Siemens
Industry
Online
Support

APPLICATION EXAMPLE

SIMATIC WinCC Unified Tips and Tricks for Scripting (JavaScript)

SIMATIC WinCC Unified (Engineering)

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

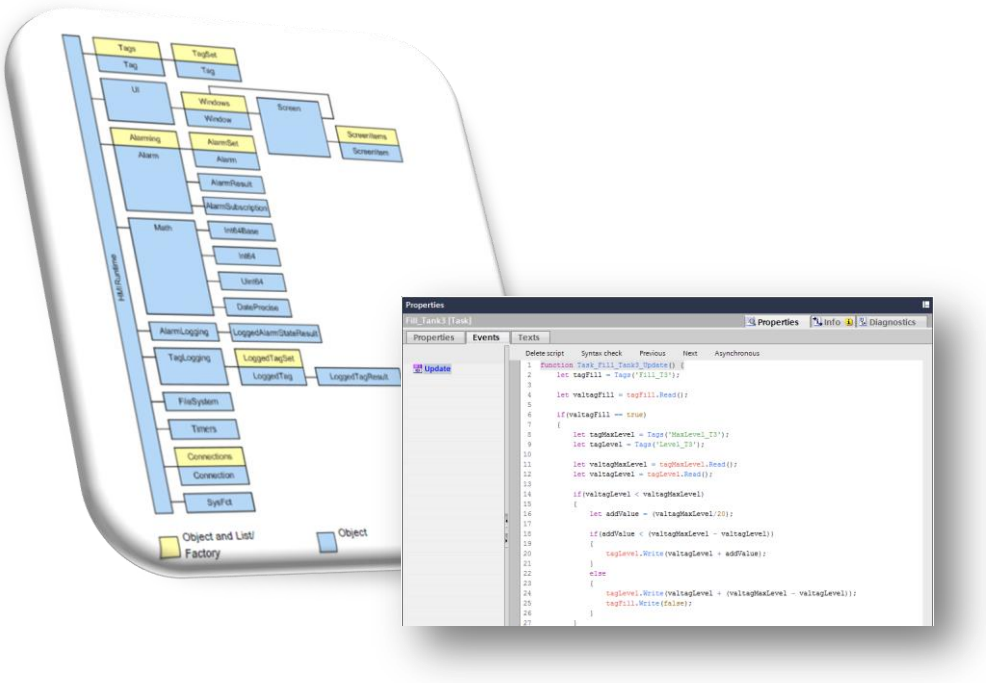
1.1. Overview

SIMATIC WinCC Unified uses JavaScript as a script language and therefore provides a modern script environment, which you can typically use to automate screens and objects.

The script environment maps individual elements of the system components via an object model, e.g. screen of the graphic runtime system. You reference this object model in your script languages, this allowing you to access different functions in an object-oriented approach.

The application example will show you how to use scripts in SIMATIC WinCC Unified. Selected examples will also serve to show you tips and tricks for manual scripting, which you can use in your application.

Figure 1-1



1.2. Components Used

The following hardware and software components were used to create this application example:

Table 1-1

Component	Quantity	Article number	Comment
SIMATIC WinCC Unified V20 (Engineering)	1	6AV2153-....1-6...	-
SIMATIC WinCC Unified V20 (Runtime)	1	6AV2154-....1-6...	-
SIMATIC HMI Unified Comfort Panel MTP700	1	6AV2128-3GB36-0AX0	Alternatively, you can use any other SIMATIC HMI Unified Comfort Panel.

2. General Information

This section will familiarize you with general information surrounding the topic of scripting in WinCC Unified.

Note

Impact of script errors on performance:

Script errors can have a significant impact on performance, as they can interrupt the execution of processes or cause dependencies to malfunction. This applies in particular to the incorrect use of tags and similar elements.

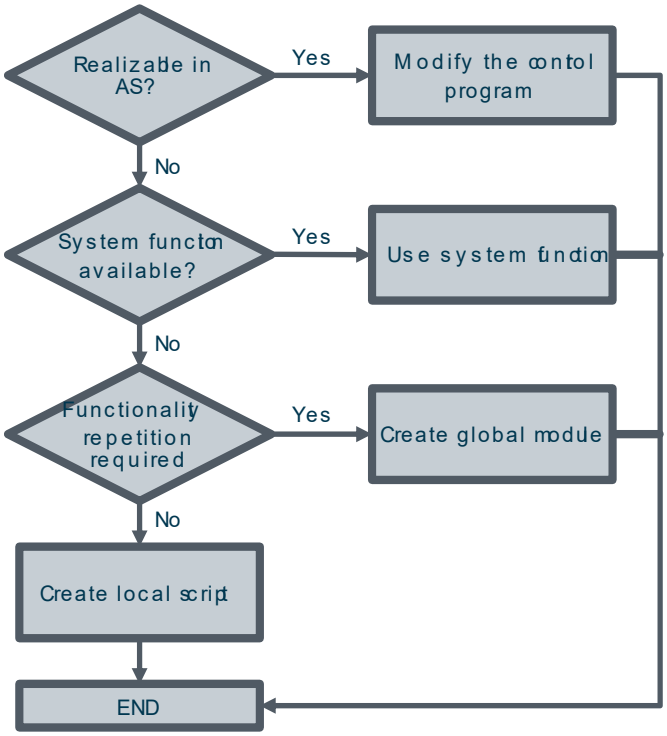
A common example:

If tags are not linked or linked incorrectly, this can lead to scripts not being executed correctly. This has a direct impact on loading times, functionality and performance.

