to the case that it is out of the current visible screen area.

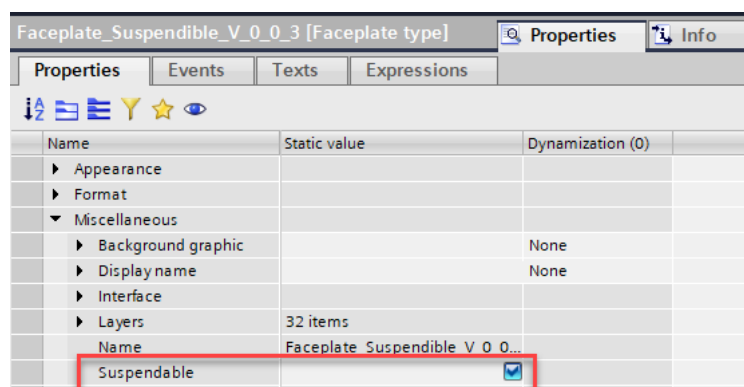


Figure 3-49 Faceplate property suspendable

3.3.1.2. Use Case: Strings in the Property Interface

Description

Within the Faceplate property interface different data types can be selected. For a string there are two options, the Multilingual text and the WString (configuration string).

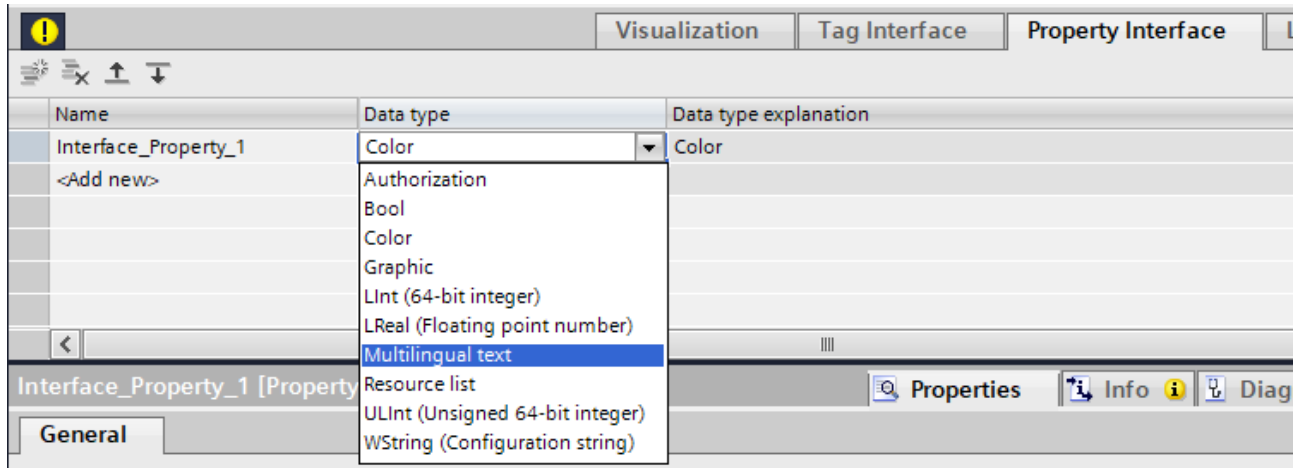


Figure 3-50 Data Types for a Property of a Faceplate interface

Solution

If you need a string that is used for a tag dynamization inside a Faceplate type, select the multilingual text. A configuration string cannot be linked to a property as a dynamization, but only through scripting.

Configuration strings are only recommended for the transfer of configuration settings e.g., trends in a trend control or alarm filters.

Multilingual strings also have the benefit of saving the text information for the different runtime languages and can be switched during runtime.

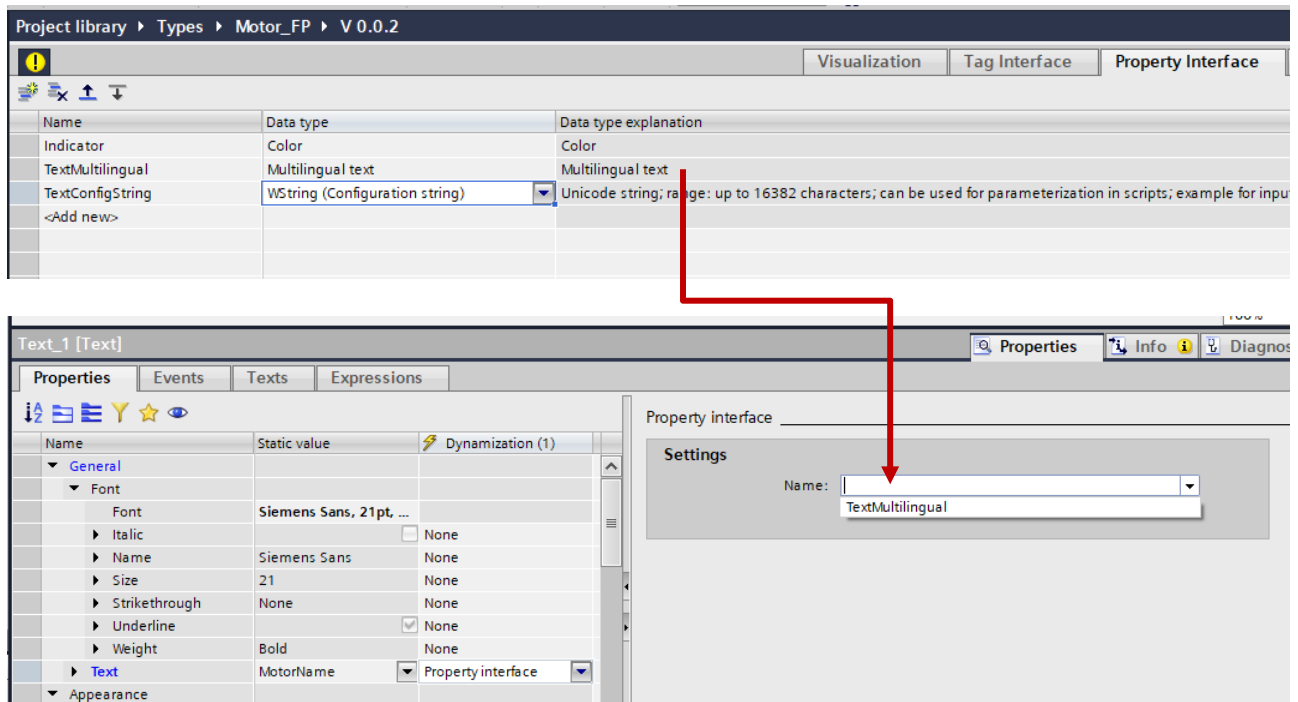


Figure 3-51 Multilingual text and configuration string in Faceplate property interface

3.3.1.3. Use Case : Dynamic Data Connection for hierarchical Faceplates

Description

Faceplates can be applied to visualize content based on a standardized data connection to the PLC. Although several Faceplate instances can be displayed on the screen simultaneously with their respective data connection, in many cases it is sufficient to have only one instance visible at a time and instead dynamize the interface property connection.

Solution 1 – Dynamic interface change using system functions V18

Until V18 you have to connect the tag properties of the Faceplate interface with a static HMI tag. To manipulate the interface connection with the corresponding tag you can use the system function *SetPropertyValue*.

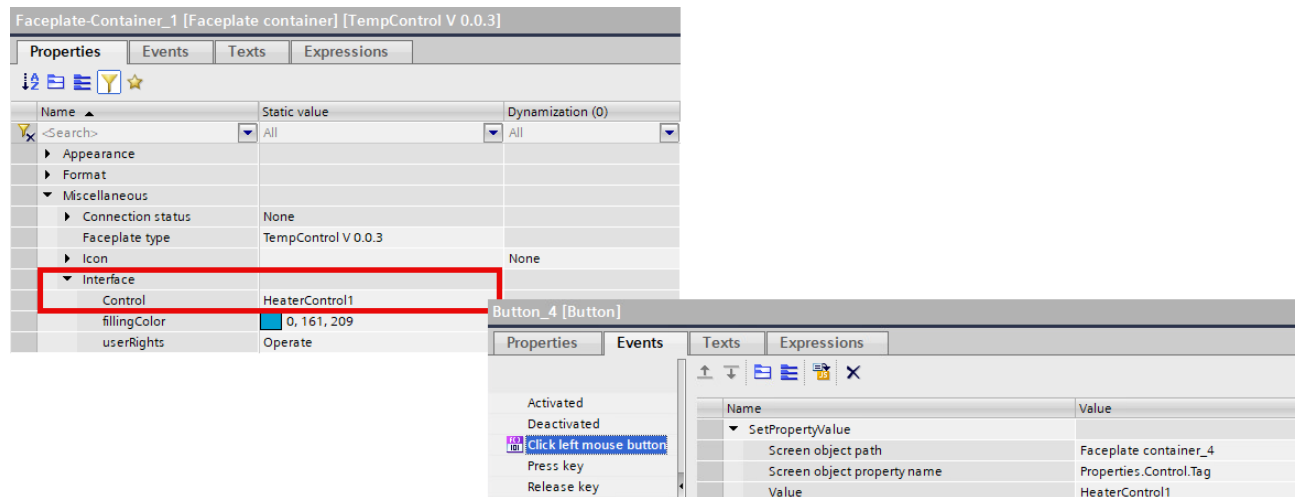


Figure 3-52 Static Faceplate interface connection

Solution 2 – Dynamic interface change using tag prefixes V19

As of V19 it is now possible to use a single Faceplate container to display separate Faceplate instances by using a prefix-based tag dynamization.

Define a tag name consisting of a static part and dynamic parts as tag parameters. The dynamic parts are tag references that are replaced in runtime with the current tag values. The resulting tag name is thus dependent on the values of the tag references.

Especially use this solution, if you want to access complex PLC UDTs that contain much more information than needed inside of the faceplate. Load each UDT as an own HMI tag instance instead of using multiplexing. Thereby only the required tags will be refreshed, not the whole UDT structure. For more information regarding the use of multiplexing see link section [Use Case: PLC UDT Arrays with Multiplexing](#).

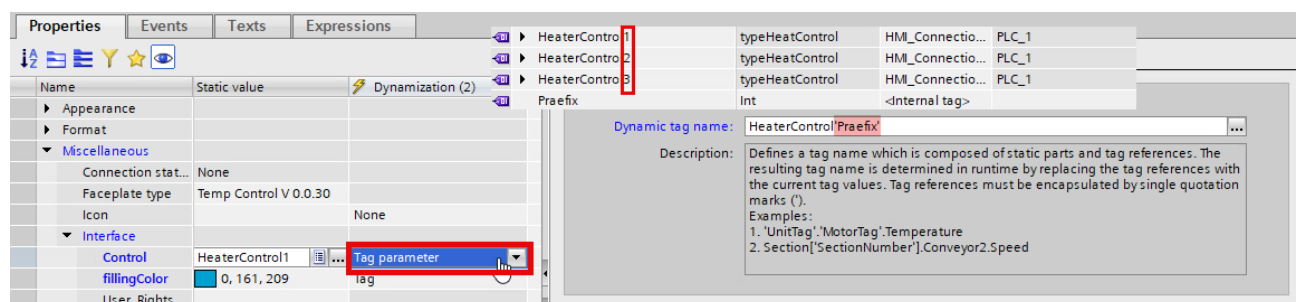


Figure 3-53 Dynamic Faceplate interface

Example:

To access HMI tags "HeaterControl1" to "HeaterControl4" of a user data type "typeControl" via a Faceplate instance, define another tag (e.g. "Praefix") as a dynamic part of a "HeaterControlPraefix" interface property. Depending on the value of the "Praefix" tag in Runtime, the resulting tag name of the HMI tag "HeaterControl1" to "HeaterControl4" results in the Faceplate interface change.

3.3.2. Text Lists in Faceplates

A text list has been configured and is now to be used in a Faceplate. As the Faceplate cannot access this text list directly, the following explains in detail how to proceed in this case.

The integration of text lists in Faceplates offers a variety of possible solutions, each with its own advantages and disadvantages. It is crucial to first clarify the purpose of the text list used in the Faceplate.

Analysis:

It is first important to determine how many texts from the text list are actually to be used in the Faceplate and how many instances of the Faceplate are used in the project.

If the number of text elements to be transferred to the Faceplate is small, there is an optimal solution. In this case, the use of individual variables in the property interface of the Faceplates offers the optimum solution. Further information on the procedure can be found under: [Use Case: Transmitting a Subset of the Text List](#).

If there is a need to utilize a greater quantity of texts from the text list, additional considerations are necessary for the configuration. In this case, a solution with individual variables for each text element of the text site is not an optimal solution, as the engineering effort increases significantly with increasing text elements and Faceplate instances. A suitable solution depends on the specific use case.

It must be checked whether the text lists used will differ from Faceplate instance to Faceplate instance of the same Faceplate. It is therefore examined whether the text lists must be configured **dynamically** or **statically** on the Faceplate.

Use Case: Static use of Text Lists V19

- A standardized text list is to be used for all instances of the Faceplate.
- The text list serves as the basic framework for all Faceplate instances.
- Further adjustments could be made to the text list, so the texts for all Faceplate instances must be adjusted quickly and easily.

[Use Case: Dynamic use of Text Lists](#)

- Different text lists are required for the individual instances of the Faceplate.
- The UseCase of the Faceplate instance is determined by the referenced text list and its referenced interface variables.

3.3.2.1. Use Case: Transmitting a Subset of the Text List

Description

This use case deals with the transfer of individual or a few text elements to a Faceplate from a configured text list. This scenario could arise when designing a text list and only necessitating a few elements from it for each Faceplate instance. A possible scenario for this use case is the control of an actuator using a Faceplate with two buttons.

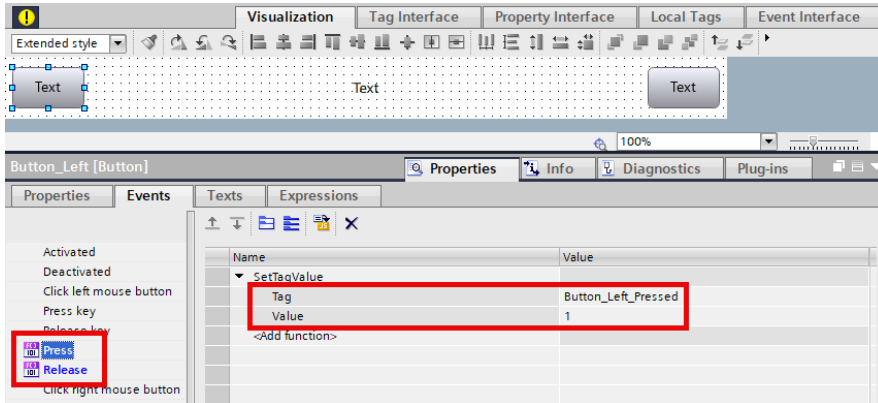


Figure 3-54 Events for Buttons on the Simple Faceplate

The button sets the variable to true, and then resets it to false upon release.

Pressing the two buttons toggles two Boolean variables between "true" and "false". The action triggered by pressing these buttons is determined by the PLC code. This flexibility allows the same Faceplate to be used for multiple actuators, such as cylinders or motors, with the button labels and Faceplate headlines adjusting accordingly.

The list includes all actuators in the system intended to be controlled through the Faceplate.

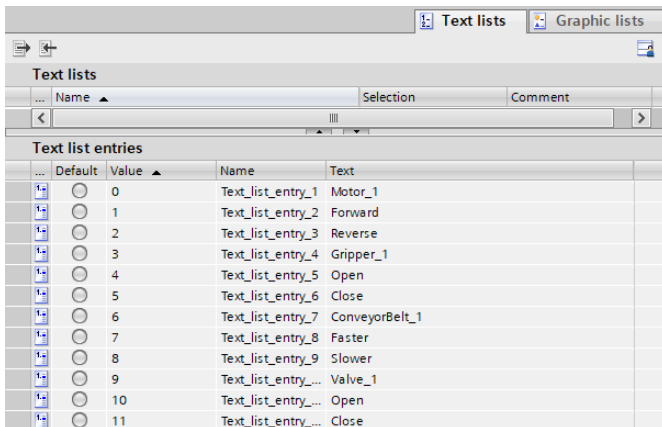


Figure 3-55 Example Text List for Labeling Control Panels for Actuators

In this scenario, a Faceplate instance is to be created to control a **motor**. From the configured text list, you now only need the text elements for the corresponding actuator:

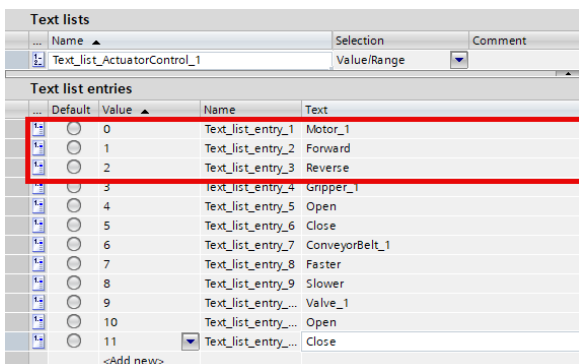


Figure 3-56 Text Elements for the Simple Faceplate

Solution

In this procedure, the individual entries in the text list that are to be transferred to the Faceplate are transferred as multilingual text. Therefore, three multilingual text variables must be configured in the Faceplate interface.

! Visualization Tag Interface Property Interface Local Tags Event Interface		
Name	Data type	Data type explanation
TextHeadline	Multilingual text	Multilingual text
TextButtonLeft	Multilingual text	Multilingual text
TextButtonRight	Multilingual text	Multilingual text
<Add new>		

Figure 3-57 Property Interface of the Simple Faceplate



Faceplate in Popup

Multilingual text objects cannot be referenced at the interface when opening a Faceplate in pop-ups and are not suitable for this use case.

The multilingual text can then be referenced directly at the desired properties of the objects. The use of a script is not necessary here.

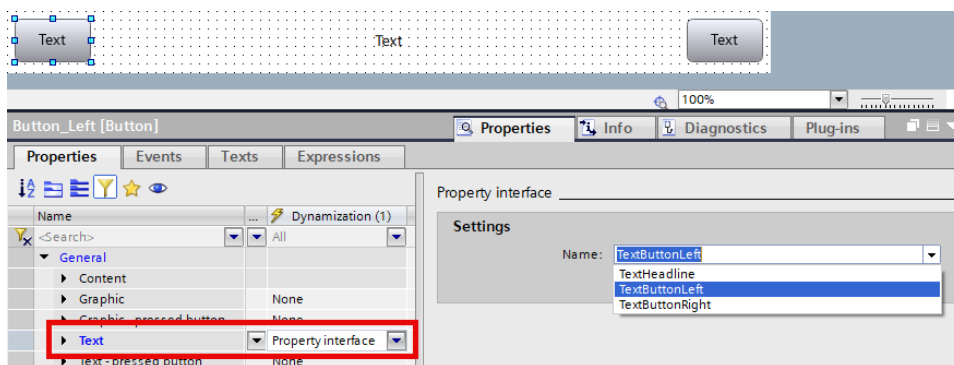


Figure 3-58 Text Property of Buttons on the Simple Faceplate

The multilingual text for the label of the left button is referenced here at the "Text" property.

The desired text elements of the text list can be defined on the Faceplate container of the instance by specifying the index in the interface. These references can differ from one Faceplate instance to another and enable dynamic use of the text elements.

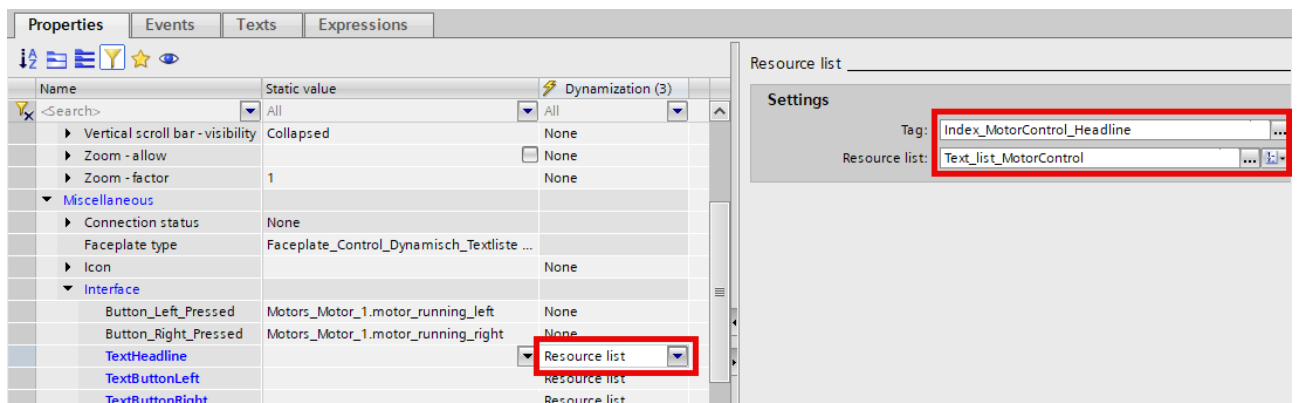


Figure 3-59 Interface of a Faceplate Container on the Simple Faceplate

This is the Faceplate-Container with all references in the Interface.

The variables that are referenced via the interface for controlling the actuators, in this case, the motor, can be configured as follows. They are then interpreted in the PLC code.

Name	Data type	Connection	PLC name	PLC tag	Address	Access mode	Acqui...
Motors_Motor_1	Motor	HMI_Conne...	PLC_1	Motors.Motor_1		<symbolic access>	T1s
motor_running_left	Bool	HMI_Connectio...	PLC_1	Motors.Motor_1...		<symbolic access>	T1s
motor_running_right	Bool	HMI_Connectio...	PLC_1	Motors.Motor_1...		<symbolic access>	T1s
motor_pos	Int	HMI_Connectio...	PLC_1	Motors.Motor_1...		<symbolic access>	T1s

Figure 3-60 PLC-UDT for the Simple Faceplate

Advantages:

- No scripting required.
- Dynamic.

3.3.2.2. Use Case: Static use of Text Lists V19

Description

This Case concerns the static use of text lists that are used in all Faceplate instances of the Faceplate without the need for dynamic adaptation on the Faceplate container.

A typical scenario is the display of the status of system components such as actuators. It is crucial that each Faceplate instance uses the same text list to ensure a consistent display of the actuator status. Any changes or additions to the status/state of an actuator should be automatically applied to all Faceplate instances of this Faceplate.