2.1. System Function or Script? (a Decision-Making Aid)

The chart below is a decision-making tool to help you determine when to use a system function, a global module or local scripts.

Figure 2-1



2.2. Trigger Types

General information

There are various triggers to run a script in the runtime. Triggers are conditions. There are three different types of trigger in WinCC Unified:

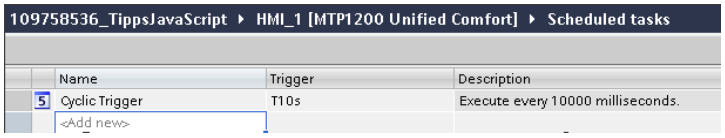
- Cyclic triggers
- Tag triggers
- Event-driven triggers

If no trigger is defined (e.g. in the task scheduler), the script is not executed.

Cyclic triggers

Cyclical triggers are time-driven and are run repeatedly after a certain time, for example every 10 seconds.

Figure 2-2



Note

Please note that the cycle time heavily influences the performance of the project.

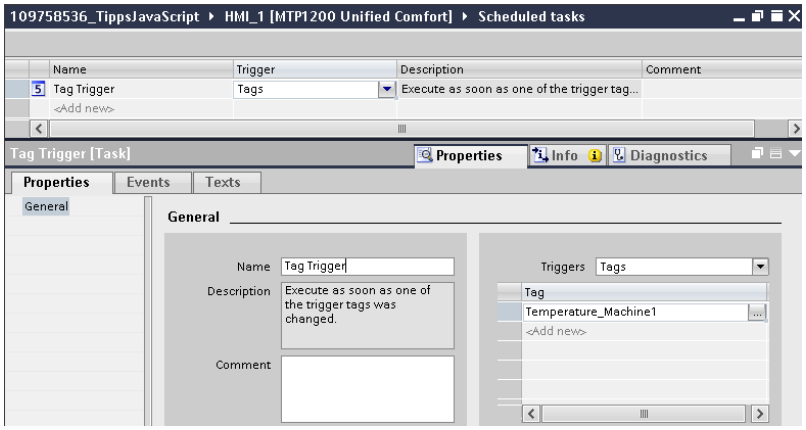
All actions from a screen must be completed within their cycle time. Apart from the runtimes of the actions, the times required for requesting the tag values and the reaction times of the automation systems must also be taken into consideration. You should only set trigger events with a cycle time of less than one second when rapidly changing variables must be queried.

Tag triggers

For a tag trigger, one or more tags must be specified, also known as the trigger tag.

Once the value of the trigger tag changes, the script is triggered and the function inside it is executed.

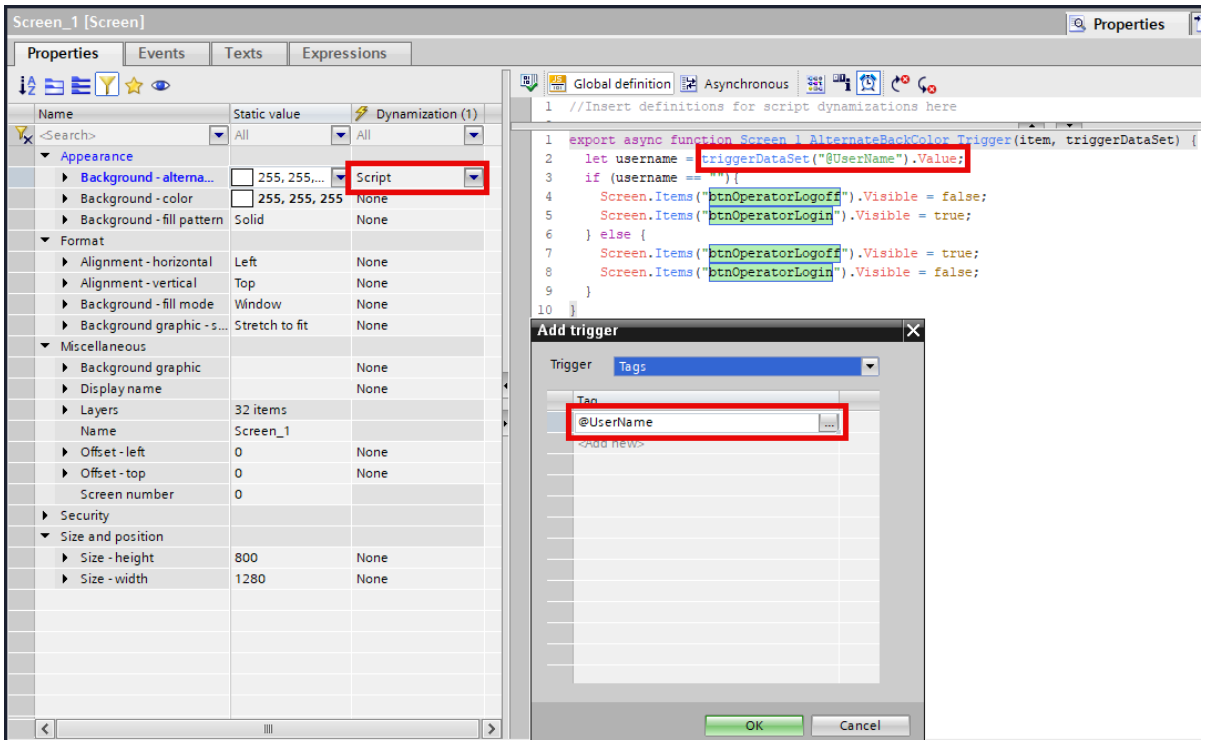
Figure 2-3



With a tag subscription, a value is always read once and in most cases then read again later in the script. The "triggerDataSet" array can be used to read a tag directly from the trigger.