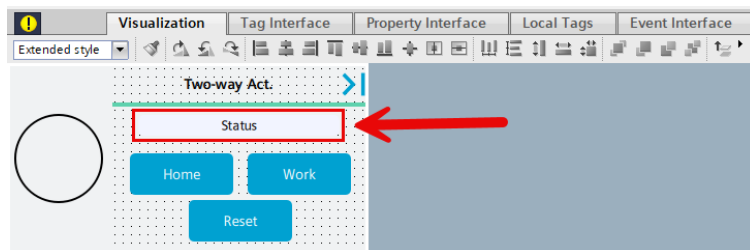


Figure 3-61 Two-Way Actuator Faceplate with a Status Text Box Displaying Actuator Status

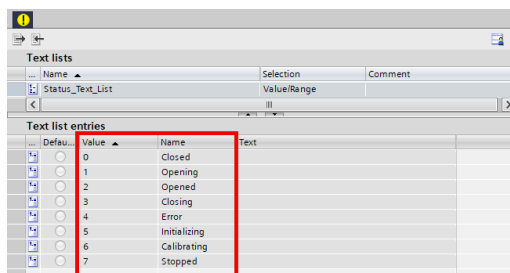


Figure 3-62 Configured Text List Reflecting Possible Actuator States

Solution: Text list-Type V19

For the static use of text lists, we recommend using the text list type, which was introduced in version 19 and can be referenced directly at the property of an object since V19 Update 2. A corresponding text list type for the desired text list must first be created in the library.

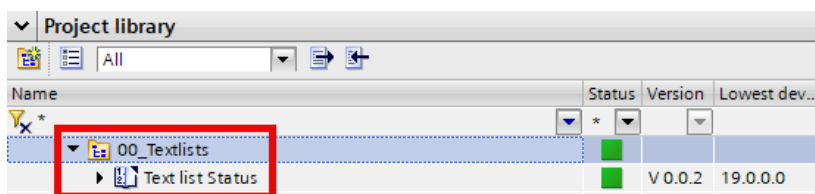


Figure 3-63 Text List Type in the Library

As the Faceplate can communicate directly with the library, the use of the text list type enables direct access to the text list. In this way, the desired text element can be referenced directly to the corresponding properties of the object in the Faceplate.

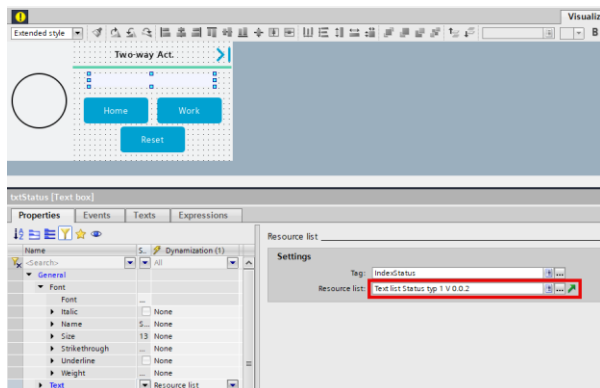


Figure 3-64 Text Property of Status Text Box on Two-Way Actuator Faceplate

Here, the text list configured in the library is referenced to the "Text" property of the text box. The index of the text list can then be set depending on the status of the actuator.

Advantages:

- Engineering: Eliminates the need to specify the text list in the interface, which minimizes both time and potential errors when referencing the Faceplate container.

3.3.2.3. Use Case: Dynamic use of Text Lists

Description

A Faceplate is to be created in which the instances of the Faceplate are each to use different text lists. The text list is therefore adapted depending on the use case of the Faceplate.

The same scenario can be used here as in the [Use Case: Transmitting a Subset of the Text List](#). The difference lies in the number of actuators to be controlled. This Faceplate is intended to create a standardized control element for system parts of a plant. The structure of the GUI remains the same for the system parts, only the logic in the PLC varies, and therefore also the labeling of the buttons.

It's now designed to control four actuators instead of just one.

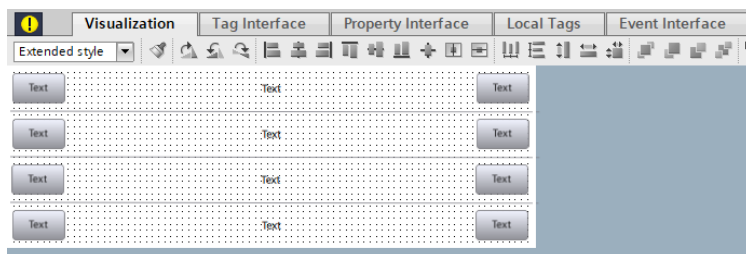


Figure 3-65 Extended Faceplate with Four Control Units

For this scenario, the same text list is used as in the [Use Case: Transmitting a Subset of the Text List](#). Now, however, the entire text content of this text list is used.

Text lists			
Name	Selection	Comment	
Text_list_ActuatorControl_1	Value/Range		

Text list entries			
Default	Value	Name	Text
<input type="radio"/>	0	Text_list_entry_1	Motor_1
<input type="radio"/>	1	Text_list_entry_2	Forward
<input type="radio"/>	2	Text_list_entry_3	Reverse
<input type="radio"/>	3	Text_list_entry_4	Gripper_1
<input type="radio"/>	4	Text_list_entry_5	Open
<input type="radio"/>	5	Text_list_entry_6	Close
<input type="radio"/>	6	Text_list_entry_7	ConveyorBelt_1
<input type="radio"/>	7	Text_list_entry_8	Faster
<input type="radio"/>	8	Text_list_entry_9	Slower
<input type="radio"/>	9	Text_list_entry_...	Valve_1
<input type="radio"/>	10	Text_list_entry_...	Open
<input type="radio"/>	11	Text_list_entry_...	Close

Figure 3-66 Required Text Elements for the Example Text List for Actuator Controls

Solution: Resource List

For this use case, a **resource list** in the **property interface** of the Faceplate can be used to dynamically transfer text lists to a Faceplate. With this solution, the desired text list can be bound to the Faceplate container, depending on the intended use (e.g. attachment part) and the texts can then be transferred to the properties of the objects using a script.

To access a text list in the Faceplate, it must first be specified in the property interface.

Visualization	Tag Interface	Property Interface	Local Tags	Event Interface						
<table border="1"> <thead> <tr> <th>Name</th> <th>Data type</th> <th>Data type explanation</th> </tr> </thead> <tbody> <tr> <td>TextList</td> <td>Resource list</td> <td>Represents the text list</td> </tr> </tbody> </table>					Name	Data type	Data type explanation	TextList	Resource list	Represents the text list
Name	Data type	Data type explanation								
TextList	Resource list	Represents the text list								

Figure 3-67 Property Interface of the Extended Faceplate

This resource list can then be read out and transferred to the properties of the objects, for which the "TextsByValues" method should be used. With this method, you can quickly and easily read out the desired entries and store them in an array. In this solution, it's not possible to directly reference the text list element at the object level, as is the case with the multilingual text.

Properties	Events	Texts	Expressions																																																									
<table border="1"> <tr><td>Name</td><td>S...</td><td>Dynamization (1)</td></tr> <tr><td><Search></td><td>All</td><td></td></tr> <tr><td>Appearance</td><td></td><td></td></tr> <tr><td> Background - alternativ...</td><td>Script</td><td></td></tr> <tr><td> Background - color</td><td>None</td><td></td></tr> <tr><td> Background - fill pattern</td><td>S...</td><td>None</td></tr> <tr><td>Format</td><td></td><td></td></tr> <tr><td> Alignment - horizontal</td><td>L...</td><td>None</td></tr> <tr><td> Alignment - vertical</td><td>T...</td><td>None</td></tr> <tr><td> Background - fill mode</td><td>...</td><td>None</td></tr> <tr><td> Background graphic - stre...</td><td>S...</td><td>None</td></tr> <tr><td>Miscellaneous</td><td></td><td></td></tr> <tr><td> Background graphic</td><td>None</td><td></td></tr> <tr><td> Display name</td><td>None</td><td></td></tr> <tr><td> Interface</td><td></td><td></td></tr> <tr><td> Layers</td><td>...</td><td></td></tr> <tr><td> Name</td><td>F...</td><td></td></tr> <tr><td> Reference color palette</td><td></td><td></td></tr> <tr><td> Suspendable</td><td><input type="checkbox"/></td><td></td></tr> </table>	Name	S...	Dynamization (1)	<Search>	All		Appearance			Background - alternativ...	Script		Background - color	None		Background - fill pattern	S...	None	Format			Alignment - horizontal	L...	None	Alignment - vertical	T...	None	Background - fill mode	...	None	Background graphic - stre...	S...	None	Miscellaneous			Background graphic	None		Display name	None		Interface			Layers	...		Name	F...		Reference color palette			Suspendable	<input type="checkbox"/>				<pre> 1 export function Faceplate_Type_AlternateBackColor_Trigger(item, triggerDataSet) { 2 // Array containing values for retrieving text from the TextList 3 let values = [0,1,2,3,4,5,6,7,8,9,10,11]; 4 5 // Retrieve TextList based on the specified values and language 6 let TextList = HMIRuntime.Resources.TextLists(Faceplate.Properties.TextList).TextsByValues(HMIRuntime.Language, values); 7 8 // Loop through each set of elements (Headline, Left Button, Right Button) 9 for(let i = 1; i < 5; i++){ 10 // Calculate the base index for each set of texts (Headline, Left Button, Right Button) 11 let indexBase = (i - 1) * 3; 12 13 // Set the text of the Headline item based on the TextList 14 Faceplate.Items("Faceplate_Headline_" + i).Text = TextList[indexBase]; 15 16 // Set the text of the Left Button item based on the TextList 17 Faceplate.Items("Button_Left_" + i).Text = TextList[indexBase + 1]; 18 19 // Set the text of the Right Button item based on the TextList 20 Faceplate.Items("Button_Right_" + i).Text = TextList[indexBase + 2]; 21 } 22 } </pre>
Name	S...	Dynamization (1)																																																										
<Search>	All																																																											
Appearance																																																												
Background - alternativ...	Script																																																											
Background - color	None																																																											
Background - fill pattern	S...	None																																																										
Format																																																												
Alignment - horizontal	L...	None																																																										
Alignment - vertical	T...	None																																																										
Background - fill mode	...	None																																																										
Background graphic - stre...	S...	None																																																										
Miscellaneous																																																												
Background graphic	None																																																											
Display name	None																																																											
Interface																																																												
Layers	...																																																											
Name	F...																																																											
Reference color palette																																																												
Suspendable	<input type="checkbox"/>																																																											

Figure 3-68 Loaded Event of the Extended Faceplate

In this case, a property dynamization of the Faceplate type was selected in which the trigger variable "@CurrentLanguage" is used to reload the text list when the language is changed.

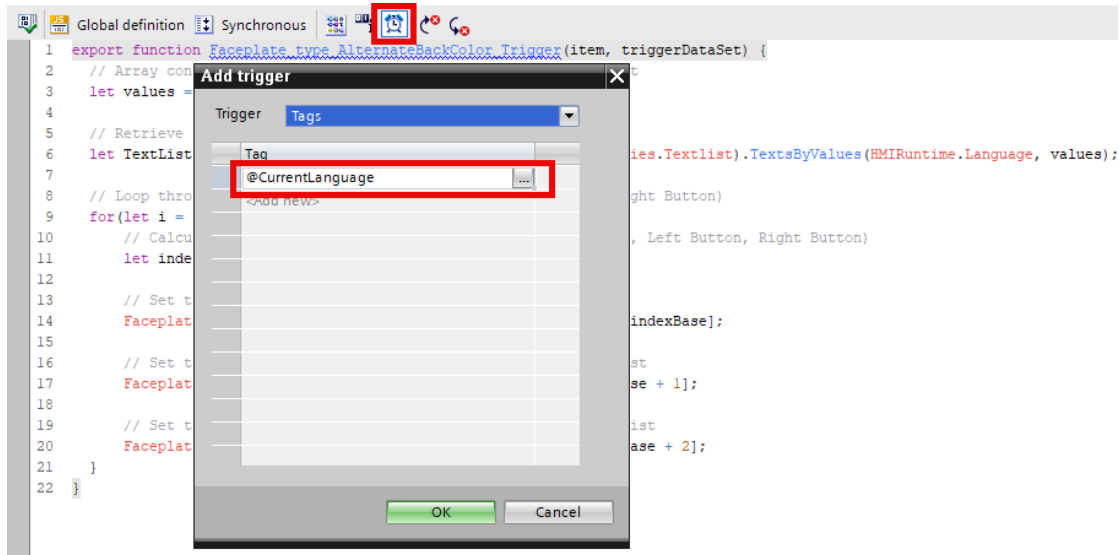


Figure 3-69 Script dynamization for changing the text list

NOTE

Name the objects consciously that they can easily be iterated with a for loop, for example to adjust the text property.

A Faceplate instance of this Faceplate-Type can now be created in a screen and the desired text list can then be transferred to the Faceplate.

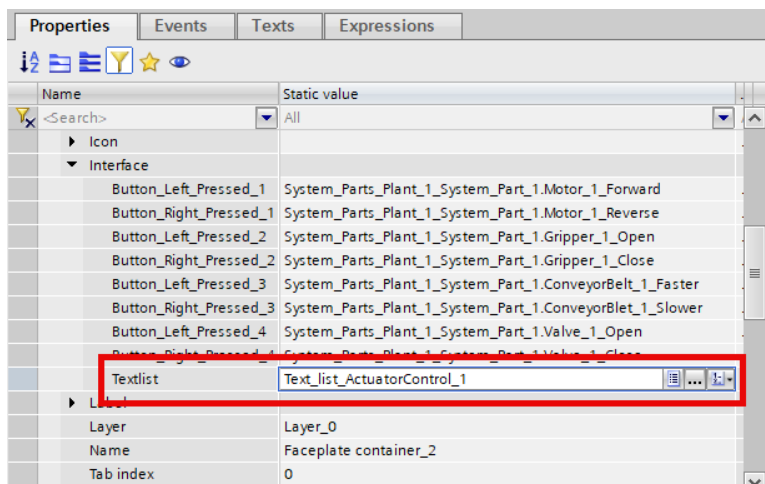


Figure 3-70 Interface of Faceplate Container on the Extended Faceplate

In this Faceplate, firstly, the text list shown above is referenced, which is then read within the Faceplate. Secondly, the required variables to control the actuators are referenced here, which are then processed in the PLC.

Name ▲	Data type	Connection	PLC name
System_Parts_Plant_1_System...	System_Part_Type_1	HM_Connectio...	PLC_1
Motor_1_Pos	Int	HM_Connectio...	PLC_1
Motor_1_Forward	Bool	HM_Connectio...	PLC_1
Motor_1_Reverse	Bool	HM_Connectio...	PLC_1
Gripper_1_Pos	Int	HM_Connectio...	PLC_1
Gripper_1_Open	Bool	HM_Connectio...	PLC_1
Gripper_1_Close	Bool	HM_Connectio...	PLC_1
ConveyorBelt_1_speed	Int	HM_Connectio...	PLC_1
ConveyorBelt_1_Faster	Bool	HM_Connectio...	PLC_1
ConveyorBelt_1_Slower	Bool	HM_Connectio...	PLC_1
Valve_1_Pos	Int	HM_Connectio...	PLC_1
Valve_1_Open	Bool	HM_Connectio...	PLC_1
Valve_1_Close	Bool	HM_Connectio...	PLC_1

Figure 3-71 PLC-UDT for the Extended Faceplate

Advantages:

- Easy to configure
- Best performance when transferring many text elements

3.4. Use of scripts

3.4.1. Script Triggers

Avoid cyclic triggers and use tag triggers instead. If you nevertheless need cyclic scripts and the use case permits it (e.g., when synchronizing data or for data exchange with databases), then configure the scripts in the Task Scheduler. The Task Scheduler runs in a separate process in the background and places less load on your project than if you had configured the scripts in the screen.

But also, be aware when using the setting "Tags-automatic", that all referenced tags of the script are added. This can lead to a lot of triggers and endless loops when writing tags.

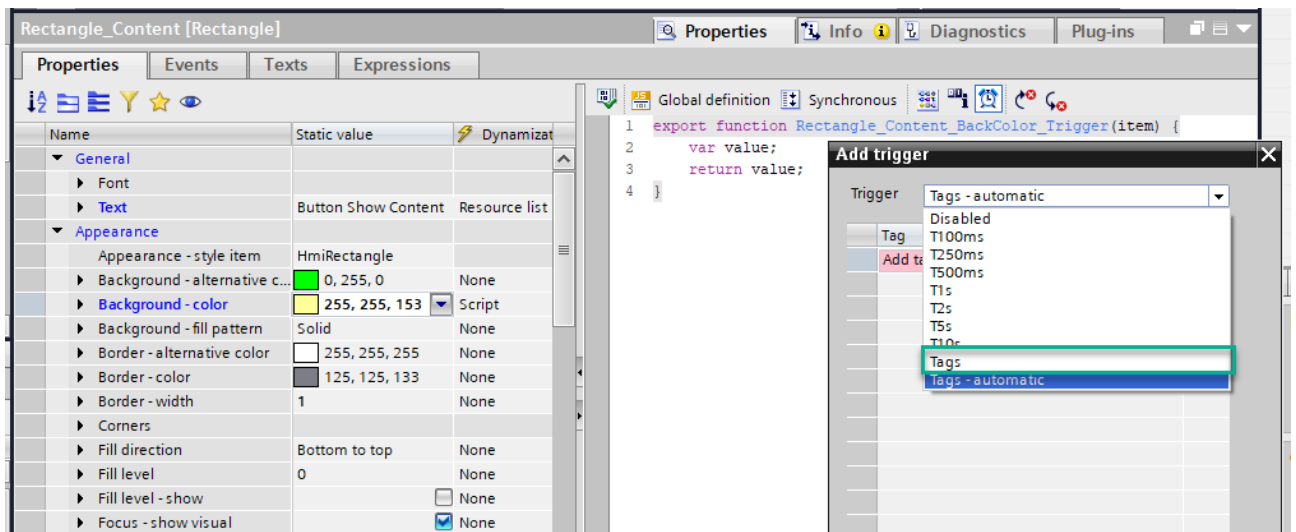


Figure 3-72 Script triggers for script dynamization

If two scripts are triggered by the same tag, combine them into one single script.



Script triggers

Use tag triggers and avoid cyclic triggers



3.4.1.1. Use Case: Scripts using the same Trigger Tag

Definition

When script dynamizations are necessary, some might be triggered with the same tag. So multiple scripts are triggered by the same tag.

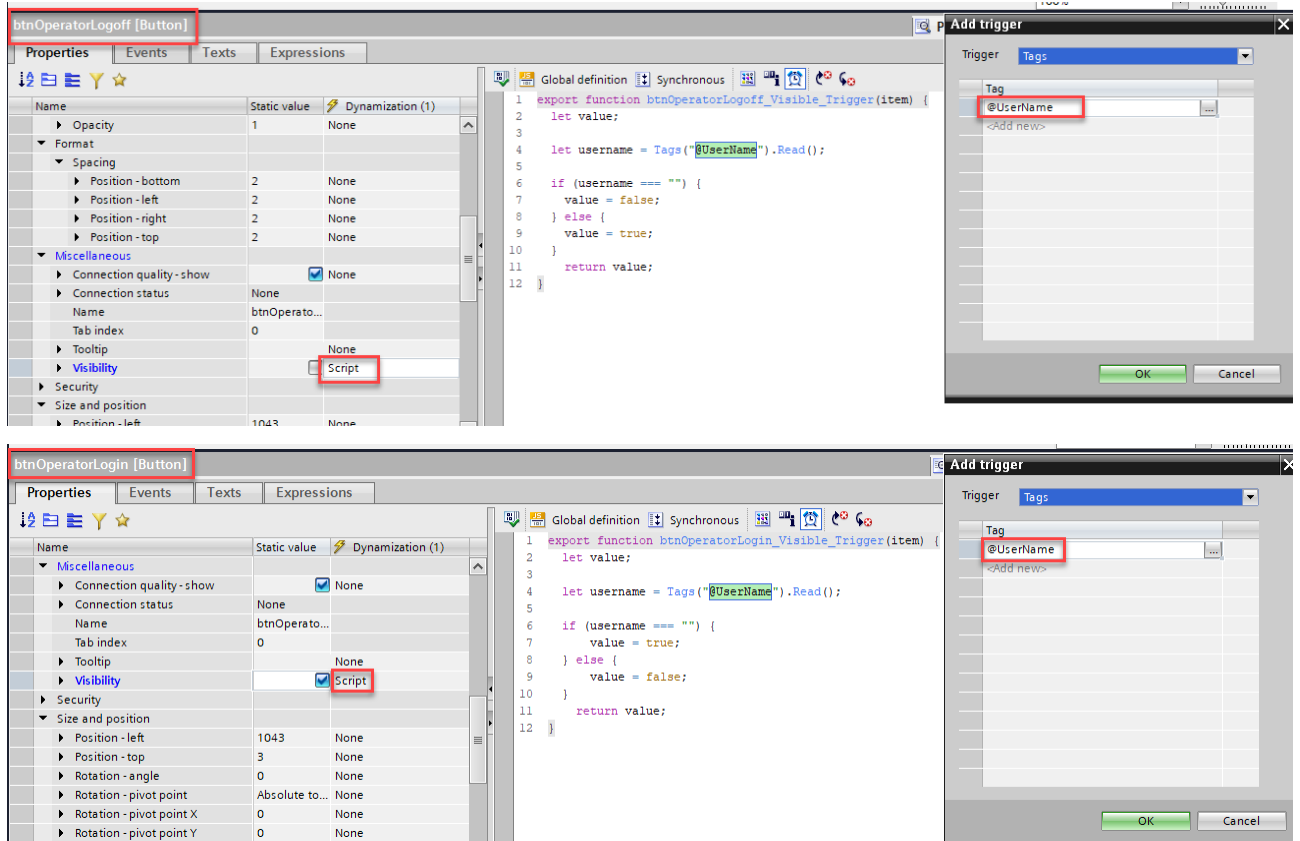


Figure 3-73 Script dynamizations with same trigger tag

Solution 1

Create a subscription in the loaded event of the screen for this tag and manage centrally the property changes with one script. There is already a snippet for the tag subscription available. Especially when the same scripting logic is implemented in the scripts (e.g., if-else) you save a lot of script execution through the one subscription.

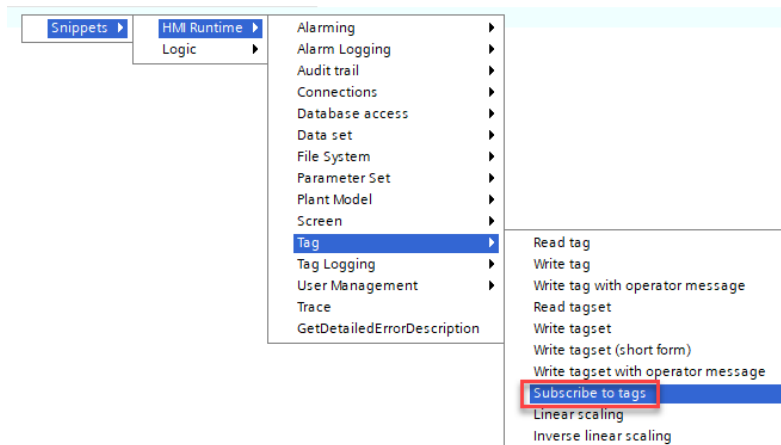


Figure 3-74 Tag subscription snippet

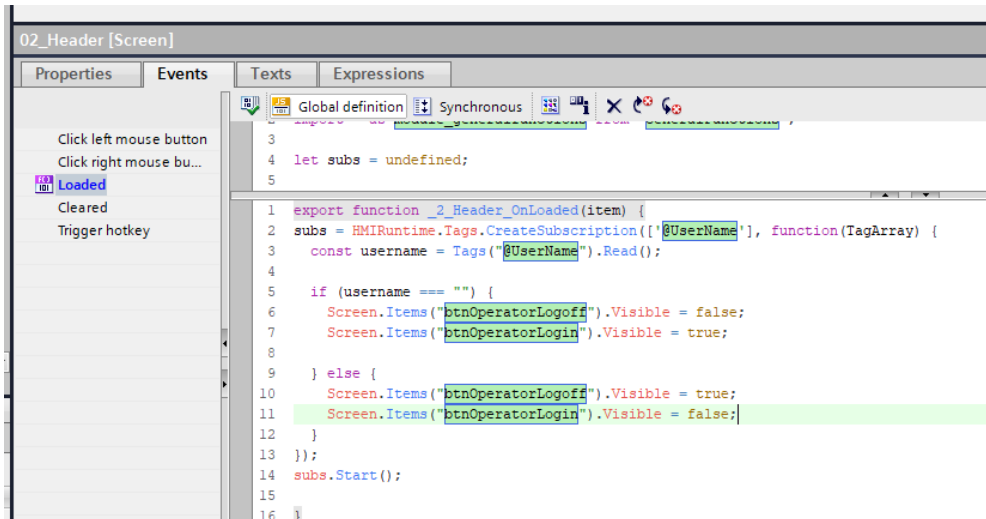


Figure 3-75 Tag subscription in loaded event of the screen

Solution 2 V19 Upd2

Create a script on a property of the screen and manage centrally the property changes with one script. Set the trigger of the script to the tag that should cause the property change. Access the trigger tag value in the script via "triggerDataSet".

triggerDataSet.Value returns the tag value, without reading the tag again, which increases performance.

It is recommended to add this script centrally to a property of the screen and not to a single screen object. The Property "Background – alternative color" has been used here because it is rarely used for other dynamizations.

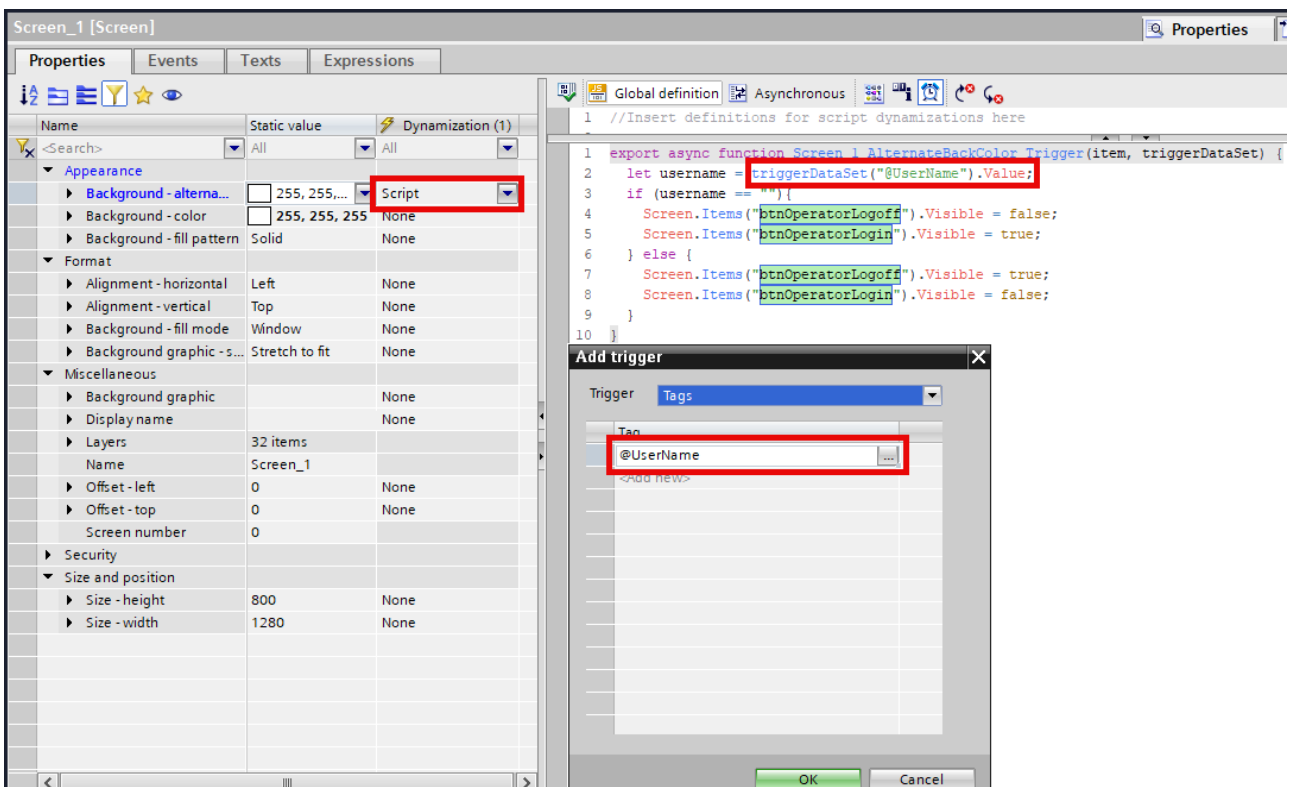


Figure 3-76 "triggerDataSet" for changing the properties of an image



triggerDataSet

V19
Upd2

To access the tag trigger properties, use triggerDataSet.



3.4.1.2. Use Case: Trigger Scripts unrelated to Screen Object Properties

Description

When a script needs to be triggered depending on a tag change, and is not related to a screen object property (no item needs to be adapted), there are a lot of options where to place this script.

In this example, three out of range rectangles are dynamized with a script for the property “Background – alternative color”. When the tag triggers the script, nothing changes for the property, but a function is called. (see [Figure 3-77](#))

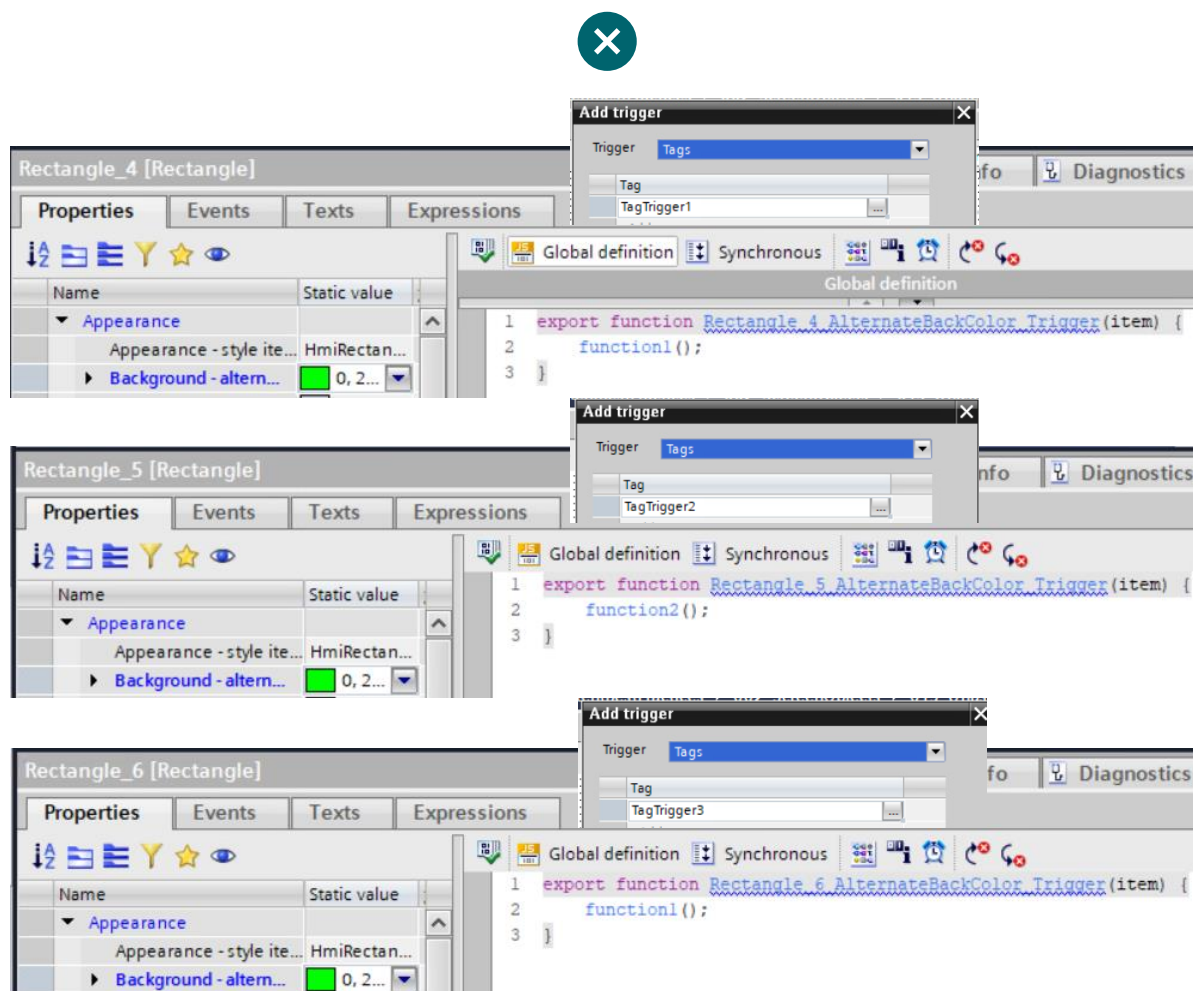


Figure 3-77 Trigger scripts with unused screen item properties

Solution 1

Instead of using out of range objects, a scheduled task can be used for this purpose. This way, non-visible and actually unused objects can be deleted. As this solution only works for code that is not referencing screen objects a second solution is shown for that use case in the following.

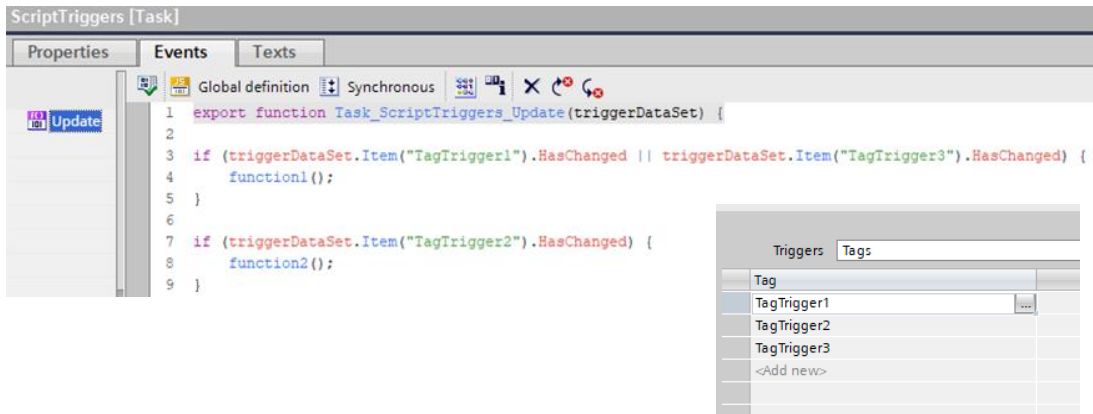


Figure 3-78 Scheduled task for tag triggered code

Solution 2 V19 Upd2

Instead of using out of range objects, a screen property can be used for this purpose. This way, non-visible and actually unused objects can be deleted and the configuration is made centrally for the screen.

A better solution is to use the Screen property "Background – alternative color", add the three Tag triggers to it. To identify which tag is triggered, the system function "triggerDataSet()" can be used. With this information the corresponding function can be called (see [Figure 3-79](#)). This way, the three out of range rectangles could be deleted.

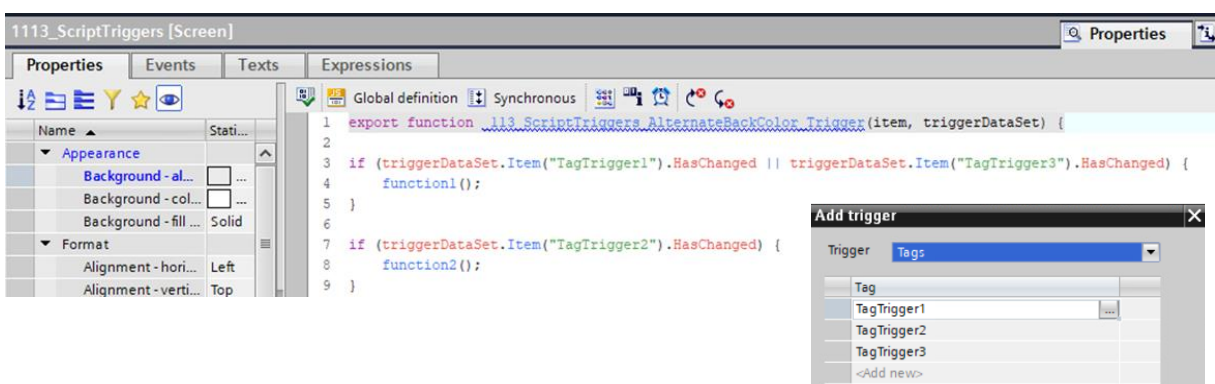


Figure 3-79 Trigger script centrally with screen property

3.4.2. Efficient Code

Efficient coding is all about reducing parts in your scripts that may cause to problems and even about part that do not have an influence on the result of the scripts. There are a lot of points that you should pay attention to. Therefore, just check if your code does not contain elements of the following list:

- Delete functions or import statements in the global definition area that are not called.
- Prefer system functions from the system function list, instead of scripting in the JavaScript editor.
- Delete empty events.

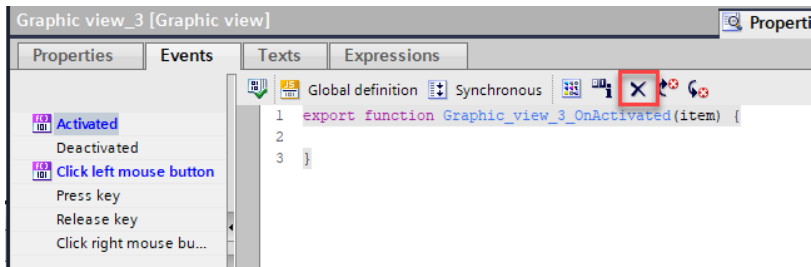


Figure 3-80 Delete empty events

- Avoid unnecessary loop iterations by targeted use of the break statement.

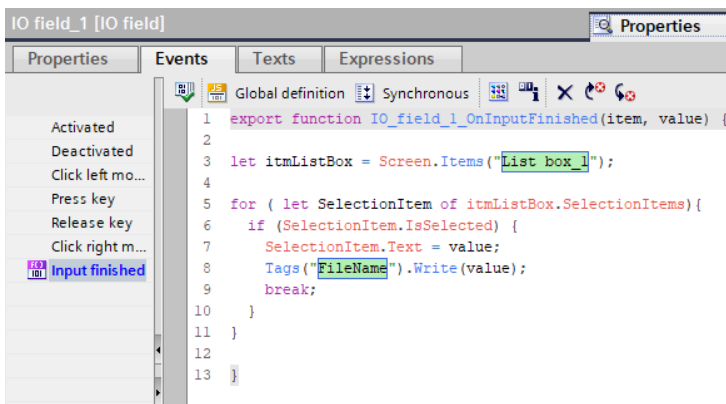




Figure 3-81 Break statement to jump earlier out of for loop

- Delete or comment out constants, variable definitions, debugging traces that are not used anymore.
- Only read tags that are used in the script.
- If two scripts are triggered by the same tag, combine them into one single script.
- Read tags once, save them in a variable and reuse this one if the tag is used multiple time in the script.
- Use a "const" definition when possible (if the value does not need to be changed in the script context) and otherwise define a variable with "let". Do not use "var" anymore, because it is a deprecated java script standard.
- Use a switch-case instead of if-else.





```

1 @export function Button_4_OnTapped(item, x, y, modifiers, trigger) {
2
3   let text = '?????';
4
5   const Temp_Value_GradCels = Tags("KettleTemperature_C").Read();
6   const Temp_Value_GradFahr = Temp_Value_GradCels * 1.8 + 32.0;
7   const Temp_K_Equivalent_0GradCels = 273.15;
8   const Temp_Units = ["°C", "°F", "K"];
9
10  const CountryID = Tags("ParamCountryID").Read();
11  const LCID_deDE = 1031;
12  const LCID_enUS = 1033;
13  const LCID_frFR = 1036;
14  const LCID_frCA = 3084;
15
16  switch (CountryID) {
17    case LCID_deDE:
18      text = Temp_Value_GradCels.toString() + Temp_Units[0];
19      break;
20    case LCID_enUS:
21      text = Temp_Value_GradFahr.toString() + Temp_Units[1];
22      break;
23    case LCID_frFR:
24      text = Temp_Value_GradCels.toString() + Temp_Units[0];
25      break;
26    case LCID_frCA:
27      text = Temp_Value_GradFahr.toString() + Temp_Units[1];
28      break;
29    default:
30      text = (Temp_Value_GradCels + Temp_K_Equivalent_0GradCels).toString() + Temp_Units[2];
31  }
32  Screen.Items("Ext_KettleTemperature").Text = text;
33 }
            
```

```

1 @export function Button_3_OnTapped(item, x, y, modifiers, trigger) {
2
3   var text = '?????';
4
5   var Temp_K_Equivalent_0GradCels = 273.15;
6
7   var LCID_deDE = 1031;
8   var LCID_enUS = 1033;
9   var LCID_frFR = 1036;
10  var LCID_frCA = 3084;
11
12
13  if ((Tags("ParamCountryID").Read() == LCID_deDE) || (Tags("ParamCountryID").Read() == LCID_frFR)) {
14    text = Tags("KettleTemperature_C").Read().toString() + "°C";
15  } else {
16    if ((Tags("ParamCountryID").Read() == LCID_enUS) || (Tags("ParamCountryID").Read() == LCID_frCA)) {
17      text = (Tags("KettleTemperature_C").Read() * 1.8 + 32.0).toString() + "°F";
18    } else {
19      text = (Tags("KettleTemperature_C").Read() + Temp_K_Equivalent_0GradCels).toString() + "K";
20    }
21  }
22
23
24  Screen.Items("Ext_KettleTemperature").Text = text;
25
26 }
            
```

Figure 3-82 Switch case vs. if-else

Use asynchronous scripts if possible. For calls that can take longer e.g. database access, only the async variant is offered anyway. For tag access the sync and the async mode is supported because tag accesses are normally faster. More information regarding the script call can be found in the [Java Script Tips and Tricks](#).

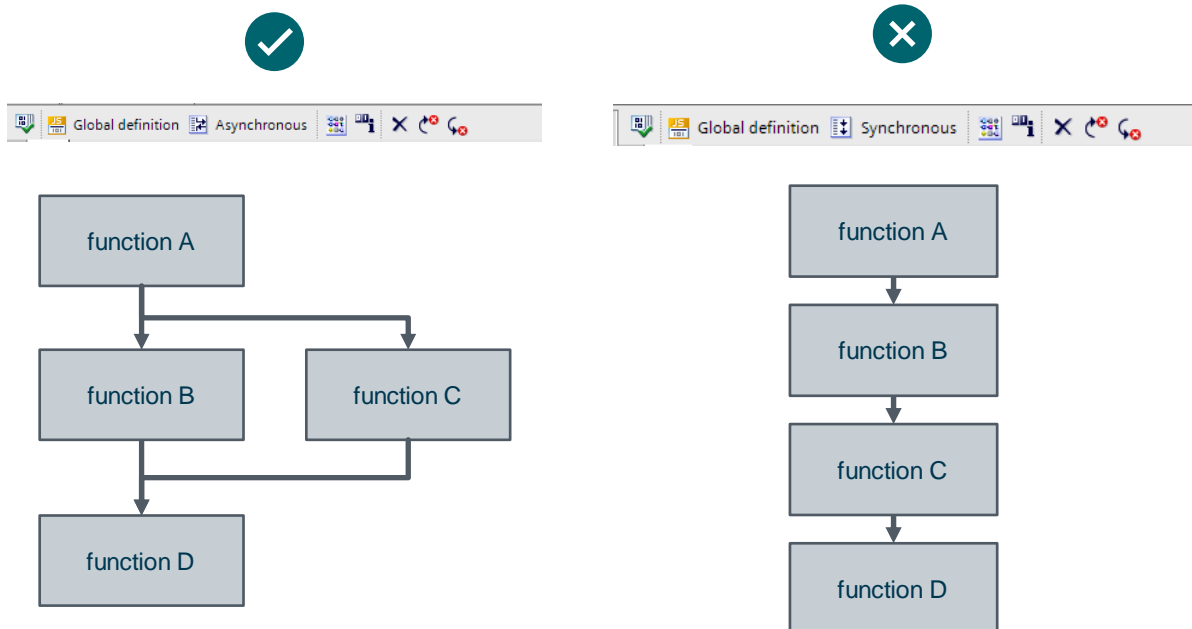



Figure 3-83 Compare of asynchronous and synchronous script call

- Be aware of how often scripts are executed, especially during the screen load process (see [Figure 2-9](#)). The following example is for a script in the on change event of the dynamized process value of a list box. The visibility of these property change events can be switched in the engineering through the eye icon .