1. Create a script for a property of the screen and manage the property changes centrally with a script. Set the trigger of the script to the tag that should cause the property change.
2. Access the trigger tag value in the script via "triggerDataSet". Note that to use this feature, the device and faceplates device version must be V19 Update 2.

Figure 2-4



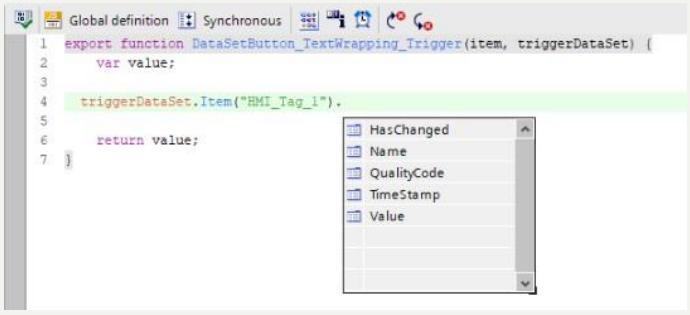
We recommend that you add this script centrally to a property of the screen, rather than to a single screen object. The "Background – Alternative Color" property was used here because it is rarely used for other dynamizations.

Note

The "TriggerTags" parameter contains a collection of all configured trigger tags with the following attributes of the tags:

- .Name
- .HasChanged
- .Value
- .QualityCode
- .TimeStamp

Figure 2-5



You can access the trigger tags and the respective attributes directly via the collection without explicitly reading the tags via ".Read()". If you want to pass user data type elements to the "TriggerTags" parameter, contact Siemens Online Support.

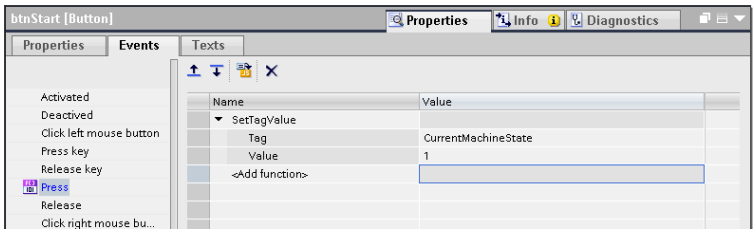
Note

Further information on the "TriggerDataSet" can be found in the Engineering guideline at: https://support.industry.siemens.com/cs/us/en/view/109827603

Event-driven triggers

For event-driven triggers, the script is always run when this event occurs. Events can be, for example, mouse clicks, keyboard operations or changes in focus.

Figure 2-6



2.3. Access Properties of a Screen Object

Using JavaScript, you can address each screen object and modify its properties.

Example

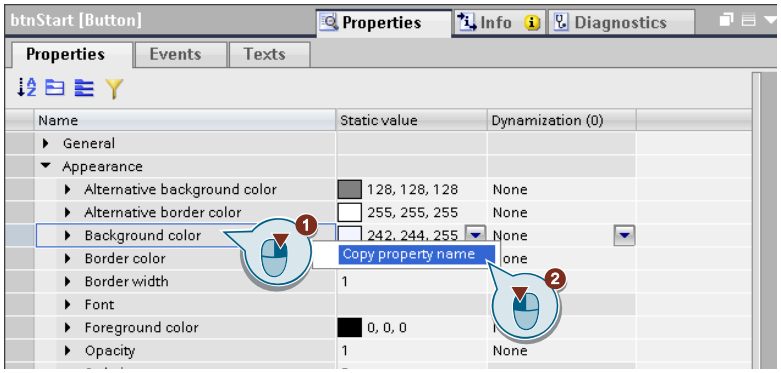
This example changes the background color of the Rectangle object with the object name "Rectangle1" to yellow.

```
Screen.FindItem('Rectangle1').BackColor = 0xFFFFFF00;
```

Find name of the property

You can find the name of the property by right-clicking on the property (1) and then clicking "Copy property name" (2).

Figure 2-7



You can then paste the property name into the desired location with the keyboard command "CTRL + V".

Note

Alternatively, you will find the properties listed under the corresponding object in the manual "IMATIC HMI WinCC Unified V19 – Programming Reference" in the section "HMIRuntime":

https://support.industry.siemens.com/cs/ww/en/view/109896132/150794106635

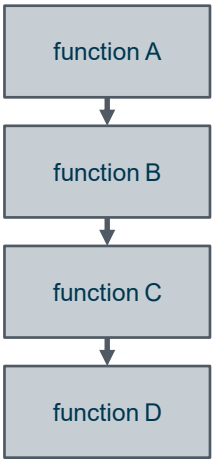
2.4. Difference Between Synchronous and Asynchronous Script Calls

Synchronous/asynchronous script call is a distinction that applied for JavaScript in general.