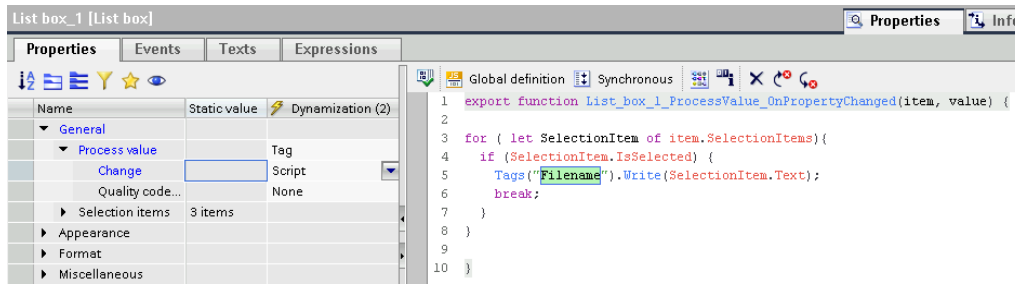


Figure 3-84 On change event from process value of a list box

- Read several text list entries at once via JS with

```
HMIRuntime.Resources.TextList(txtList).TextsByValue(HMIRuntime.Language, value);
```

instead of using a for loop. V19

```
let value = {}
for(let i = 1; i >= count; i++)
{
  values.push( i + currentPage * 6);
}
let texts = HMIRuntime.Resources.TextLists(txtList).TextsByValue(HMIRuntime.Language, values);
```

Figure 3-85 Read test list entries at once

3.4.2.1. Use Case: Write and Read multiple Tags

Description

To write and read single tags a lot of system functions are provided like SetTagValue() and also through Tags("Tagname").Write("value") or Tags("Tagname").Read() the tags can be addressed through scripting.

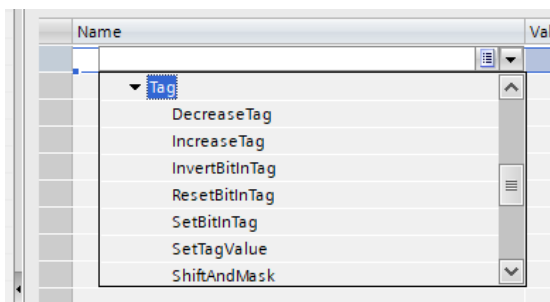


Figure 3-86 Excerpt from the system function in the tags section

Example:



```

Global definition | Synchronous
1 export function Button_2_OnTapped(item, x, y, modifiers, trigger) {
2   const value = Tags("TagSetValue").Read();
3
4   HMIRuntime.Tags.SysFct SetTagValue(MyTag1, value);
5   HMIRuntime.Tags.SysFct SetTagValue(MyTag2, value);
6   HMIRuntime.Tags.SysFct SetTagValue(MyTag3, value);
7   HMIRuntime.Tags.SysFct SetTagValue(MyTag4, value);
8   HMIRuntime.Tags.SysFct SetTagValue(MyTag5, value);
9   HMIRuntime.Tags.SysFct SetTagValue(MyTag6, value);
10  HMIRuntime.Tags.SysFct SetTagValue(MyTag7, value);
11  HMIRuntime.Tags.SysFct SetTagValue(MyTag8, value);
12  HMIRuntime.Tags.SysFct SetTagValue(MyTag9, value);
13  HMIRuntime.Tags.SysFct SetTagValue(MyTag10, value);
14
15
16  Screen.Items("TextSetTag").Text = "";
17
18  for (let i = 1; i < 11; i++){
19
20    Screen.Items("TextSetTag").Text += " Value of element " + i + "=" + Tags("MyTag" + i).Read() + "\r\n";
21  }
22 }

```

Figure 3-87 Multiple calls of SetTagValue

But there could be also a tag set created, that covers multiple tags in a container.

Solution

Create a tag set when writing or reading multiple tags and use its Write and Read method to only send one collective order to the PLC.

The following screenshot shows the customization of the script with TagSet functionality.



```

Global definition | Asynchronous
1 export async function Text_box_21_OnTapped(item, x, y, modifiers, trigger) {
2   const value = Tags("TagSetValue").Read();
3
4   //TagSet creation
5   let ts = Tags.CreateTagSet(["MyTag1"], ["MyTag2"], ["MyTag3"],
6   ["MyTag4"], ["MyTag5"], ["MyTag6"], ["MyTag7"],
7   ["MyTag8"], ["MyTag9"], ["MyTag10"]]);
8
9   //Read the TagSet
10  ts.Read();
11
12  //Empty text field for the new values
13  Screen.Items("TextSetTag").Text = "";
14
15
16  for (let i = 0; i < ts.Count; i++){
17    ts[i].Value = value;
18    //Show all values at the text field
19    Screen.Items("TextSetTag").Text += " Value of element " + i + "=" + ts[i].Value + "\r\n";
20  }
21
22  //Write new values to each tag at the TagSet
23  ts.Write();
24
25 }

```

Figure 3-88 Reading multiple tags with one TagSet



Read and Write Tags

Use a TagSet to read and write multiple tags instead of single calls of SetTagValue. Especially for synchronous reading and writing of several control variables.



For more information regarding scripting read the [SIMATIC WinCC Unified – Tips and Tricks for Scripting \(JavaScript\)](#).

3.5. Others

- Reduce empty textlist entries
- Be aware of the system limits

Screens

	Unified Comfort 7-12"	Unified Comfort 15-22"
Maximum size in the engineering system	20,000 * 20,000 pixels	
Maximum size in runtime	20,000 * 20,000 pixels	
Number of screens	1200	
Number of lower-level screen windows	10	
Number of objects per screen	800	1200
Number of objects from the "Controls" area per screen	40	80
Number of tags per screen	600	800

Figure 3-89 Excerpt from the system limits defined in [19](#)

- Only relevant for PC-RT: Script Debugger should be disabled in productive use of the runtime project

3.5.1. Acquisition Cycle

When a PLC tag is registered, it is configured through the HMI how often the value gets updated from the PLC to the HMI. This time is defined as the acquisition cycle. If the PLC detects a value-change after one cycle time, the value is communicated from the PLC to the HMI. If the value remains the same, there is no communication between HMI and PLC.

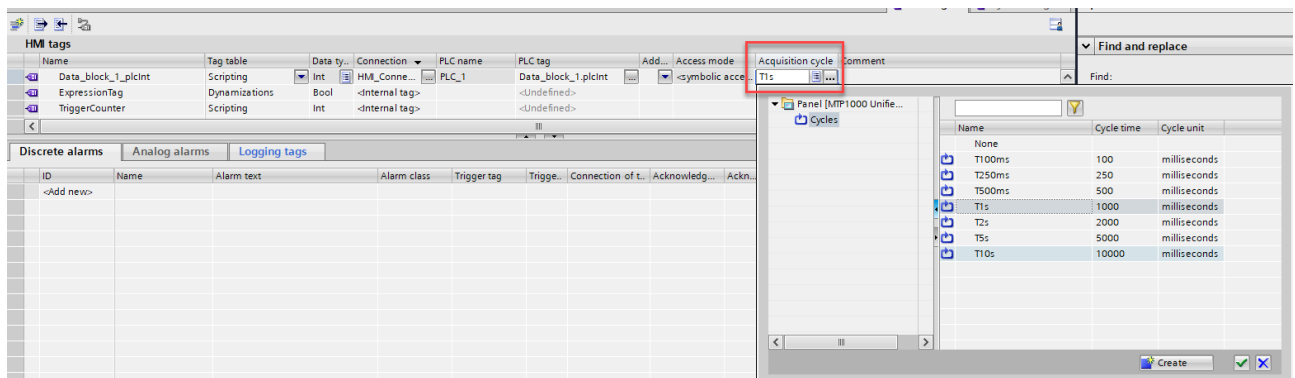


Figure 3-90 Acquisition cycle for HMI tags with PLC connection

In general it is advised to configure a higher acquisition cycle as this reduces communication load.



Acquisition Cycle

Prefer in general a high / long acquisition cycle for PLC tags and apply the inching system functions to prevent mistakes in overwriting values.



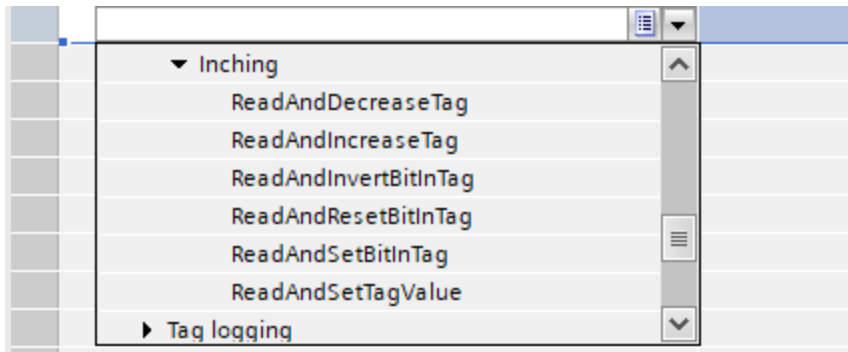


Figure 3-91 Inching system functions for PLC tags

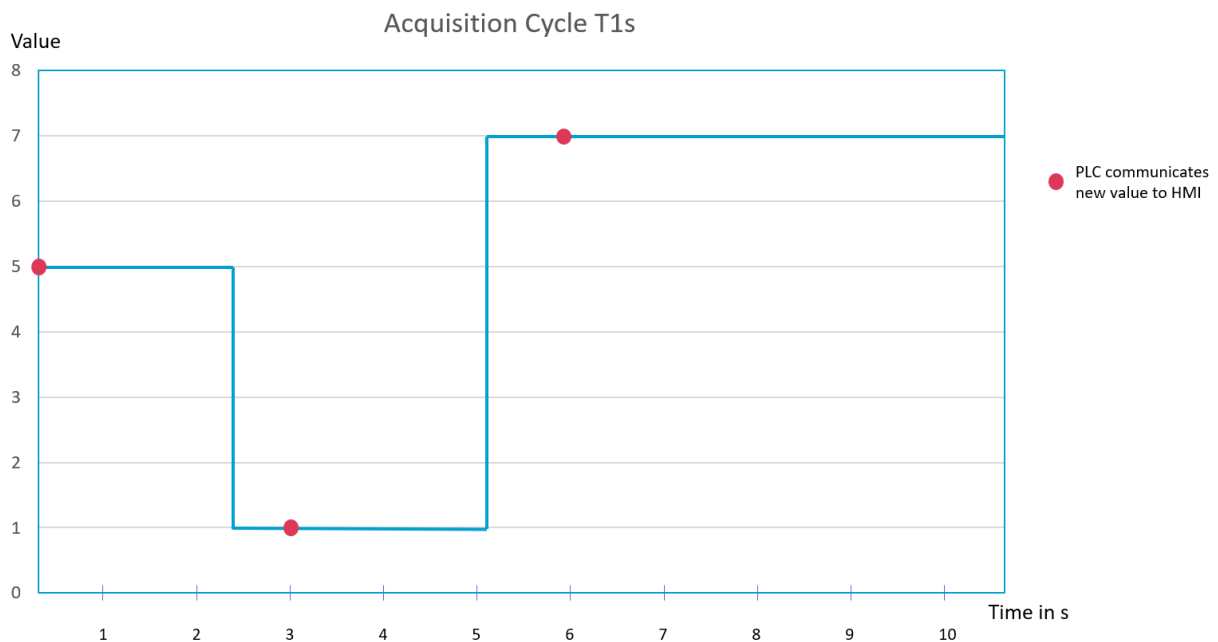


Figure 3-92 HMI-PLC Communication for Tag with Acquisition Cycle of T1s

Please keep in mind that PLC tags with a higher acquisition cycle are updated less often and therefore a change in value is only visible in Runtime after the current cycle time is over.

3.5.1.1 Use Case: Writing a PLC Tag in Screen Loaded Event

Definition

Screens inside of screen windows can also be set by number. If depending on a tag, a different screen should be displayed inside a screen window, it can be necessary to read and/or write PLC tags inside of the screen loaded event.

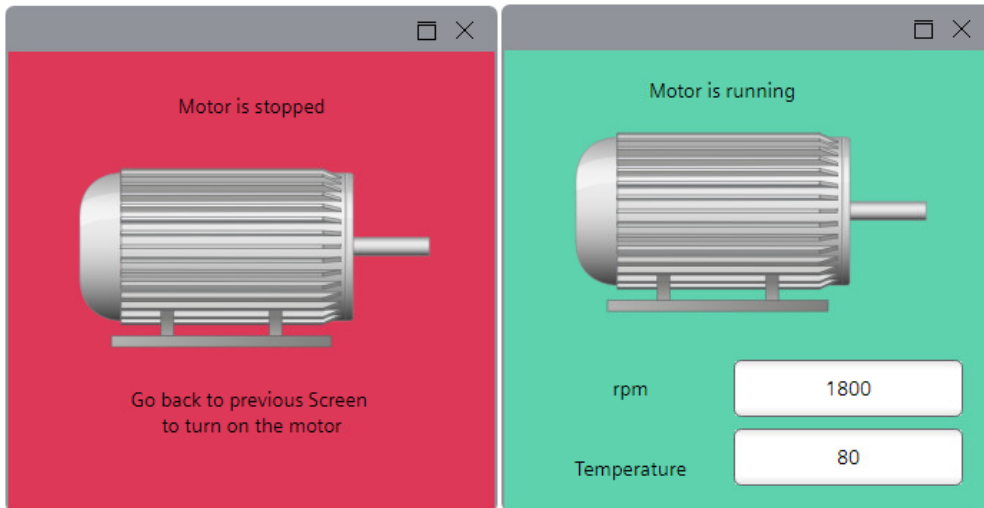


Figure 3-93 Different screens that should be displayed

Solution

In the following screen loading event, a PLC tag is written which is used for the dynamization of a screen.

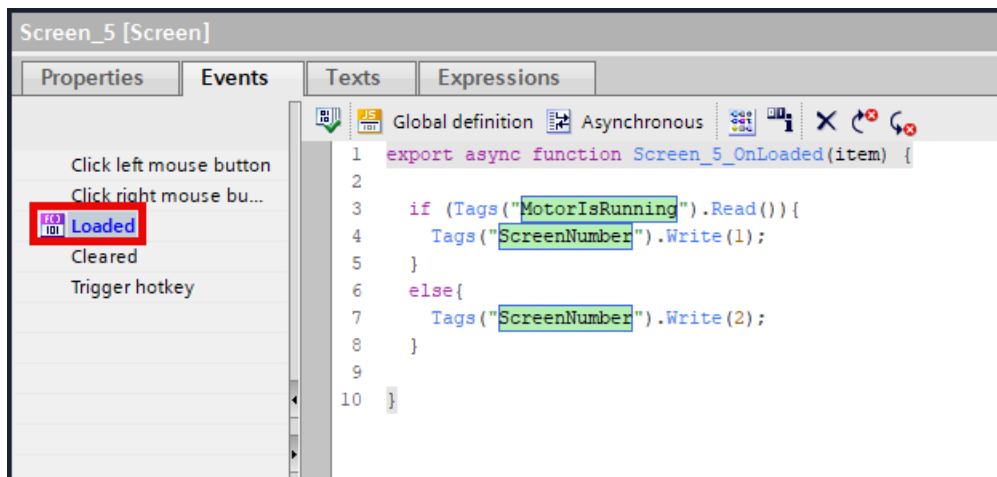


Figure 3-94 Writing a PLC tag in the loading event of an image

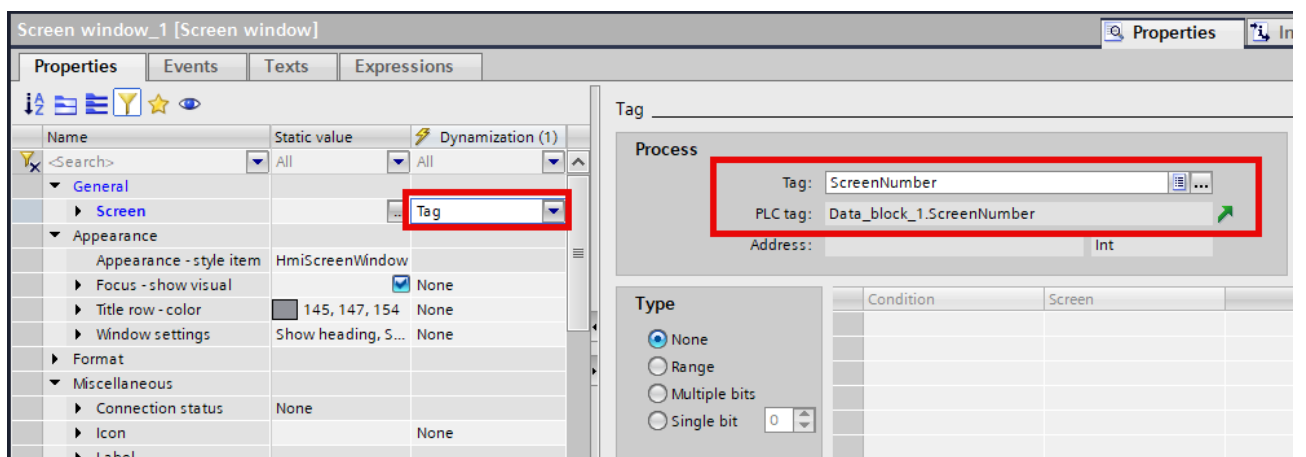


Figure 3-95 Dynamization of screen via Screen Number

As the written tag "ScreenNumber" is a PLC tag, the new value will be visible after one complete acquisition cycle time. Therefore configure a small acquisition cycle for PLC tags that are written in the screen loaded event.

**Acquisition Cycle Loaded Event**

Configure in small / short acquisition cycle for PLC tags that need to be written in a screen loaded event.



3.5.2. Use Case: PLC UDT Arrays with Multiplexing

Definition

To describe objects (e.g., a motor) with tag values, UDTs are a good way to structure the data that belongs to the object.

Data_block_1								
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint
1	▼ Static			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	▼ Motor1	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	Name	WString	WSTRING#"	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	Speed	Real	0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Acc	Real	0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	▼ Motor2	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	Name	WString	WSTRING#"	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
8	Speed	Real	0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
9	Acc	Real	0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Figure 3-96 Small motor PLC UDT

If there are a lot of instances of this object (e.g., 20), but not all data of all instances need to be available at once in runtime, there is the option of multiplexing. Thereby only a subset from this array (e.g., 5 motors) is addressed and when the data of another motor is required only the reference to the instance needs to be changed by selecting another index of the array.

Data_block_1									
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Supe
10	▼ Motors	Array[0..20] ...		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
11	▼ Motors[0]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
12	Name	WString	WSTRING#"	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
13	Speed	Real	0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
14	Acc	Real	0.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
15	▶ Motors[1]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
16	▶ Motors[2]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
17	▶ Motors[3]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
18	▶ Motors[4]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
19	▶ Motors[5]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
20	▶ Motors[6]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
21	▶ Motors[7]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
22	▶ Motors[8]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
23	▶ Motors[9]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
24	▶ Motors[10]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
25	▶ Motors[11]	"UDTMotor"		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Figure 3-97 Array of 20 motor PLC UDTs

Sometimes the UDTs are much more complex and store a lot of data that might be necessary for the PLC but not for the HMI.

Name	Tag table	Data type	Connection	PLC name	PLC tag
servoToControl3	PowertagsIndex_Servo	LFB_typeServoInt...	HMI_Conne...		<Multiplex tag>
commands	PowertagsIndex_Servo	LBC_typeInterfaceCo...	HMI_Connectio...		<Multiplex tag>
refreshConfiguration	PowertagsIndex_Servo	Bool	HMI_Connectio...		<Multiplex tag>
editConfiguration	PowertagsIndex_Servo	Bool	HMI_Connectio...		<Multiplex tag>
saveConfiguration	PowertagsIndex_Servo	Bool	HMI_Connectio...		<Multiplex tag>
acknowledge	PowertagsIndex_Servo	Bool	HMI_Connectio...		<Multiplex tag>
servoCommands	PowertagsIndex_Servo	LFB_typeServoCom...	HMI_Connectio...		<Multiplex tag>
configuration	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>
monitoring	PowertagsIndex_Servo	_typeServoPro...	HMI_Connectio...		<Multiplex tag>
diagnostics	PowertagsIndex_Servo	LAxisCtrl_typeDiagno...	HMI_Connectio...		<Multiplex tag>
servoToControl1	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>
servoToControl2	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>
servoToControl4	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>
servoToControl5	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>
servoToControl6	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>
servoToControl8	PowertagsIndex_Servo	LFB_typeServoInterf...	HMI_Connectio...		<Multiplex tag>

Figure 3-98 Complex PLC UDT with more data than required in HMI

Solution

If only a small amount of the data from the UDT instances is required in the visualization, do not use multiplexing. Even if only one datapoint from the UDT is linked to an HMI property, when changing the concrete instance of the UDT, the whole structure is loaded from PLC and not only the part that is needed.

For data stored in complex PLC UDTs load each UDT as an own instance and do not use arrays with multiplexing. Every time an index is changed the whole structure at the PLC will be refreshed even if it is not in use.



Multiplexing of PLC UDTs

Only use multiplexing for simple PLC UDTs or when all data from the UDT instances is also required in the visualization.



4. Analysis of an existing Project

When there is already an existing project, regardless of whether problems have already occurred during runtime or not, it is always beneficial to carry out an analysis whether there are opportunities for improvement in the engineering of a project. In this section a guideline is provided that shows you all possibilities to further analyze your WinCC Unified project and therefore also gives you an overview about a possible step-by-step analysis procedure.

If there is a known problem in runtime (screens are loaded slowly, some functionality is not working as intended), you can directly jump in the object or script analysis. Otherwise, you can test the project on the target device in runtime and check if everything works as expected.

For an analysis it is also important to be familiar with the general runtime workflow (e.g., the order of running scripts) and what is running maybe in the background (e.g., cyclic scripts for timers), to be able to go to the responsible location (script, configuration, task) in the engineering when a problem is found. If there is to less knowledge about the running scripts and order of processing during runtime, the Google Chrome script debugger gives a good opportunity to automatically navigate through all the executed scripts during screen load, event execution through a button click or a specific screen change.

Another method is to put traces into scripts and check through the RTIL Trace Viewer when they are triggered.

Once you are familiar with the general script execution and processes in your project, you have a good knowledge base for the further analysis.

The following scheme shows a possible procedure in which order and how to choose the tools for analyzing the project, when to look at the runtime or when to focus more on the scripting part of the project. The analysis also consists of two parts, the object and the script analysis. The order, which one is made first, can be changed depending on the project.

All procedures that are mentioned in the flow chart are described in detail in the following chapters.

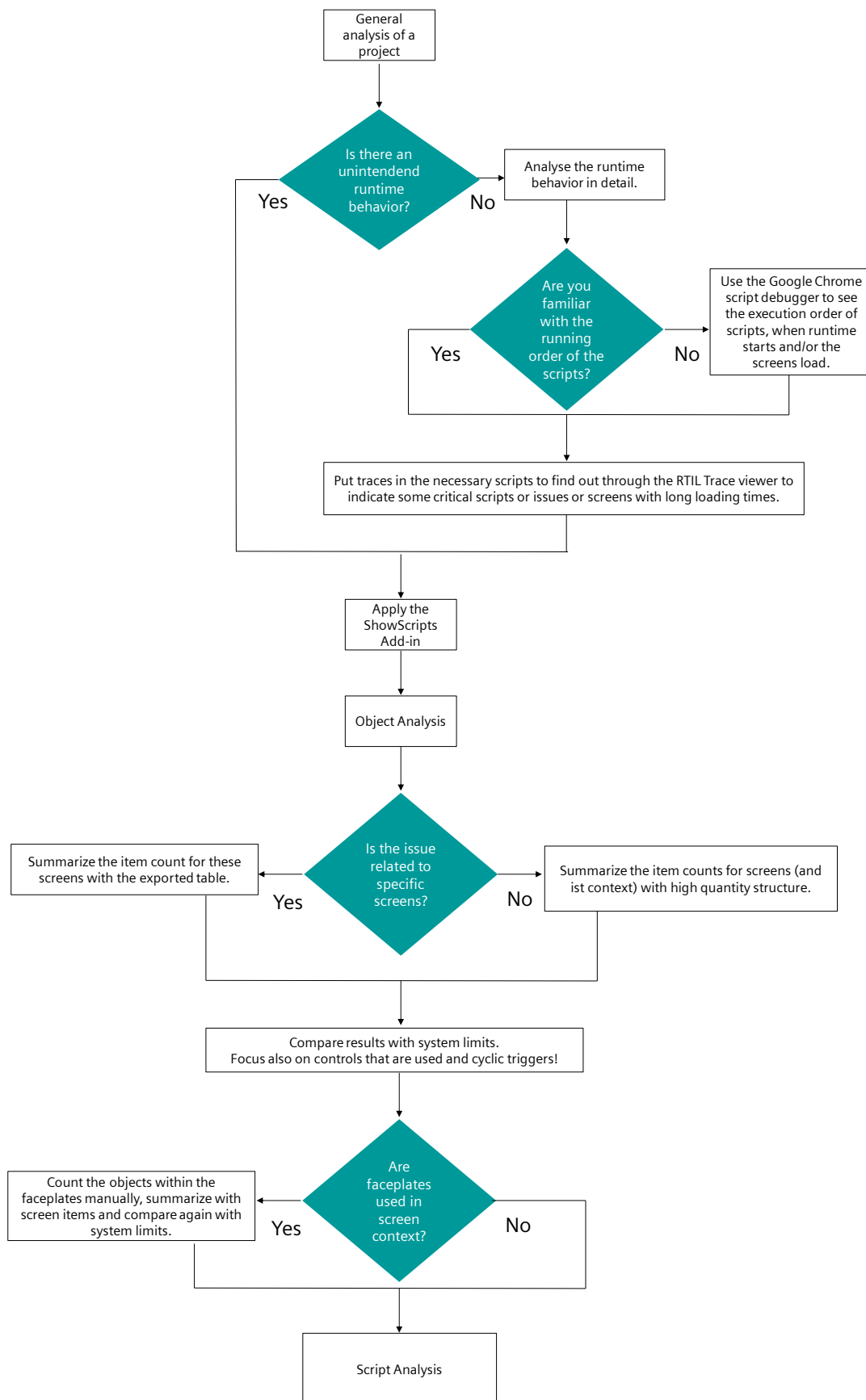


Figure 4-1 Project analysis startup flow chart

The following flowchart shows the different possibilities for script analysis.

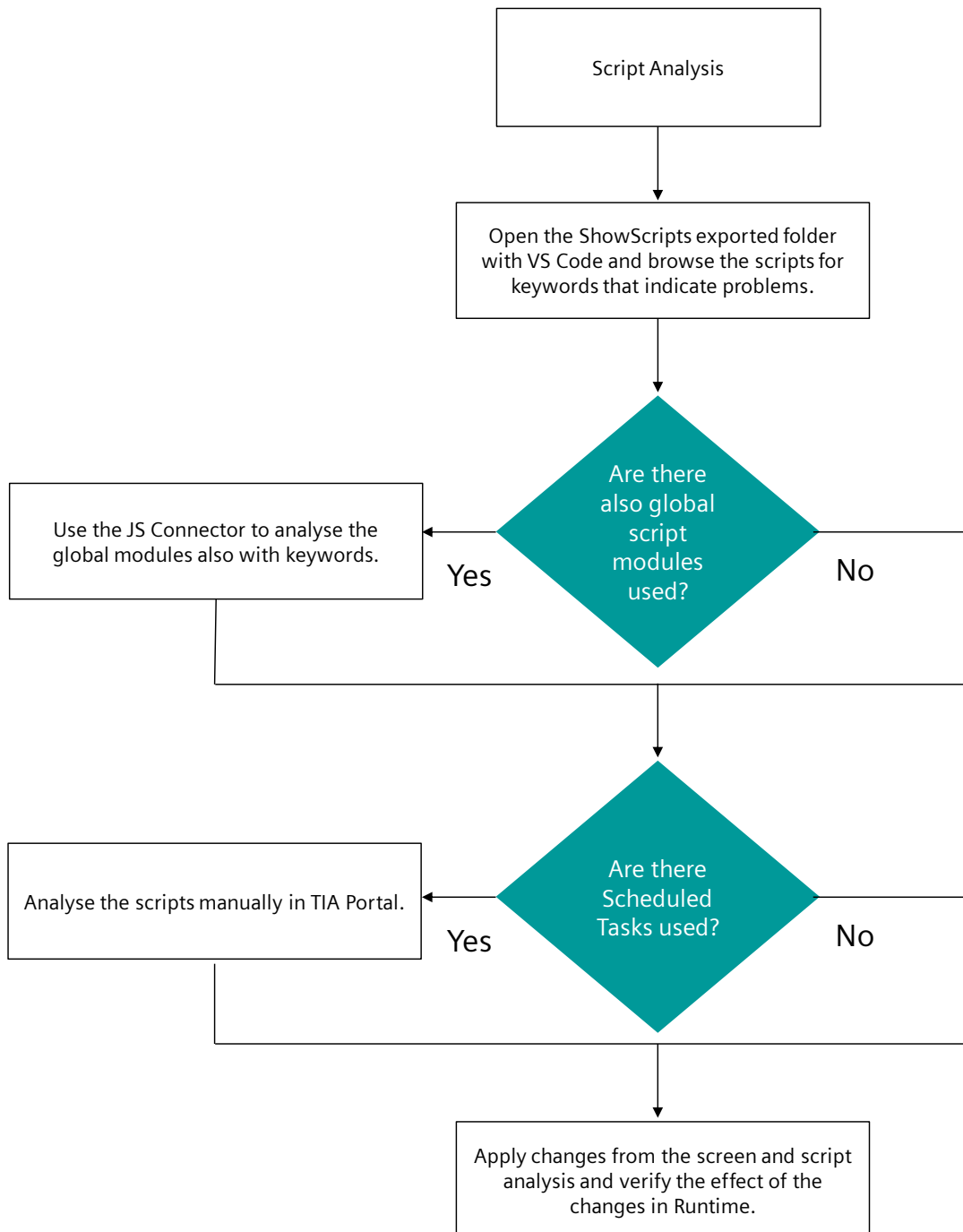


Figure 4-2 Project analysis script analysis flow chart

4.1. ShowScripts Add-In

The ShowScripts Add-In is a TIA Portal Add-In which provides an overview about HMI devices in TIA Portal projects and their quantity structures regarding screens, dynamizations, scripts and events.

The following chapters describe how to retrieve the Add-In and analyze the export data of an HMI device regarding efficient engineering.

4.1.1. Download and Installation

The Add-In is an open-source tool and provided free of charge via [Github](#). Navigate to the releases to download the latest version.

The screenshot displays the GitHub repository for 'tia-portal-applications / TIA-Add-In-ShowScripts'. The repository overview shows a list of files and folders, including 'OpennessConsoleApplication', 'ShowScripts', '.gitignore', 'EXPORT.yml', 'LICENSE', 'README.md', and 'ShowScripts.sln'. The 'Releases' section is highlighted, showing the latest release 'ShowScripts v18.17.0' with 4 releases in total. The release page for 'ShowScripts v18.17.0' is shown below, featuring a description, a 'Assets' section with 3 items, and a 'Source code (zip)' link. The 'Assets' section lists 'ShowScripts_V18.17.zip' (42 KB, last month), 'Source code (zip)' (Oct 2), and 'Source code (tar.gz)' (Oct 2).

Figure 4-3 ShowScripts Add-In on GitHub

NOTE

Please be aware to download the most current version of the Add-In to take benefit from the latest bugfixes.

After the download of the .zip folder of the ShowScripts Add-In, it needs to be installed in TIA Portal. Please refer to the respective [Use Case: TIA Add-Ins](#) in the Siemens Industry Online Support for a description how Add-Ins can be installed and used.

The Add-In is ready for use as soon as you can see the green status in the Add-Ins task card in the top right corner.

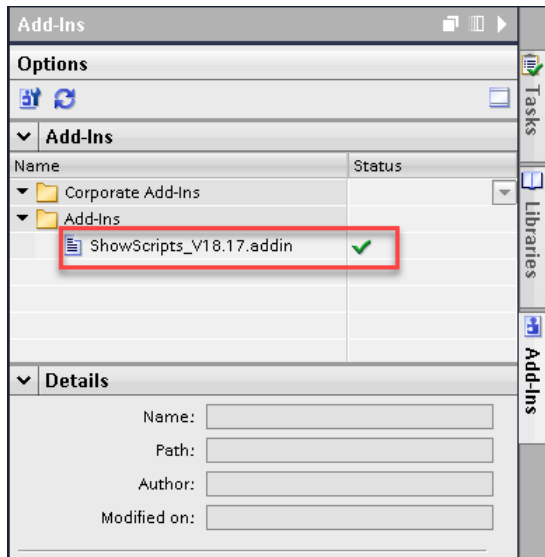


Figure 4-4 ShowScripts Add-In installed in TIA Portal

4.1.2. Usage of the ShowScripts Add-In

Function scope

The function scope of the ShowScripts Add-In is the export of relevant quantity structures and scripts of a WinCC Unified based device which means:

- All scripts which are used on the screen items within every screen as a js file
- A tabular overview (csv file) of all screens and its respective screen item amount

Starting the analysis with ShowScripts

1. For getting started with the ShowScripts Add-In open the to be checked project in TIA Portal and follow the steps below.
2. Right-click on the Unified device you want to analyze with the Add-In.
3. If the Add-In is ready to use (see chapter [4.1.1](#)) the option "ShowScriptCode" will be available at the right-click menu. There are two "Export" and one "Import" option available.
 - **Export all scripts of HMI:** Option for the very first export of scripts and screen overview
 - **Export all scripts of HMI overwrite:** Option for subsequent exports to overwrite existing export files after a previous export failure
 - **Import all scripts to HMI:** Option for the import of adapted scripts back to the project

For the initial analysis of a device in your project select the option "Export all scripts of HMI" to start the project analysis.

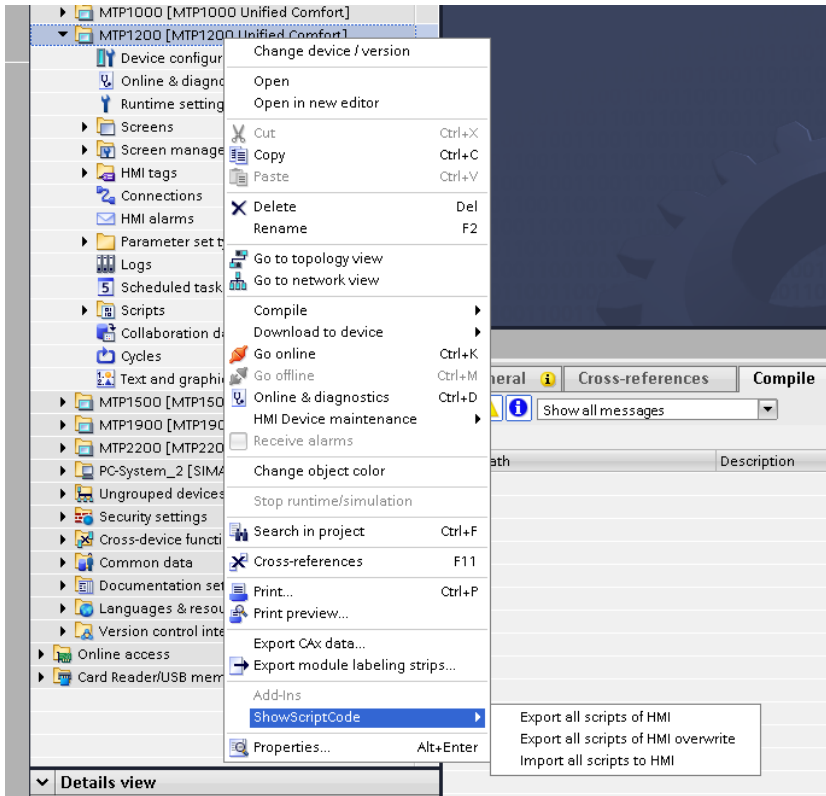


Figure 4-5 Initial export via ShowScripts Add-In

As next step the TIA Portal will ask you for confirmation that the Add-In will access your project's data via Openness which you need to confirm with "Yes to all".

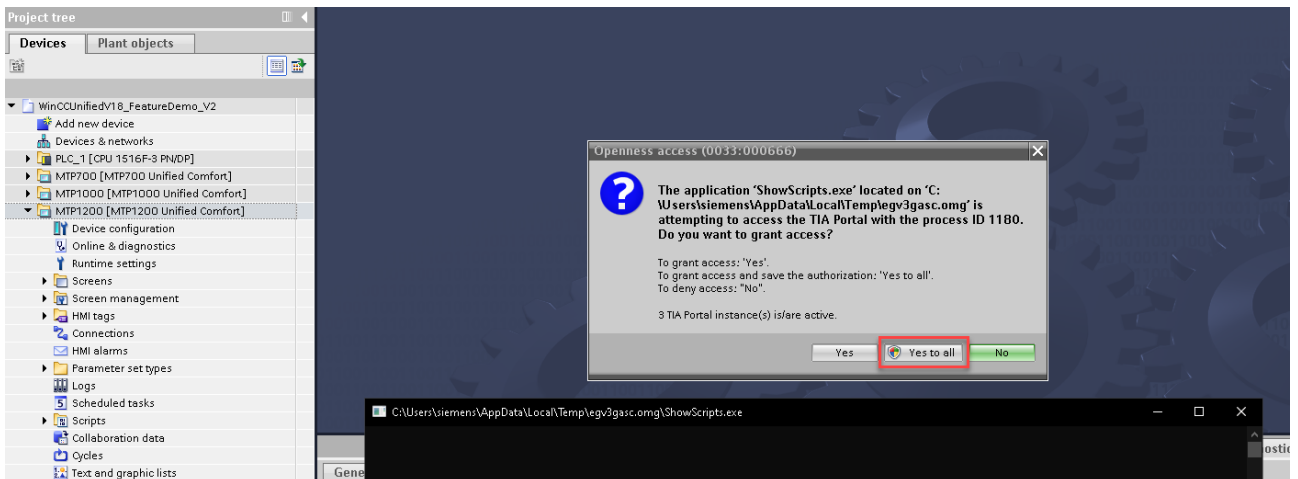


Figure 4-6 Confirmation of the Add-In's project access

After "yes to all" has been executed, a window opens in which you have to assign a screen name.

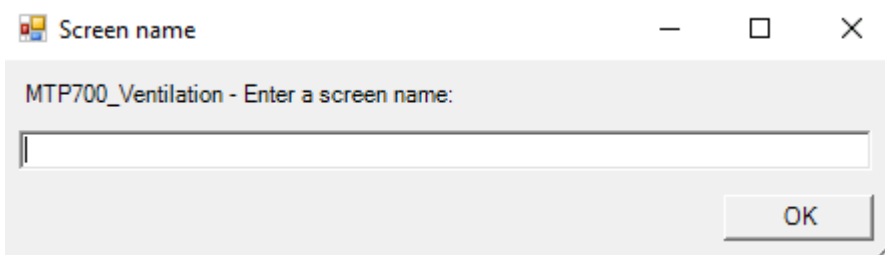


Figure 4-7 Enter a screen name

Afterwards you will see the analysis progress of the Add-In in the command window. If the tool runs successfully, the command window will close itself after running through all the screens in the device. In the folder "UserFiles" of your TIA Portal project folder there will be a new folder created with the respective device name in which you can find the exported scripts and a csv-based screen overview file.

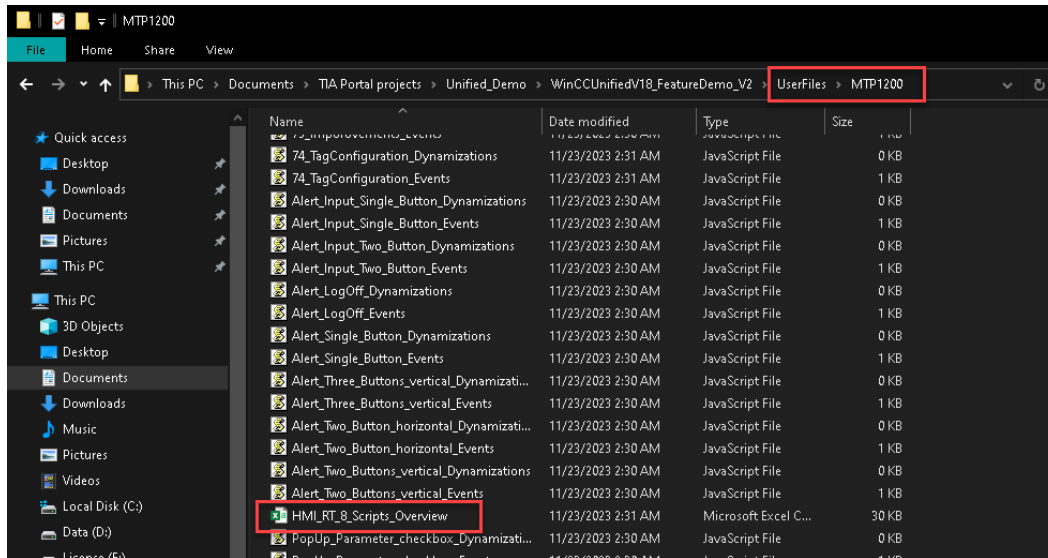


Figure 4-8 Results of the Add-In export in UserFiles > "DeviceName"

4.1.3. Preparation of the Screen Overview File

By default, the exported screen overview file has the format csv. To be able to adapt the file format and save the changes later you need to initially open the file e.g., with Microsoft Excel and save it separately with the file format ".xlsx". Afterwards you can work on this file and save the shown implemented changes permanently.

After the file gets opened the first time the values are still in comma-separated format and it is hard to evaluate the analysis.

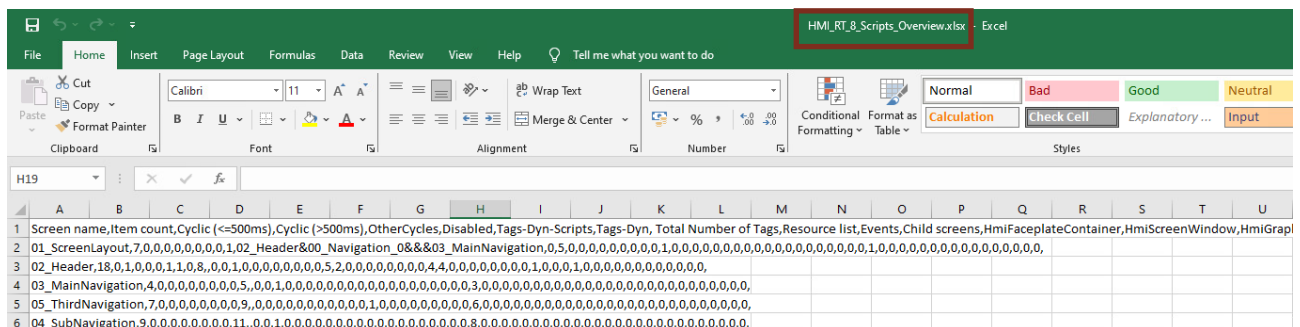


Figure 4-9 Screen overview file with comma-separated format

Therefore, the first step is to bring the analysis data into a well readable format.

1. Mark the first column which contains the comma-separated data.
2. Go to the register card "Data" and select the option "Text to Columns".
3. A wizard will open which guides you to get the right format. It is important to set the delimiter "comma" in the second step of the wizard. Afterwards the wizard can be finished, and you will see the values separated into columns.

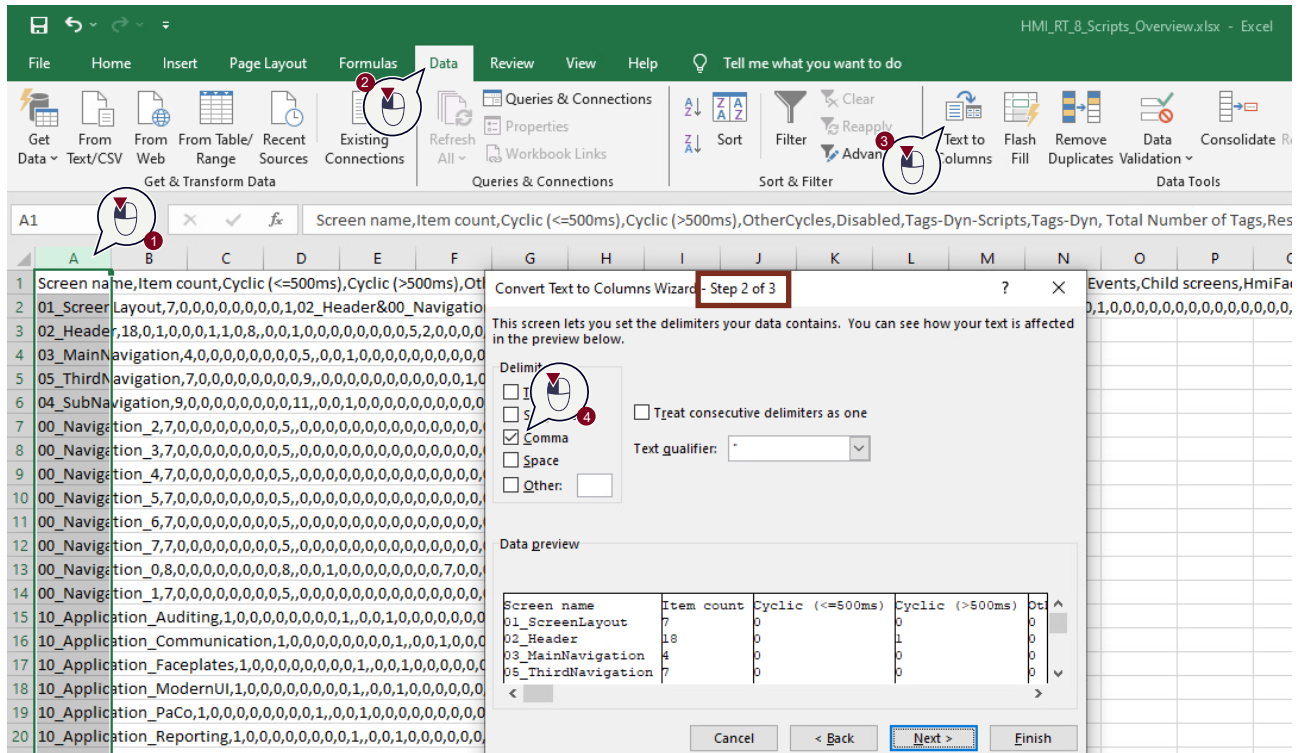


Figure 4-10 Preparation of the analysis data – step 1

- The readability can be increased even more by auto-fitting the column width depending on the table head texts. Therefore, all columns need to be marked and the option “AutoFit Column Width” in the register card “Home” needs to be selected.