Synchronous

In synchronous script calls, the functions in the script are executed in order. The next function only begins when the one before it is complete.

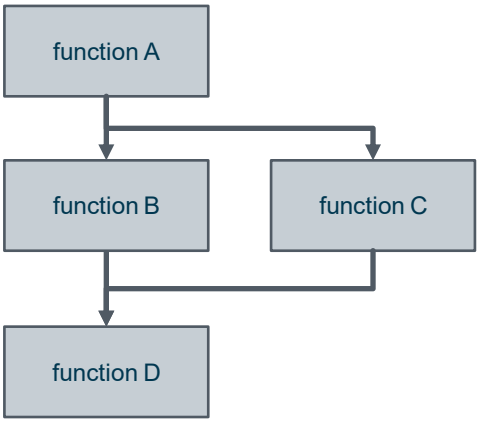
Figure 2-8, Simplified representation of synchronous script flow



Asynchronous

In contrast to the above, there are also asynchronous script calls. In this case, functions can be executed in parallel, thus allowing them to be processed more rapidly.

Figure 2-9, Simplified representation of asynchronous script flow



Typically, asynchronous script calls are used in the context of timers (e.g. "HMIRuntime.Timers.SetTimeout()"), access to network files, or time-intensive database queries.

When using asynchronous script calls, there are however other differences which affect how the script runs. For example, you can use the "await" operator to wait for the result of a function.

If you use multiple complex and therefore more time-intensive functions in a script (for example if you want to read two network files and establish a database connection), then it is recommended to use the Promise.all method.

Note You can find more information about "await" and the Promise object in the "Mozilla Developer Network".

https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Promise/then

Note You can find additional information about "Promise.all" in the "Mozilla Developer Network":

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

2.5. Script Threads

SIMATIC WinCC Unified processes the scripts in Node.js processes. In this context, a Node.js process is always single-threaded. This means that only one CPU core is available to run the code to process the script, meaning that only one script per process can be handled at a time.

In WinCC Unified itself, only two processes are available for script handling:

- one process for all scripts which are run by the user (all scripts in screens)
- and one process for all scripts which are running in the "Task Scheduler".
- Both processes run on two separate CPU cores and can therefore process scripts in parallel.

Note For more information about the script contexts, see the Engineering guidelines at: https://support.industry.siemens.com/cs/us/en/view/109827603

2.6. Tag Types

Tags play a central role in the management of data, and their correct use is crucial for the efficiency and stability of our application. We differentiate between internal and external tags, whereby the internal tags are divided into system-wide and session-local tags. We also look at the use of JavaScript tags and working with datasets. Each of these tag types has its specific areas of application and benefits, which are explained in detail below.

2.6.1. Internal and External Tags

Process values are forwarded in Runtime via tags. Process values are data that are stored in the memory of one of the connected automation systems. They represent the status of a system, for example in the form of temperatures, fill levels or switching states. WinCC works with two kinds of tags:

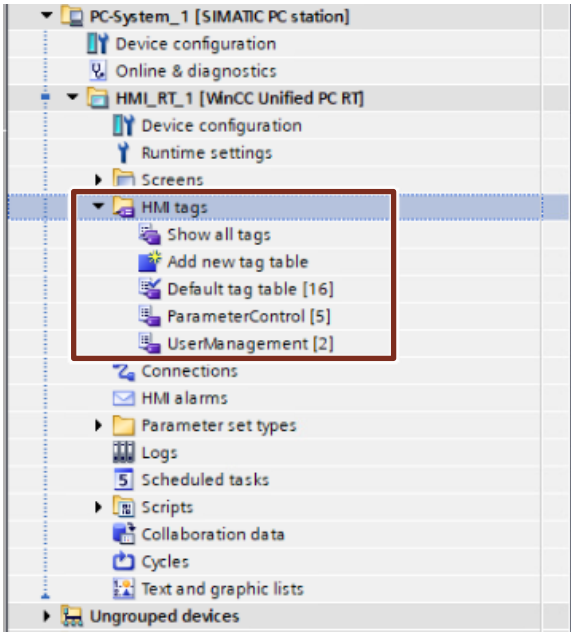
- Internal tags
- External tags

The two types are explained in more detail below.

Internal tags

Internal tags have no process connection and only transmit values within WinCC. The tag values are only available as long as the runtime is running. You can create tags in different tag tables for a better overview. You then access the individual tag tables directly in the "HMI tags" node in the project directory. The "Show all tags" table can be used to display the tags from all tag tables.

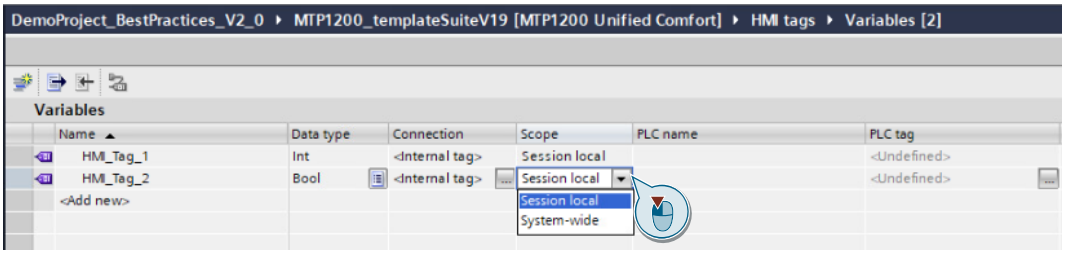
Figure 2-10



Internal tags are stored in the memory of the Operator Panels. Therefore, only this Operator Panel has read and write access to the internal tags. You can create internal tags to perform local calculations, for example. With the internal tags, a further distinction is made between the "system-wide" and "session-local" tags. Internal tags apply "system-wide" by default.

The scope of an internal tag can be changed to "Local session". To do this, the "Scope" column in the tag table must be made visible at the beginning. This option is available by right-clicking the top row of the table.

Figure 2-11



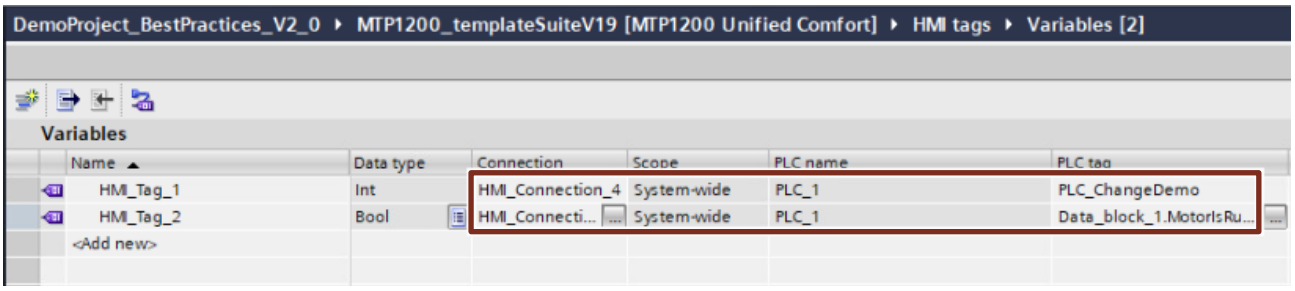
Session-related data is processed independently in each local user session in a multi-user environment. The use of session-local tags is supported in Unified Collaboration and in the Web Client. A typical use case for the session-local tags would be the user-defined parameter control. The tags can be accessed directly via an IO field. The values of a local session tag are not saved and are lost at the end of a session.

External tags

External tags allow data to be exchanged between the components of an automation system, e.g

between an HMI device and a PLC. An external tag is the figure of a defined storage location in the PLC. Read and write access can be set from the PLC. Which PLC tag is mapped is displayed in the tag table under "PLC tag".

Figure 2-12



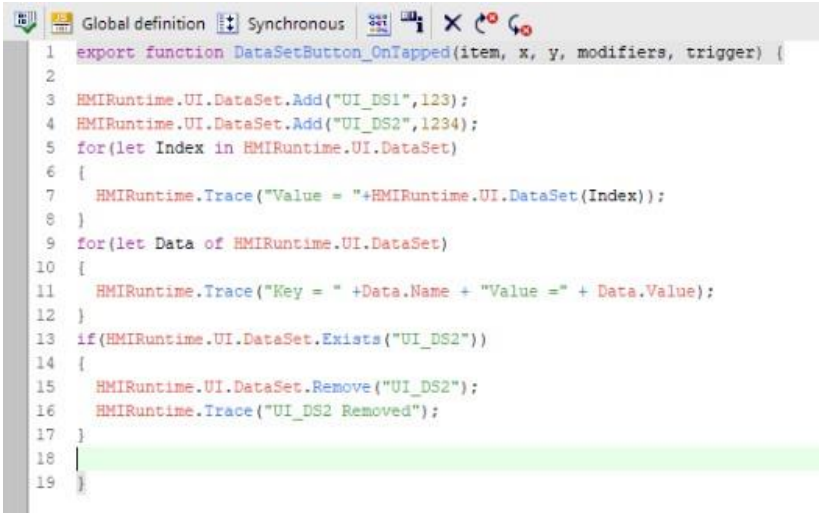
2.6.2. DataSets

The "DataSet" object is a list of objects that enable data exchange in Runtime. A "DataSet" object is defined globally (on the "UI" object) or on a screen (the "Screen" object). A screen-local dataset exists as long as the screen on which it was initialized is displayed. The global DataSet object exists for the entire duration of the runtime.

Session-local DataSet

The session-local DataSet is very similar to the session-local internal tag. This object can be used to save data for a user session, but it is independent of other sessions in a multi-user environment. This type of tag types is mainly used for calculations from the scripting context. Whenever the data does not need to be linked and directly output to an IO field, DataSets are a good choice. The session-local DataSets are available across screens in a browser session in comparison to those that are session-local.