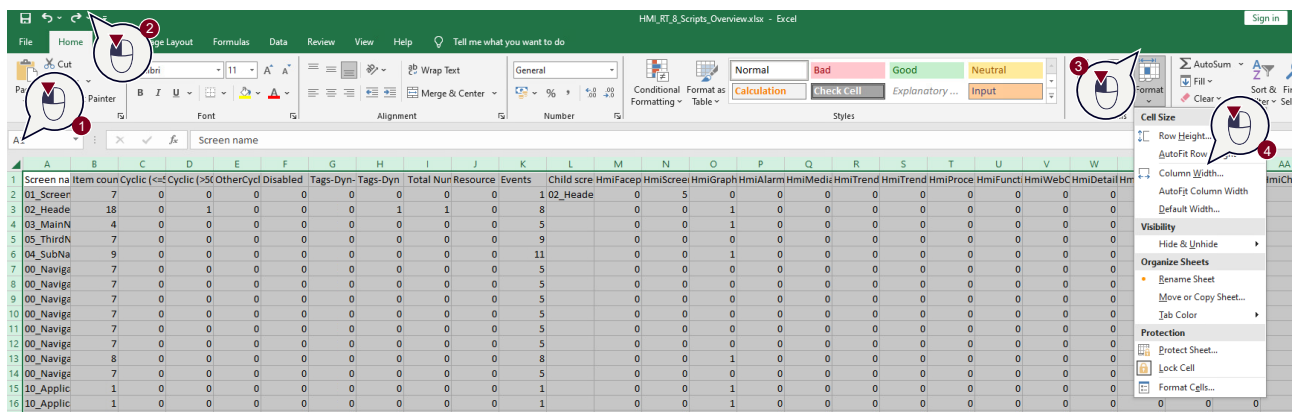


Figure 4-11 Preparation of the analysis data – step 2

After these two steps the readability of the screen analysis data is increased considerably, and the analysis of the project data can be started.

1	Screen name	Item count	Cyclic (<=500ms)	Cyclic (>500ms)	OtherCycles	Disabled	Tags-Dyn-Scripts	Tags-Dyn	Total Number of Tags	Resource list	Events	Child screens
2	01_ScreenLayout	7	0	0	0	0	0	0	0	0	0	1 02_Header&00_Navigation_0&&03_MainNavigation
3	02_Header	18	0	1	0	0	0	1	1	0	8	
4	03_MainNavigation	4	0	0	0	0	0	0	0	0	5	
5	05_ThirdNavigation	7	0	0	0	0	0	0	0	0	9	
6	04_SubNavigation	9	0	0	0	0	0	0	0	0	11	
7	00_Navigation_2	7	0	0	0	0	0	0	0	0	5	
8	00_Navigation_3	7	0	0	0	0	0	0	0	0	5	
9	00_Navigation_4	7	0	0	0	0	0	0	0	0	5	

Figure 4-12 Finalized preparation of the screen analysis file

4.1.4. Analysis of the Screen Overview File

This chapter contains information about how the data in the screen overview file can be related to the best practice topics mentioned in chapter 3 and the system limit information of WinCC Unified based devices which can be found in the respective manual of each version (see 19).

NOTE

For the following chapters be aware that the ShowScripts Add-In currently does not support the analysis of objects and configurations inside Faceplates but only the Faceplate containers itself and its properties.

Faceplates must be therefore analyzed manually in the library view of the TIA Portal project.

4.1.4.1. Analysis of Screen Context

With the formatted screen overview file, it is possible to see the screen context (screens with subordinate screen window levels) very comfortably.

NOTE

The screen context describes the total number of simultaneously existing screens windows with its objects, tags and dynamizations in WinCC Unified based applications.

In the example below you can see that the screen "02_ScreenLayout" has several child screens which means that the screen contains several screen windows with or without configured screens.

1	Screen name	Item count	Cyclic (<=500ms)	Cyclic (>500ms)	OtherCycles	Disabled	Tags-Dyn-Scripts	Tags-Dyn	Total Number of Tags	Resource list	Events	Child screens	HmiFaceplate
2	01_ScreenLayout	7	0	0	0	0	0	0	0	0	0	02_Header&00_Navigation_0&&03_MainNavigation	
3	02_Header	18	0	1	0	0	0	1	1	0	8		
4	03_MainNavigation	4	0	0	0	0	0	0	0	0	5		
5	05_ThirdNavigation	7	0	0	0	0	0	0	0	0	9		
6	04_SubNavigation	9	0	0	0	0	0	0	0	0	11		
7	00_Navigation_2	7	0	0	0	0	0	0	0	0	5		
8	00_Navigation_3	7	0	0	0	0	0	0	0	0	5		
9	00_Navigation_4	7	0	0	0	0	0	0	0	0	5		

Figure 4-13 Example for screen window hierarchy in "child screens" column

Every child screen which means a screen window with or without configured screen is separated from another one with an "&" letter. If there are two "&" letters following on each other this means that there is a screen window in the respective screen which has an empty "screen" property like shown below.

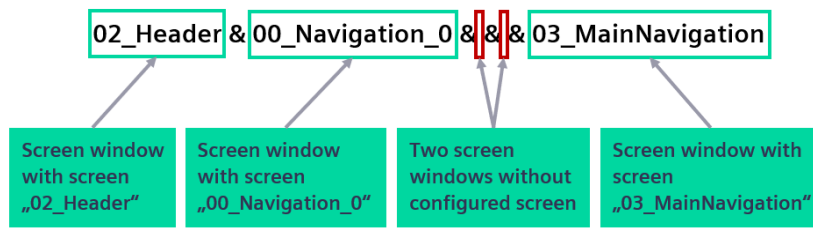


Figure 4-14 Explanation of “child screens” column based on “01_ScreenLayout” screen

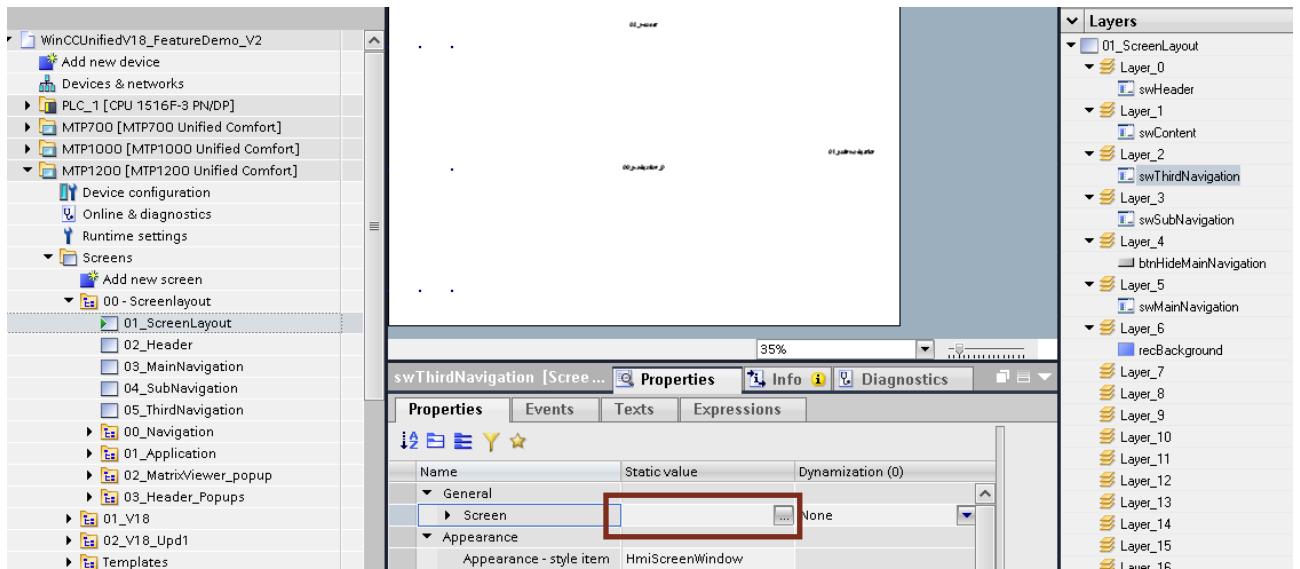


Figure 4-15 Example for screen window with empty screen property

Advantage of knowing the screen context structure

In the WinCC Unified manual further information about the system limits related to screens is listed. An extract was already mentioned in [3.1.2 Tidy up the Screen / Observe System Limits](#) .

NOTE

Be aware that the screen related system limits mentioned in the manual always refer to the overall screen context which can contain several further screen window levels. This can result in a larger footprint on the runtime load due to the additional objects, tags and dynamizations.

Screens		Unified Comfort 7-12"	Unified Comfort 15-22"
Maximum size in the engineering system		20,000 * 20,000 pixels	
Maximum size in runtime		20,000 * 20,000 pixels	
Number of screens		1200	
Number of lower-level screen windows		10	
Number of objects per screen	800	1200	
Number of objects from the "Controls" area per screen	40	80	
Number of tags per screen	600	800	

Figure 4-16 Screen related system limits of a Unified PC station

In complex applications it can be hard to check the whole screen context in the engineering system. Same applies to the runtime application itself where screen windows can be invisible but still contain loaded screens with all its objects, tags and dynamizations.

The ShowScripts Add-In provides a better understanding of the screen context in your application with the child screens information. This helps for avoiding overloaded screen contexts in an early stage of the project development or for later implementation of measures to reduce the load footprint by using the best practice recommendations in chapter 3.

Since the content of Faceplate instances on screens is not evaluated yet by the ShowScripts Add-In, the number of objects generated by them needs to be counted manually.

4.1.4.2. Usage of Controls

The exported screen overview file can also give you further hints about screens containing controls which means basic, advanced and custom controls. As already mentioned in chapter 3.1.1 the rendering footprint of controls can be much higher than other objects as basic objects and elements.

If you detect the usage of several controls within one screen or screen context evaluate a possible reduction of these controls like described in chapter 3.1.5. Wherever possible from operating point of view, splitting controls in different screens or screen contexts can also have a positive impact on the loading time of a screen. Especially two controls of the same type like in the figure below (marked with 1), can often be combined by one.

1	Screen name	Item count	Child screens	HmiAlarmControl	HmiMediaControl	HmiTrendControl	Hmi
2	01_ScreenLayout	7	02_Header&00_Navigation_0&&03_MainNavigation	0	0	0	0
3	02_Header	18		0	0	0	0
4	03_MainNavigation	4		0	0	0	0
5	05_ThirrdNavigation	7		0	0	0	0
6	04_SubNavigation	9		0	0	0	0
7	00_Navigation_2	7		0	0	0	0
8	00_Navigation_3	7		0	0	0	0
9	00_Navigation_4	7		0	0	0	0
10	00_Navigation_5	7		0	0	0	0
11	00_Navigation_6	7		0	0	0	0
12	00_Navigation_7	7		0	0	0	0
13	00_Navigation_0	8		0	0	0	0
14	00_Navigation_1	7		0	0	0	0
15	10_Application_Auditing	1		0	0	0	0
16	10_Application_Communication	1		2	0	1	0
17	10_Application_Faceplates	1		0	0	0	0
18	10_Application_ModernUI	1		0	0	0	0
19	10_Application_PaCo	1		0	0	0	0

Figure 4-17 Information about control usage per screen

4.1.4.3. Usage of Dynamizations

There is even more object related information that can be analyzed by use of the screen overview file of the ShowScripts Add-In. You can find information about tags, dynamizations and events configured on each screen. This helps you to get an overview on which part of the application to look at when opening the TIA Portal project for further analysis, e.g. when cyclic scripts are used on several screens (refer to section 3.2).

The information can be also taken as reference if you want to have a closer look on the scripts of certain via Visual Studio Code (see chapter 4.1.5).

The related columns in the screen overview file will be elaborated more in detail in the following sections:

Used tags per screen

The "total number of tags" is the sum of all tags that are used as trigger on the screen. These trigger tags will be subscribed to the PLC on screen loading and will influence the screen loading time. Trigger tags are used in dynamizations only.

1	Screen name	Total Number of Tags
165	69_a_Subscription	4
166	69_b_RootWindow	0

Figure 4-18 Total tags column in overview file

NOTE Expressions for the dynamization of objects (available since WinCC Unified V18) cannot be evaluated by the ShowScripts Add-In yet and are therefore not included in the counts of the screen overview file.

Examples for the described dynamizations can be found in chapter 3.2 Use of Dynamizations.

Tag dynamizations

The "Tags-Dyn" column lists the amount of tag dynamizations configured on a screen.

1	Screen name	Tags-Dyn
165	69_a_Subscription	4
166	69_b_RootWindow	0

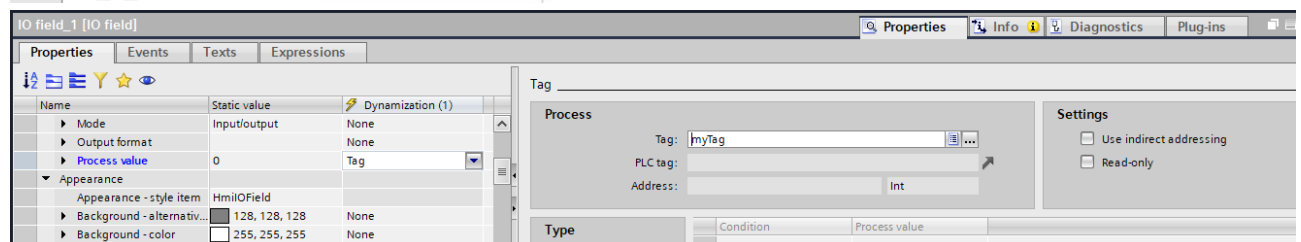


Figure 4-19 Tag dynamization column in overview file

Script dynamizations (tag triggered)

The "Tags-Dyn-Scripts" column lists the amount of script dynamizations based on a tag trigger per screen.

Screen name	Tags-Dyn-Scripts
01_LiveDemo_JS	2
02_HandsOn_JS	2

Figure 4-20 Script dynamization (tag triggered) column in overview file

Script dynamizations (cyclic)

There are three columns referring to cyclic script dynamizations based on default available and own defined cycles.

Screen name	Cyclic (<=500ms)	Cyclic (>500ms)	OtherCycles
01_LiveDemo_JS	0	0	0
02_HandsOn_JS	0	0	0

Figure 4-21 Script dynamization (cyclic) columns in overview file

Events

The "Events" column lists the sum of following configurations:

14. Events on screens and screen items

15. The amount of "on-change" triggered scripts at properties of screen items

Screen name	Events
01_LiveDemo_JS	6
02_HandsOn_JS	1

Figure 4-22 Events columns in overview file

4.1.5. Analysis of the exported Scripts

Next to the screen overview file the ShowScripts Add-In exports all configured scripts within a screen and its objects.

As shown below two export files (js-format) for each screen are generated:

1. "Dynamizations" file: contains all script dynamizations configured on a screen and its objects.
2. "Events" file: contains all "on-change" triggered scripts (on screen object properties) and scripts triggered on events (of screens and objects)

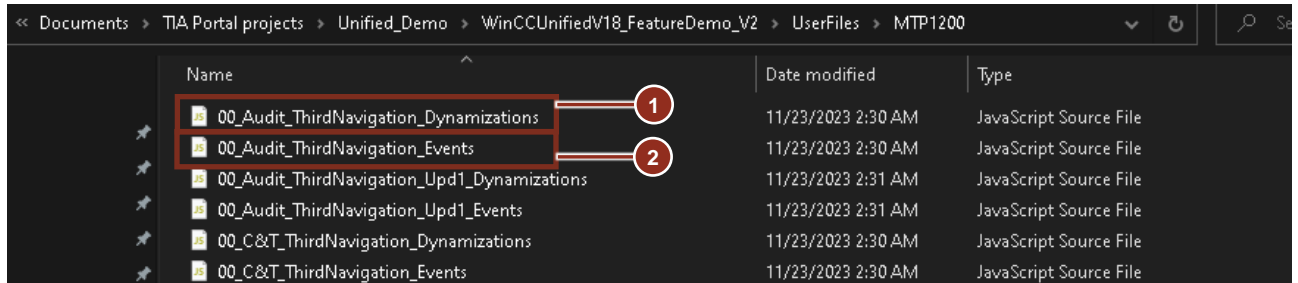


Figure 4-23 Script export files of ShowScripts Add-In

NOTE

Currently the ShowScripts Add-In does not support the export of scripts inside Faceplates, global modules, scheduled tasks and libraries. The scripts inside Faceplates and scheduled tasks must be analyzed separately in the TIA Portal project.

For the global module and library scripts there is an alternative since WinCC Unified V19 to retrieve script exports by use of the WinCC Unified JS Connector (see chapter [4.2](#))

Script analysis using VS Code

All exported scripts can be analyzed quite comfortably according to the best practice recommendations in chapter [3](#).

This can be done with any editor like Notepad++ for each file separately or if you want to execute a global search through all the script files by use of the free of charge code editor "Visual Studio Code" (VS Code).

Therefore, download VS Code and once it is installed follow the further steps.

1. Open the program and drag & drop the whole export folder of ShowScripts Add-In from the Windows Explorer into the VS Code "Explorer" area.

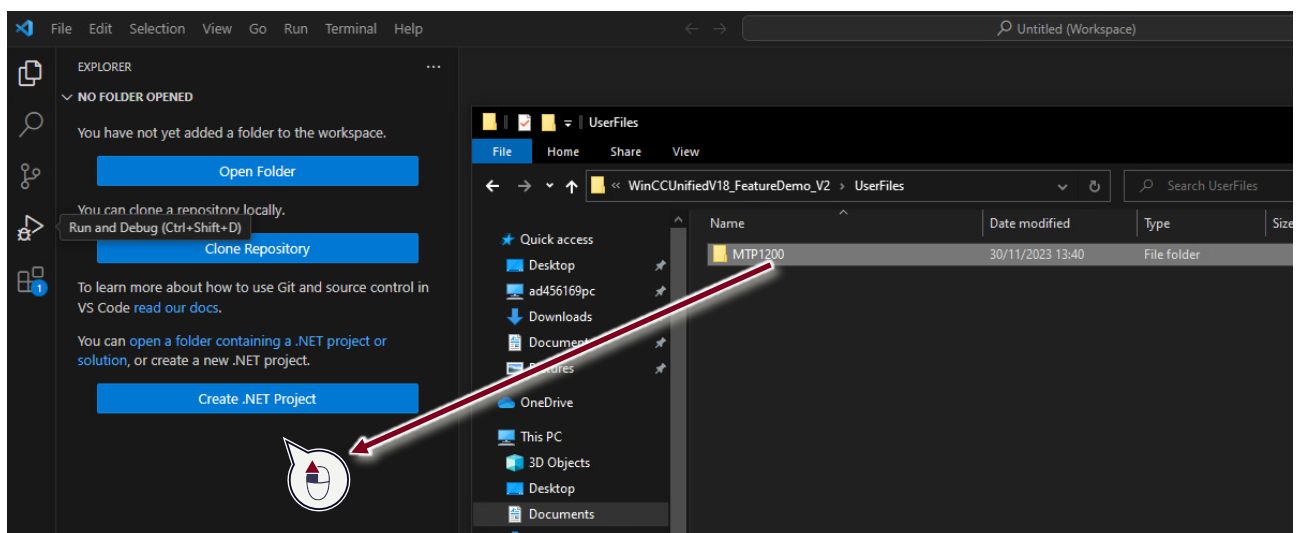


Figure 4-24 Opening script export folder with VS Code

2. Afterwards the content of the export folder will get listed in the left area of VS Code (1). As soon as you select any of the script export files the content will be displayed in the work area of the application (2).

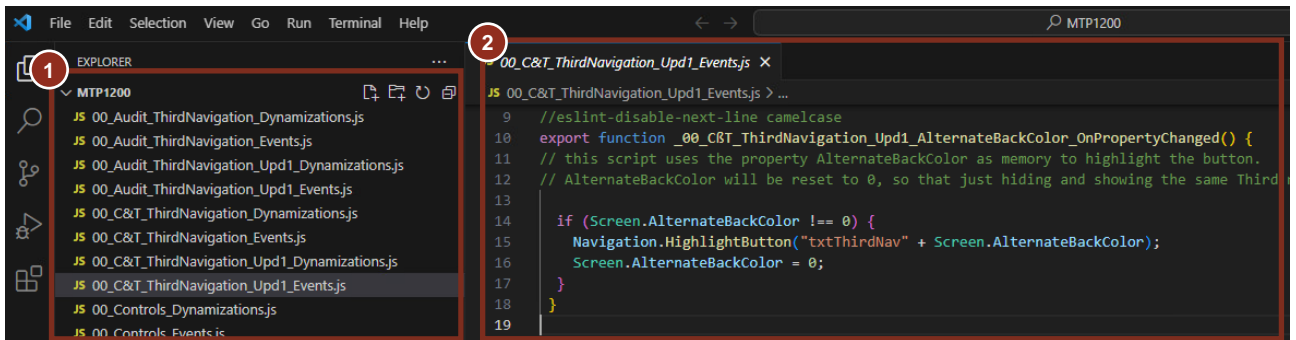


Figure 4-25 Opened script folder in VS Code

3. The identification of the script type within WinCC Unified is also possible via the function title. In the figure below two examples are shown:

- a. "On-change" triggered script on the property "AlternateBackColor" of the screen "00_C&T_ThirdNavigation_Updater1"
- b. "Click left mouse button" event on the object "txtThirdNav1"

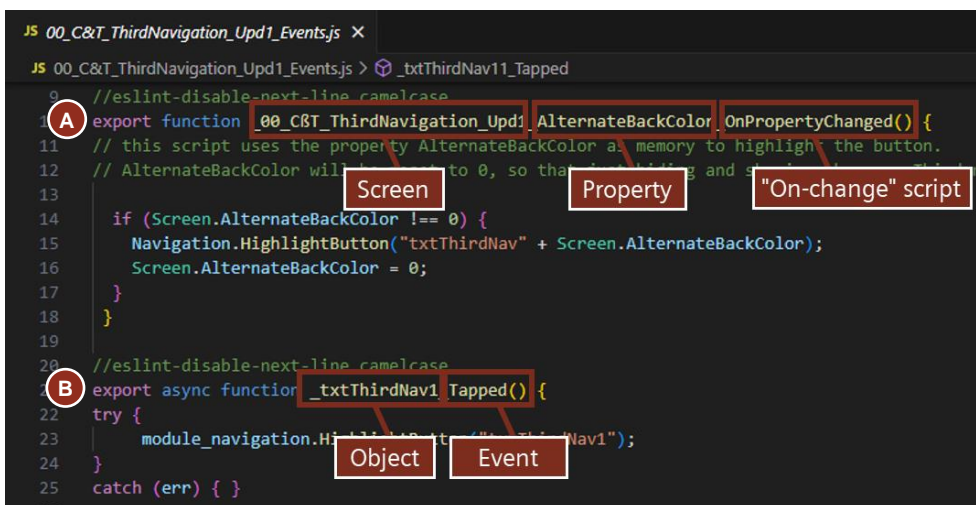


Figure 4-26 Extract of script export in VS Code

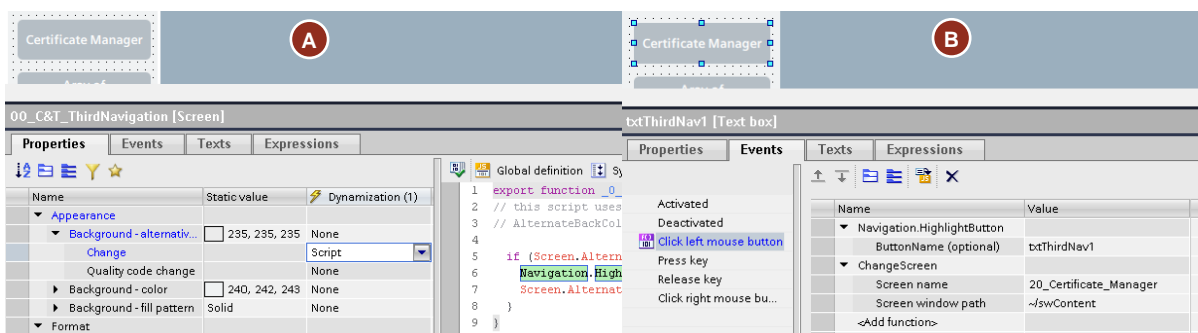


Figure 4-27 Location of the exported scripts in TIA Portal project

NOTE

Be aware that the ShowScripts Add-In also exports function lists as scripts (see example B). The reason is that system functions are only engineering related but during runtime they get executed as script code as well.

4. By the use of the global search function of VS Code the exported scripts can be analyzed according to the best practice recommendations in chapter 3.
5. In the example below the scripts get scanned for example for the keyword "SetTagValue" to be able to find relevant scripts for the replacement with tag sets as described in chapter 3.4.2.1.
- 6.

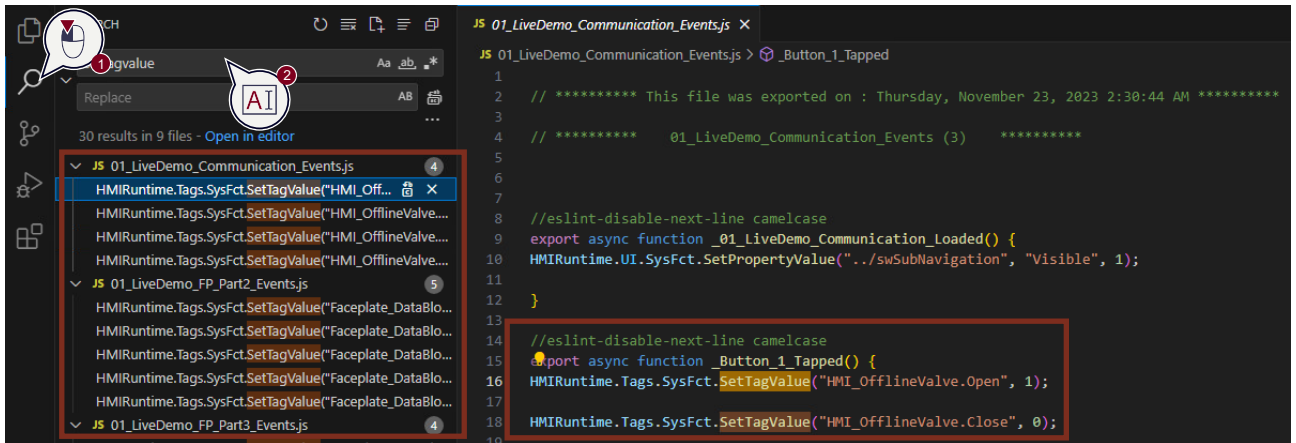


Figure 4-28 Keyword search in VS Code for "SetTagValue"

Other examples to search for could be among others:

- « for » to find unnecessary loop runs
- « alarm » for the usage of computing-intensive methods, e.g. GetActiveAlarms()
- « await » Numerous uses of await in connection with asynchronous function calls

NOTE

Furthermore, Visual Studio Code can use the style guide configuration, that offers verification and automatic correction of script code inside the Visual Studio Code editor according to the programming style guide specifications. The style guide configuration can be found also [on SIOS](#)

4.2. WinCC Unified JS Connector V19

A first step for the screen related script export and analysis can be done by use of the ShowScripts Add-In. If you detect the usage of global module scripts in the screen related scripts you can make use of the “Simatic WinCC Unified JS Connector” which is a Visual Studio Code extension to export, edit and import these scripts of the global modules.

NOTE The WinCC Unified JS Connector extension is supported since WinCC Unified ES V19.

Below you can see an example for the usage of global modules. If you want to see the content of the used scripts in the imported module the WinCC Unified JS Connector can be used.

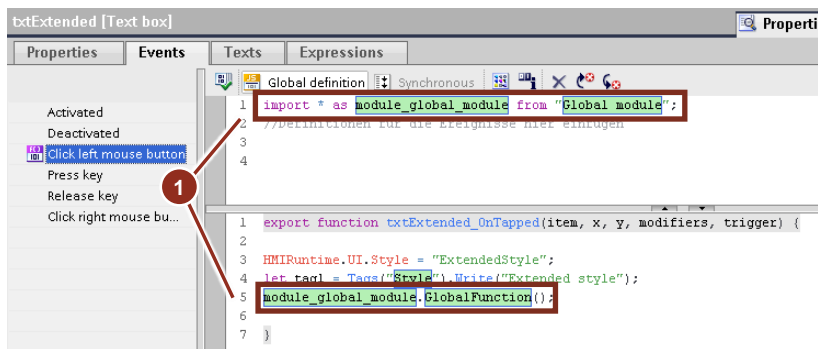


Figure 4-29 Usage of a global function in WinCC Unified script editor

The WinCC Unified JS Connector extension is free of charge and available in a [separate entry](#) of the Siemens Industry Online Support (SIOS). With the extension following workflow can be applied by users for project analysis and optimization:

1. Export of scripts in global modules from the TIA Portal project

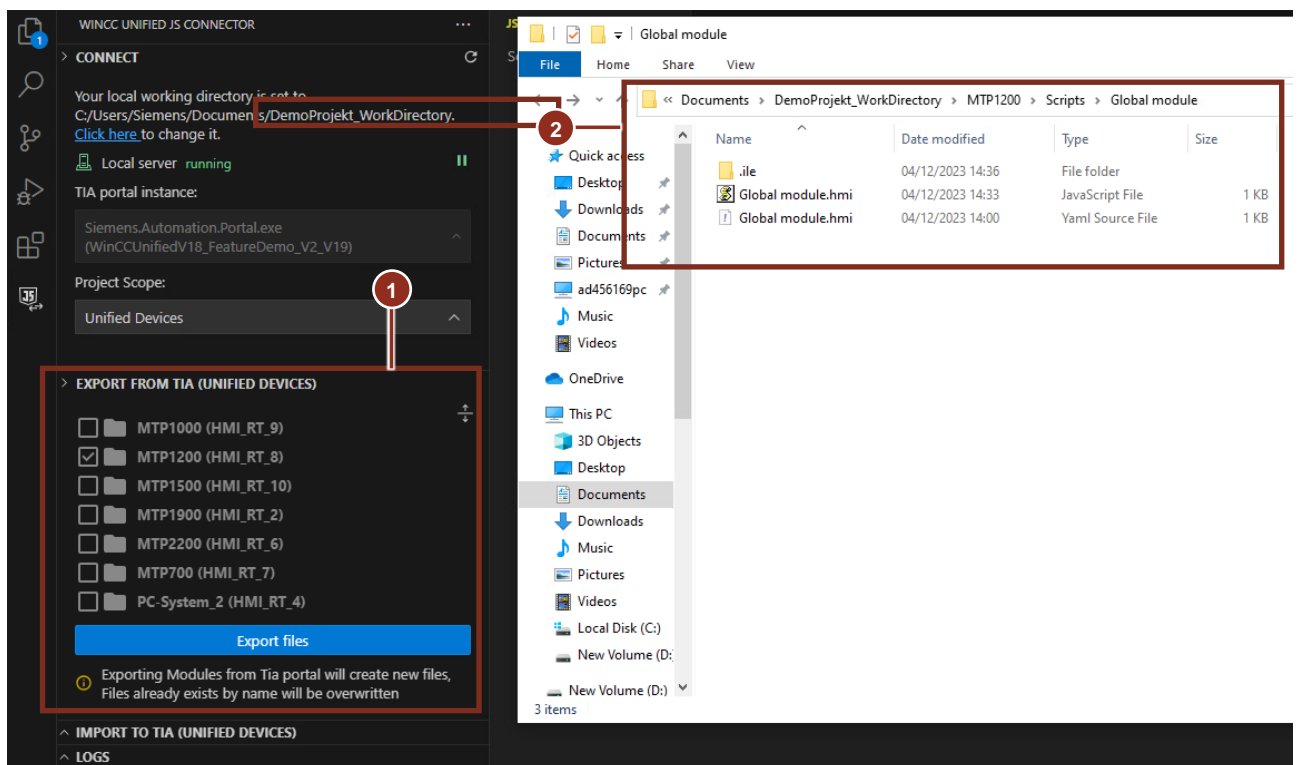


Figure 4-30 Export function of the extension (1) with the exported files in the work directory (2)

2. Analysis and adaption of the scripts in VS Code (according to the best practices in chapter 3)

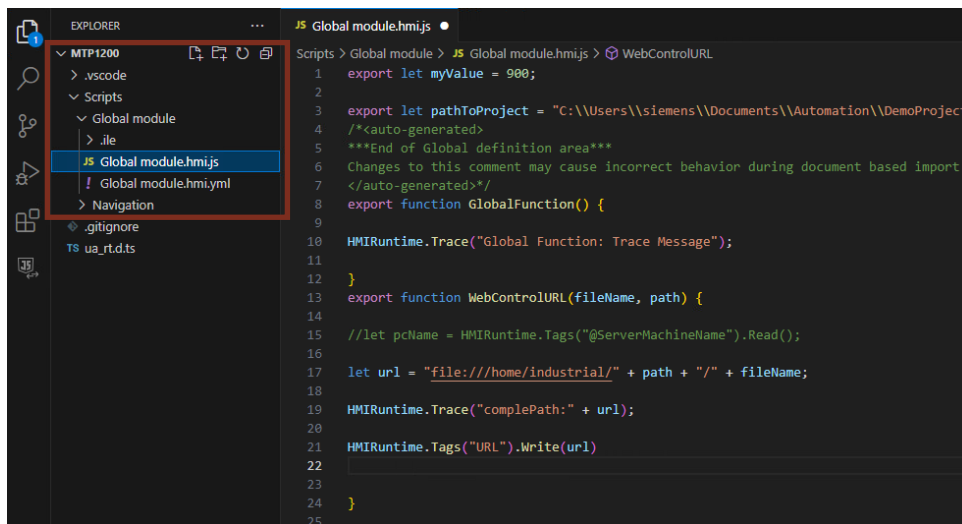


Figure 4-31 Analysis of an exported global module script via VS Code

3. Import of the adapted scripts to the TIA Portal project

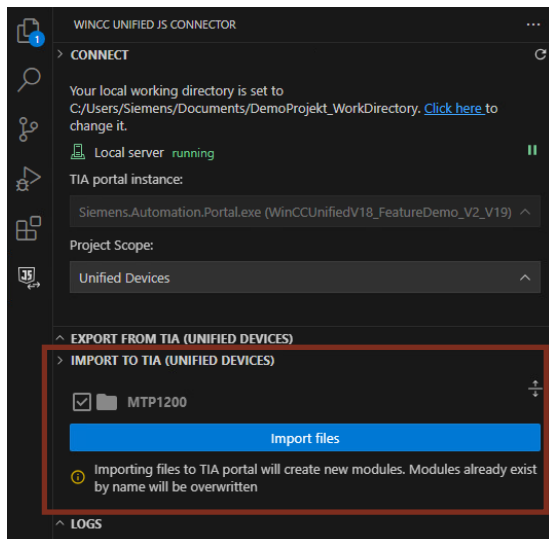


Figure 4-32 Import of scripts to the TIA Portal project

For more detailed information about the installation and usage of the WinCC Unified JS Connector please refer to the provided manual in the extension related [SIOS entry](#).

4.3. Runtime Analysis

If you have successfully analyzed the project using the above mentioned ShowScripts Add-In and the WinCC Unified JS connector and applied changes and optimizations in the screen engineering or script code, it is possible that there are still misbehaviors during screen change or the screen loading time is still not sufficient enough.

To find further potential for project optimization it can be useful to analyze problematic screen change more in detail by use of following tools:

1. RTIL Trace Viewer (for all Unified based devices)
2. Script Debugger for WinCC Unified (only for Unified PC Runtime and Panel Simulation at the Engineering PC)
- 3.

4.3.1. RTIL Trace Viewer

The RTIL Trace Viewer is an external application provided with the WinCC Unified installation. It displays all available trace messages that a Unified Basic/Comfort Panel or PC Runtime provides during runtime operation.

NOTE

The setup procedure for using the RTIL Trace Viewer on WinCC Unified based devices is described more detailed in the following FAQ:

[Use of Trace Viewer with WinCC Unified Comfort Panel or WinCC Unified PC Runtime](#)

The procedure described for the Unified Comfort Panels also applies the identically for the Unified Basic Panels.

Usage of the RTIL Trace Viewer for project analysis

The Trace Viewer can be used for further analysis of following scenarios:

1. When capturing a screen change there can be further errors or warnings which are related to script execution or in general to the project configuration.
2. These trace messages can show you a certain influence on the overall screen change performance and need to be resolved as far as possible.
- 3.

An example for such errors is visible below in which a configured script tries to access an object ("Button_4") in the runtime which does not exist and therefore error messages are shown.

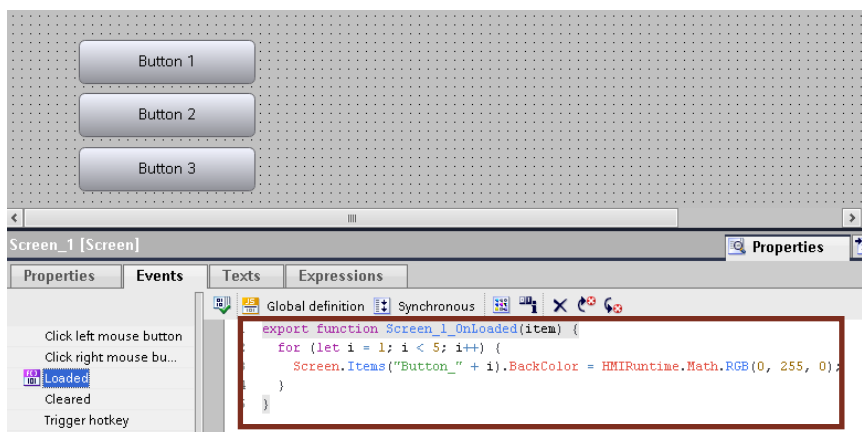


Figure 4-33 Script access to buttons on a screen

4. The execution result of the script is shown understandably in the RTIL Trace Viewer and needs to be resolved by adapting the loop in the script.

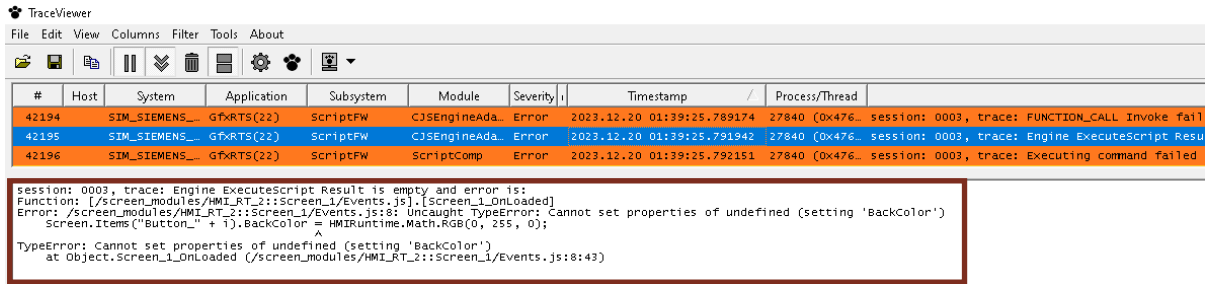


Figure 4-34 Result of script execution in the trace viewer application

NOTE

For better analyzability of the traces in the RTIL Trace Viewer, special display filters can be applied to the default view. In order to see just the script related traces you can set the filter for “Subsystem” to “ScriptFW”.

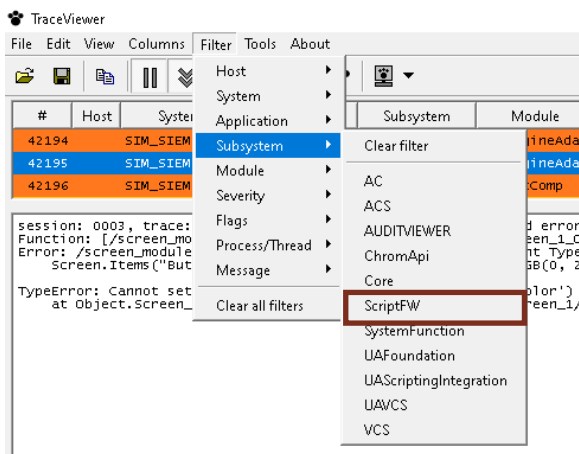


Figure 4-35 Script filter in RTIL Trace Viewer

- 5.
6. The Trace Viewer can be an alternative to detect and understand possible failures of scripts in the project. As there is no further debugging possibility on the Unified Basic/Comfort Panel hardware the usage of trace messages in conjunction with the RTIL Trace Viewer can help to:
 - See which scripts get executed in certain operating scenarios
 - Check how often these scripts are executed
 - How long the scripts are being processed

For the Unified PC Runtime, a useful alternative to the Trace Viewer could be the Script Debugger which is described in the next chapter.

NOTE

Further information about the usage of the trace viewer and configuration of trace messages can be found in the document [SIMATIC WinCC Unified - Tips and Tricks for Scripting \(JavaScript\)](#) in chapter 5.16.2 “Diagnostics via RTIL Trace Viewer”.

4.3.1.1. Additional Flags

For further analyses additional options can be enabled in the Trace Viewer, to get the following information:

- See which scripts get executed in certain operating scenarios
- Check how often these scripts are executed
- How long the scripts are being processed
- Number of variables read during screen construction

The Performance Flag

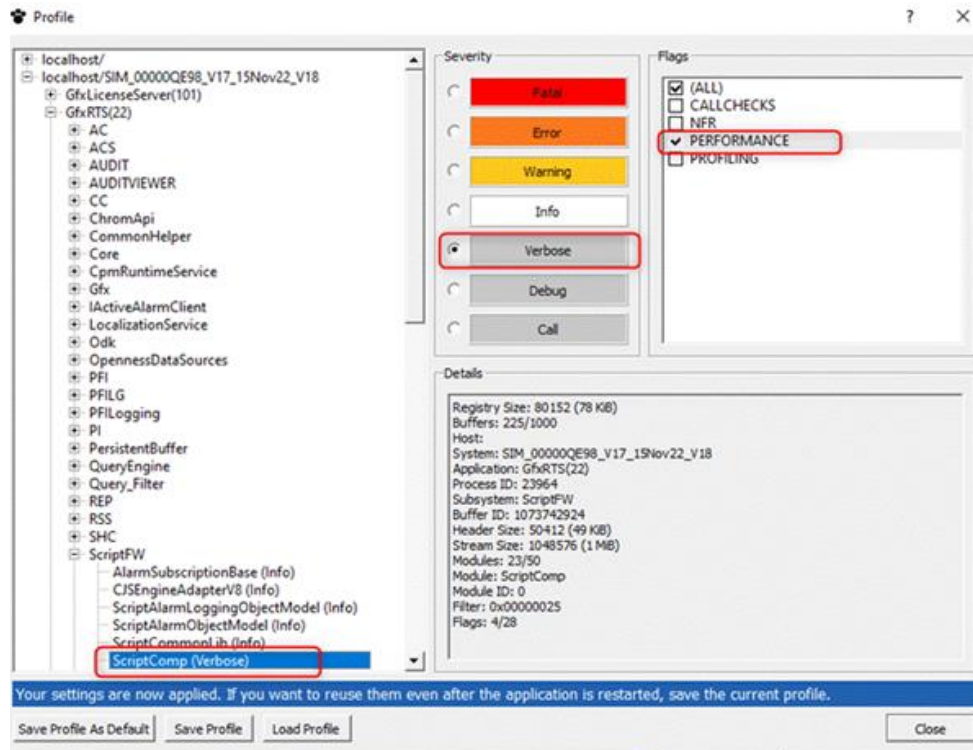


Figure 4-36 Set the performance flag

With the flag activated, you gain an overview of the amount of scripts being executed. This allows you to infer whether scripts are recurring, indicating a loop or cyclic trigger. Additionally, you receive the following information such as DoCommand (script duration) and ExecuteCommand (script trigger duration) times. You can analyze these times and improve them through script optimization.