Figure 2-13

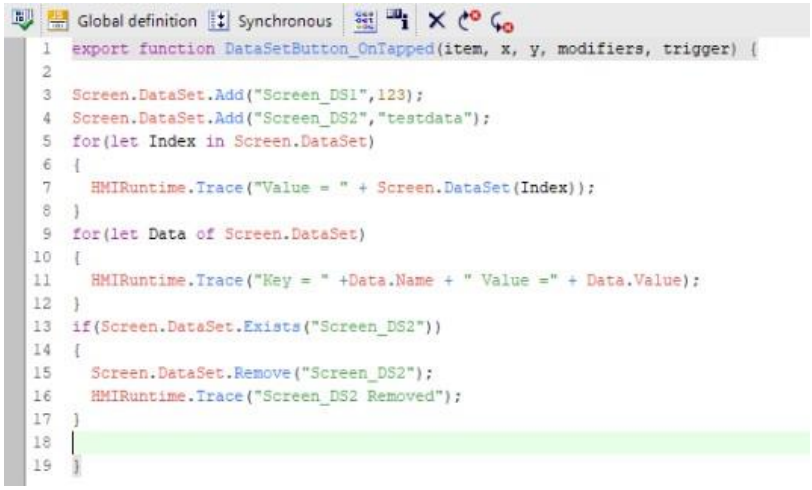


The session-local dataset is defined via the UI object. As can be seen in the screen above, a new DataSet can be created and filled using the "Add" method.

Screen local DataSet

The screen-local DataSet is only valid for a specific screen on which it was created. They are defined via the screen object and are more comparable to the classic JavaScript tags. While JavaScript tags must be defined separately for the event and property areas, the Screen local DataSets can be used across all contexts.

Figure 2-14



Note

The DataSet object has the following limitations:

- Only alphanumeric data types such as Number, String, or Bool can be stored.
- JavaScript objects or classes cannot be saved.
- Triggers for changes are not an option.

2.6.3. JavaScript Tags

JavaScript tags can be used to avoid problems with the internal tag management system. With an asynchronous function call such as the "Write" method, complications can arise if the tag is written and then read again in the same function scope. The subsequent read will normally return the old value, as this has not yet been updated at the time of reading. A JavaScript tag solves this problem. In the following example, the new value of the tag is first saved in a JS tag. This can then be passed via return.

```
const valueToWrite = 55;
Tags('HMI_Tag_1').Write(valueToWrite);
const value = valueToWrite;
return value;
```

JavaScript tags are always subject to the function scope. All tags and functions that belong to the function scope can be used in any program line that is located between the curly brackets of the function command block. All other program parts cannot access these names.

In WinCC Unified, they are also limited to the respective script context in which they are called. For example, they cannot be written in the event area and then read in the properties area.

Figure 2-15

As long as the function is not called, its scope is abstract and does not occupy any memory. JavaScript tags can also be created in the global definition area. The value of tags within a global module can only be changed via its functions. Other places of use (e.g. image properties) can only read these tags.

3. Script Configuration

3.1. Configuring Local Scripts

Supported objects

You can configure scripts to the following points of use in SIMATIC WinCC Unified:

- Screens
- Screen objects
- Tasks

Depending on the object for which you configure the scripts, you can execute different functions.

Table 3-1

Execution context of the editor	Script context and referencing
"Scripts" Editor in the "Screens" Editor	Each process screen has two independent scrip contexts: • Context for dynamizing properties (Section 3.1.1) • Context for evaluating events (Section 3.1.2) Both script contexts of a process screen reference the same global modules. However, each context receives its own copy of the tags defined there.
"Scripts" Editor in the task planner	All tasks share a script context. Different jobs can access common global tags. All tasks reference all global modules of a target system (Section 3.1.3).

3.1.1. Dynamizing object properties via scripts

You can dynamize the relevant properties of screen and screen objects via script, i.e. change the property for the runtime, e.g. change font color, show/hide visibility.

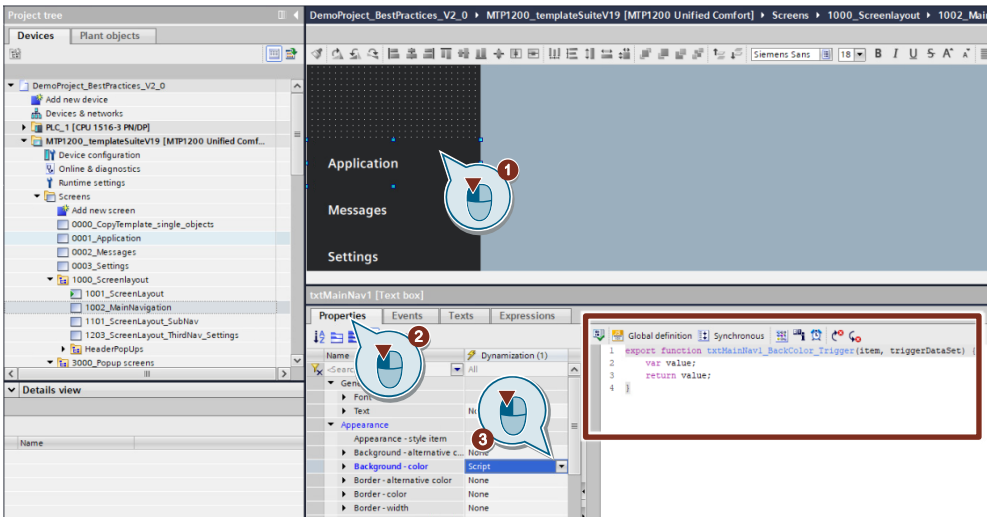
The following steps are required for dynamization by script:

1. Select the screen object on the screen.

1. Open the "Properties > Properties" tab of the screen object.

2. Change the property to be dynamized in the "Dynamization" column to "Script".

Figure 3-1



3. The Script Editor then opens in the Inspector window next to the object properties.

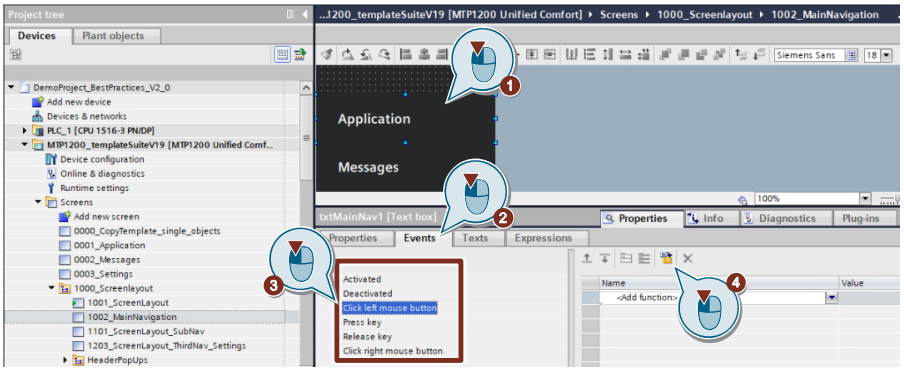
3.1.2. Calling Up Scripts via Events

To execute certain functions in operation, e.g. invert a tag with a button or toggle the language, SIMATIC WinCC Unified provides the option of calling up scripts via events.

The following configuration steps are necessary to call up a script via an event at a screen object (e.g. a button):

4. Select the screen object on the screen.
5. Open the "Properties > Events" tab of the screen object.
6. Select the event that calls the script in the navigation area.
7. Now create a new script with the "Convert to script" button.

Figure 3-2



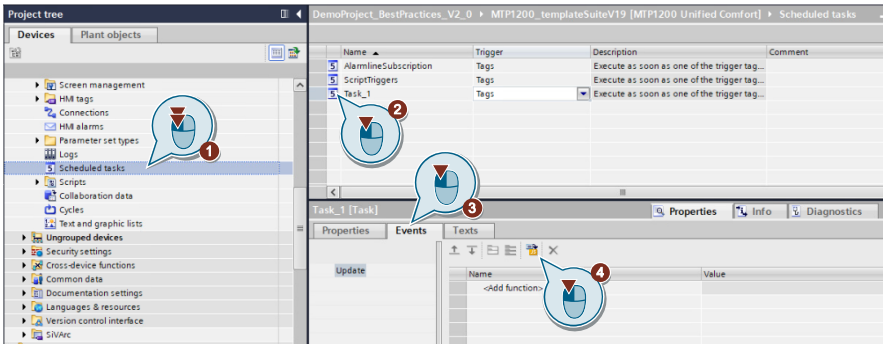
3.1.3. Calling Scripts via Scheduled Tasks

In addition to the "Events" at screen objects, you can also call up scripts in the "Scheduled tasks".

The following steps are required for this:

1. In the project tree, open the "Scheduled tasks".
2. Add a new task or select the existing task.
3. Open "Properties > Events".
4. Now create a new script with the "Convert to script" button.

Figure 3-3



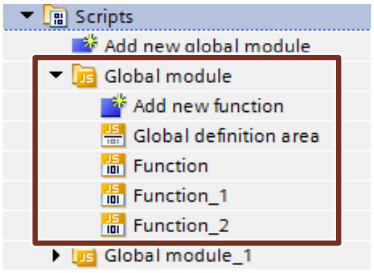
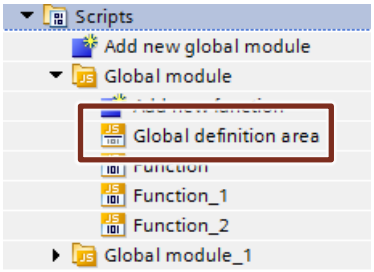
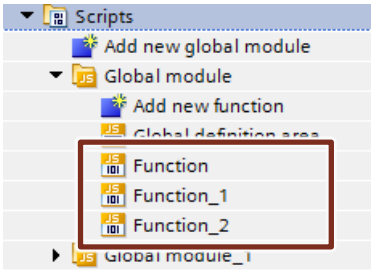
3.2. Configuring Global Script Modules

With the SIMATIC WinCC Unified V16.0 version, you have the option of using global modules in the script context. These are appropriate for:

- Creating functions that are not directly linked to an screen object, an screen, or a task and can therefore be used more than once.
- Triggering scripts cyclically in the background (e.g. task scheduler).