The STAT_HISTO Flag

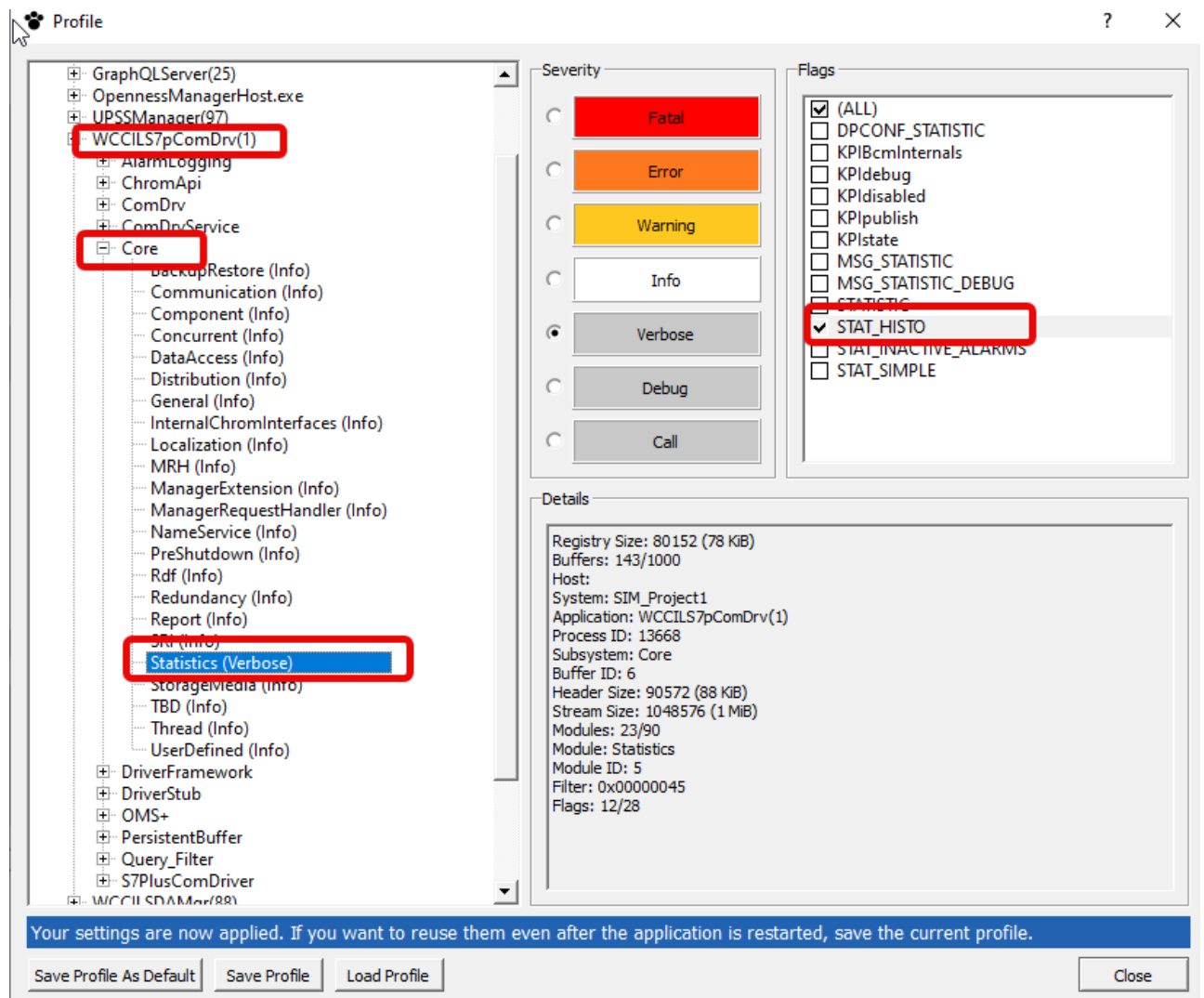


Figure 4-37 Enable the STAT_HISTO flag

Using this flag, you gain an overview of how many variables are being read during screen construction (screen load). The quantity of variables read during screen construction (screen load) impacts performance.

4.3.2. Script Debugger

On a Unified PC Runtime and Panel Simulation at Engineering PC there is the possibility to use the Script Debugger in combination with Google Chrome.

The debugger offers you further possibilities for script analysis offering typical functions like setting breakpoints and step-by-step execution.

NOTE

A detailed description about the setup and usage of the Script Debugger in WinCC Unified is available in the [WinCC Unified V19 manual](#).

Usage of the Script Debugger for screen change analysis

With the Script Debugger of the Unified PC Runtime certain operation scenarios can be further evaluated to:

- See which scripts get executed in certain operating scenarios. When a breakpoint is set e.g. in a loaded event, by clicking through the single steps, also all following script executions are shown.
- Check how often these scripts are executed, by counting how often a breakpoint is reached in a script.

If you want to debug the script executions during the screen change the project needs to be prepared in a certain manner to be able to debug the simultaneously executed scripts. The following steps show how to configure a breakpoint in the load event of the base screen and trigger the loading again for stepping through all scripts that are executed during the initial screen load as this is the more difficult configuration from the two mentioned above.

1. Activate the script debugger in the runtime manager

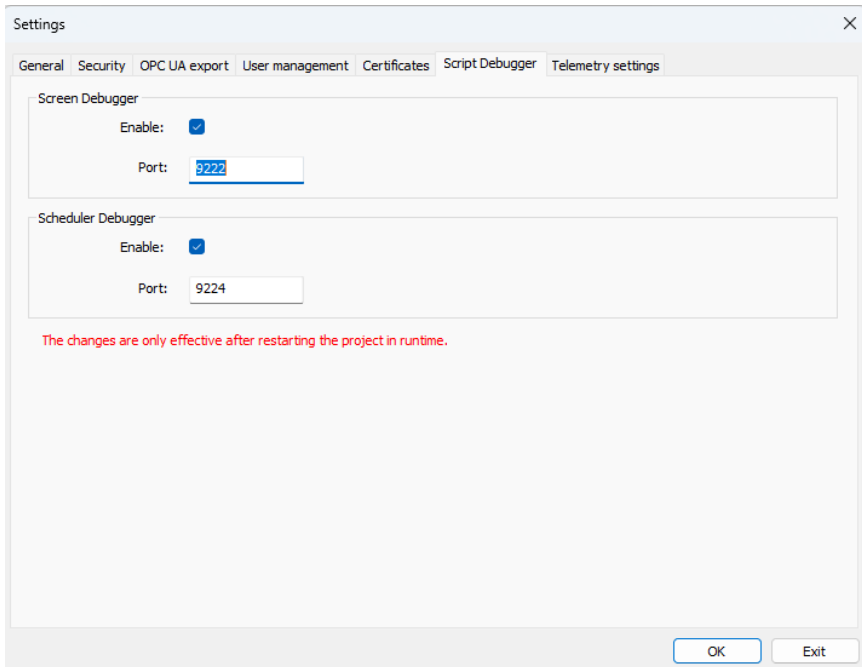
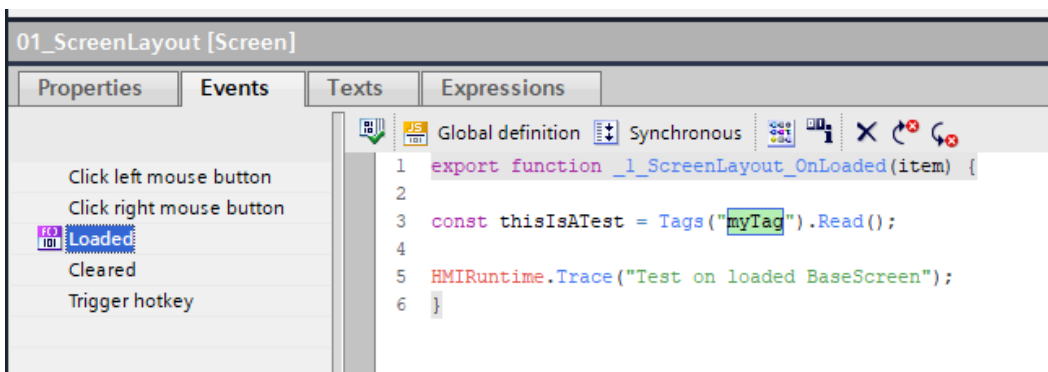


Figure 4-38 Enable script debugger in runtime manager settings

2. Configure a code in the loaded event of the projects base screen. This is used later to set a break point at the position where the base screen starts loading. Define an event that you can trigger during runtime (e.g., through a mouse click event of a screen item), that toggles the base screen from one that is not your current base screen and then back to the original base screen. If you only set the base screen again to the original one, the loaded event will not be triggered.



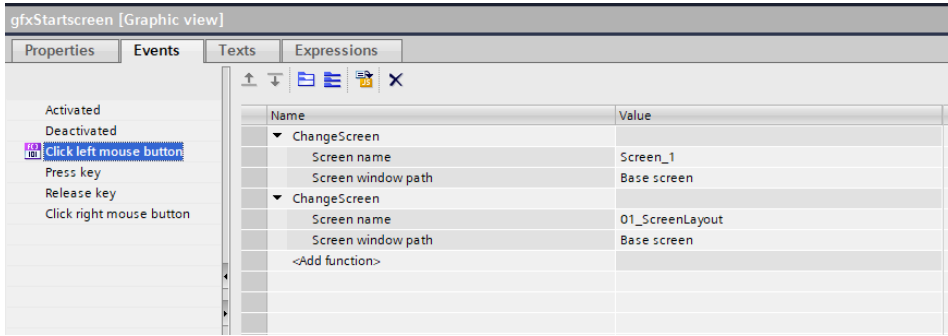


Figure 4-39 Refresh of the base screen and code for the loaded event

- 3.
4. Start the Runtime
5. Open an additional browser window and type: `chrome://inspect/`

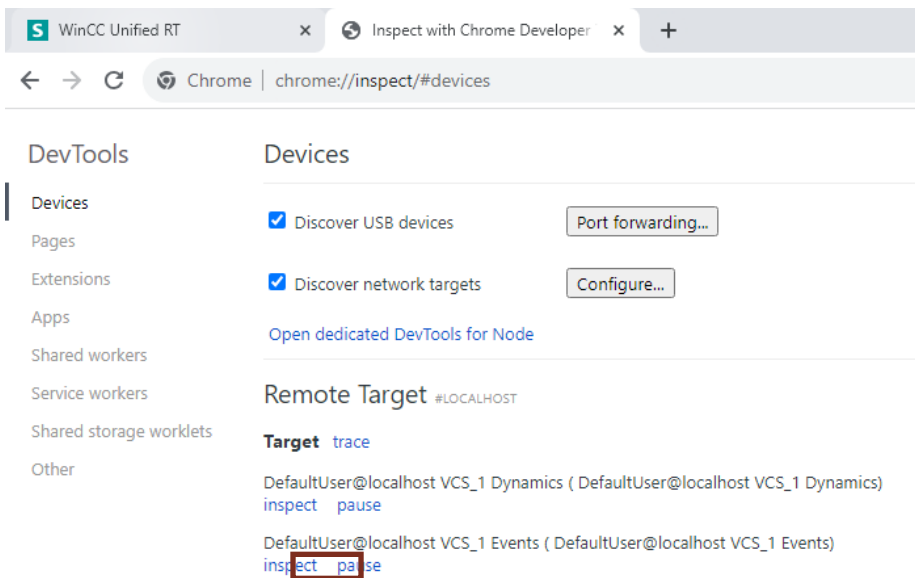


Figure 4-40 Chrome dev tools page

6. Click on the inspect link of the event context. With that, a new browser client is started

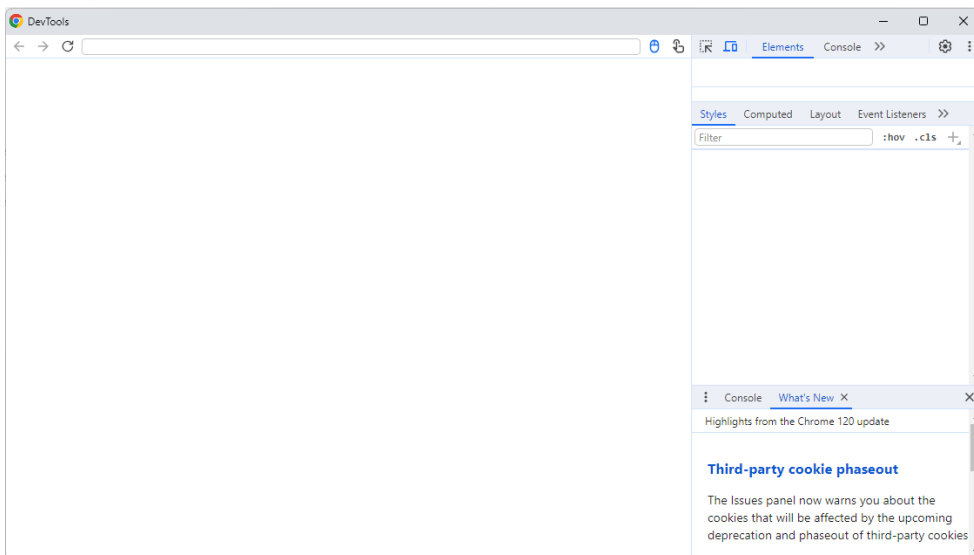


Figure 4-41 DevTools browser window

- Configure the browser window by dragging the right window more to the left and switch to the source tab. Unfold the (no domain) directory and select your base screen Event.js. There is the previous defined code in the loaded event of the base screen. The list shows all current loaded scripts. The list therefore differs depending on the current opened screens.

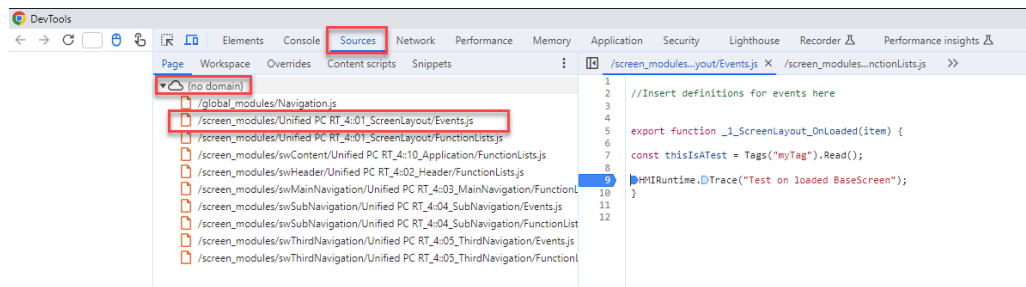


Figure 4-42 Script debugger - set break point

- Set a break point in the loaded event
- Go back to the Runtime and trigger the event for the base screen change
- The debugger browser windows will automatically open and jump to the breakpoint. You can also see on the right side the values of the already executed lines. Click on the "Step into next function" button to go step by step through the script

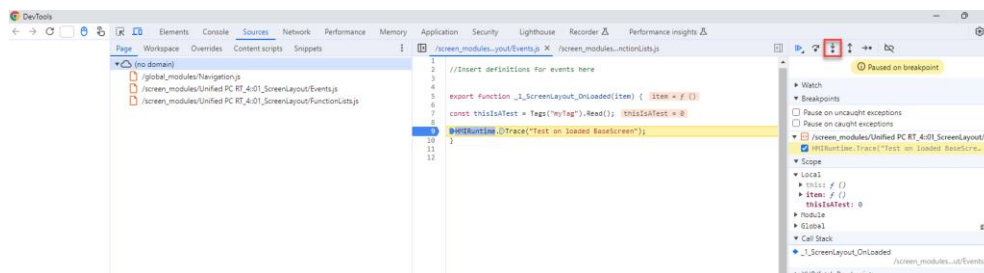


Figure 4-43 Script debugger - jump into break point

You can follow that global modules are loaded

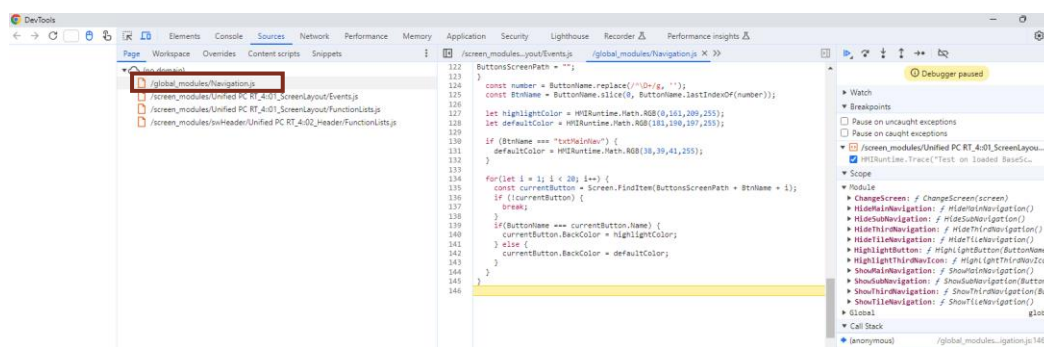


Figure 4-44 Script debugger - loading script modules

And loaded event of nested screen windows and Faceplates

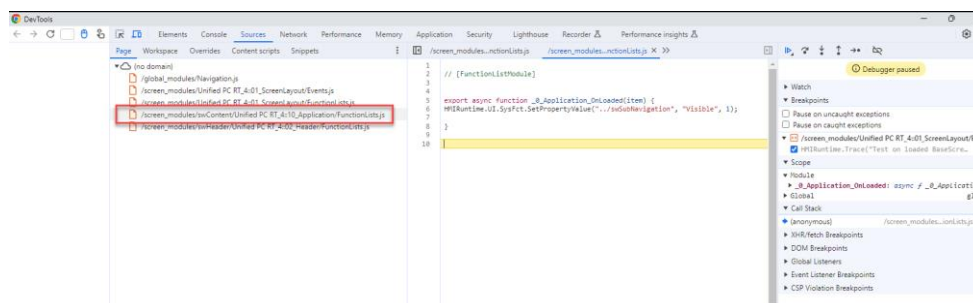


Figure 4-45 Script debugger - loaded event screen window

Every script that is executed will be shown. Once you want to end the debugging and execute the rest of the scripts at once, click on the “Resume script execution” button. If there are more breakpoints defined, the debugger will jump to the next breakpoint.

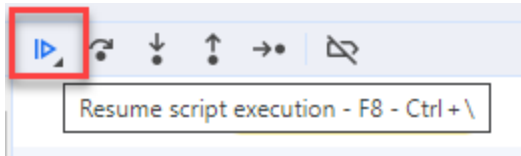


Figure 4-46 Resume script execution

12.

13. For every new download a new DevTools session needs to be opened. That is the reason for the manual trigger of the loaded event of the base screen

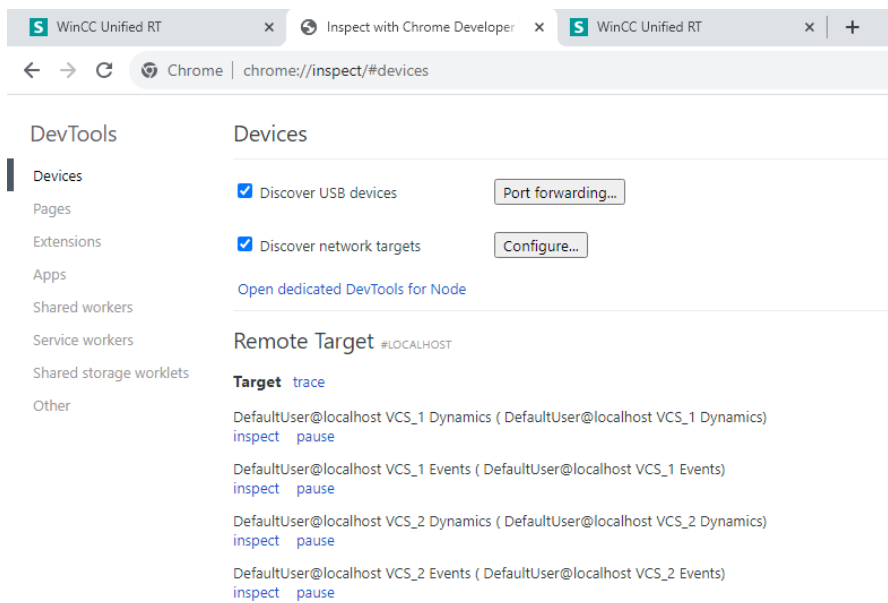


Figure 4-47 Google Chrome inspect page for selection runtime session

5. Appendix

5.1. Service and support

SiePortal

The integrated platform for product selection, purchasing and support - and connection of Industry Mall and Online support. The SiePortal home page replaces the previous home pages of the Industry Mall and the Online Support Portal (SIOS) and combines them.

- Products & Services
In Products & Services, you can find all our offerings as previously available in Mall Catalog.
- Support
In Support, you can find all information helpful for resolving technical issues with our products.
- mySieportal
mySiePortal collects all your personal data and processes, from your account to current orders, service requests and more. You can only see the full range of functions here after you have logged in.

You can access SiePortal via this address: sieportal.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

Industry Online Support app

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



5.2. Links and literature

Nr. Thema

11	Siemens Industry Online Support https://support.industry.siemens.com
12	Link to this entry page of this application example https://support.industry.siemens.com/cs/ww/en/view/109827603
13	Beginners Guide: Quick entry and conversion with WinCC Unified https://support.industry.siemens.com/cs/ww/en/view/109810917
14	HMI design with the HMI Template Suite https://support.industry.siemens.com/cs/ww/en/view/91174767
15	SIMATIC WinCC Unified Tutorial Center (Videos) https://support.industry.siemens.com/cs/ww/en/view/109782433
16	SIMATC WinCC Unified – Tips and Tricks for Scripting (JavaScript) https://support.industry.siemens.com/cs/ww/en/view/109758536
17	TIA- Add-In -ShowScripts https://github.com/tia-portal-applications/TIA-Add-In-ShowScripts
18	Developing WinCC Unified JavaScript code and checking style guide with Visual Studio Code https://support.industry.siemens.com/cs/ww/en/view/109801600
19	SIMATIC HMI WinCC Unified V19 for system limits https://support.industry.siemens.com/cs/ww/en/view/109828368/170192329611
10	SIMATIC WinCC Unified Corporate Designer V19 https://support.industry.siemens.com/cs/ww/en/view/109824234

5.3. Change documentation

Version	Date	Modification
V1.0	01/2024	First version
V2.0	06/2024	Update for V19 Upd2