Term overview

Table 3-2

Designation	Description	Screen
Global module	• Global modules are stored in the project tree under "Scripts". • Each global module contains its own definition area and one or more functions. • Global modules are very well suited for grouping functions.	Figure 3-4 
Global definition area	• In the definition area of a global module, you define local tags that you can access in all functions of the global module. • You can also use these tags in local scripts (such as image properties) using the Export/Import command.	Figure 3-5 
Global function	• Each global module can contain several functions. • You can define transfer parameters in a function, which you can then process in the script. • Each function has a return value. • You can also use the Import command to use functions in local scripts (e.g. image properties).	Figure 3-6 

Configuration

If you want to use global modules, the general configuration procedure is as follows:

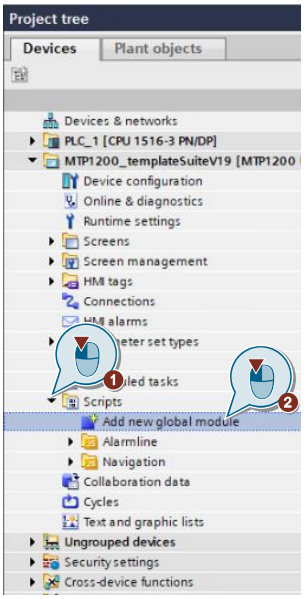
1. Add a global module (Section 3.2.1).
2. Create global definition area (Section 3.2.2).
3. Edit global function (Section 3.2.3).
4. Import and use content from global modules (Section 3.2.4).

3.2.1. Adding a Global Module

To add a global module:

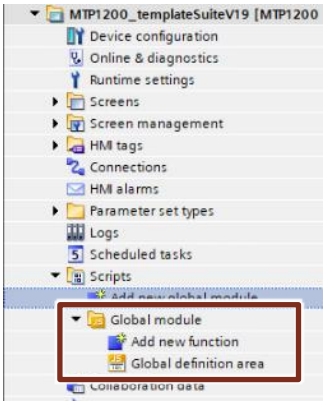
1. Open the "Scripts" folder in the project tree.
2. Click on "Add new global module".

Figure 3-7



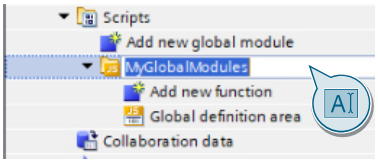
A new folder "Global module" appears in the folder "Scripts". This already contains a "Global definition area".

Figure 3-8



3. If necessary, rename the folder "Global modules", e.g. to "MyGlobalModules".

Figure 3-9



3.2.2. Creating a Global Definition Area

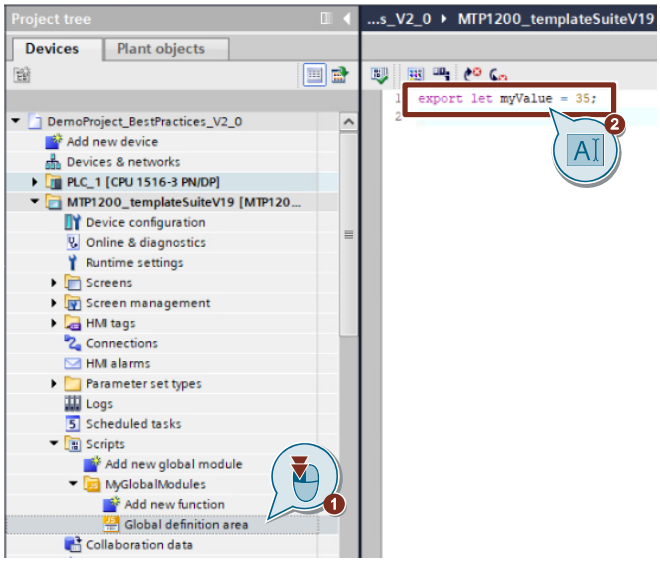
The definition area of a global module is used to access the same tag values in several functions of the same global module.

Configuration

1. Open the "Global definition area" folder by double-clicking on it.
2. Add the code below to the work area.

```
export let myValue1 = 35;
```

Figure 3-10

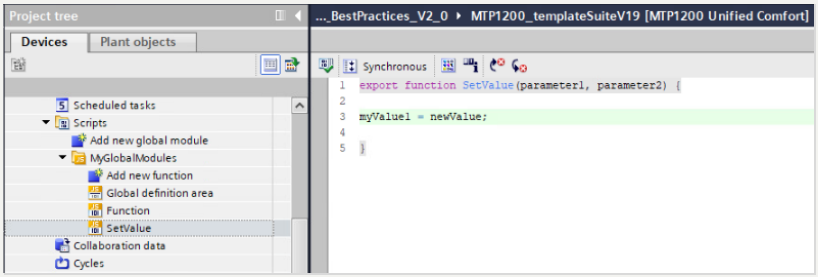


Note

The value of tags within a global module can only be changed via its functions. Other places of use (e.g. image properties) can only read these tags.

If you want to change the value of tags of a global module (here: "myValue1") from a local usage location, you have to call a script of the same global module (here: "SetValue") and enter the desired value via a parameter (here: "newValue").

Figure 3-11



3.2.3. Editing Global Functions

You edit global functions like the global definition area within a global module.

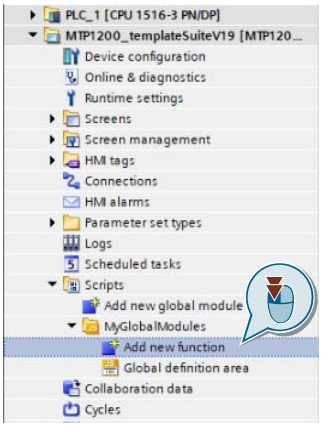
Example

In this example, three values (a value from the global definition area and two transfer parameters) are to be added together in a function. The return value of the function should be the sum ("result") of the three values.

Configuration

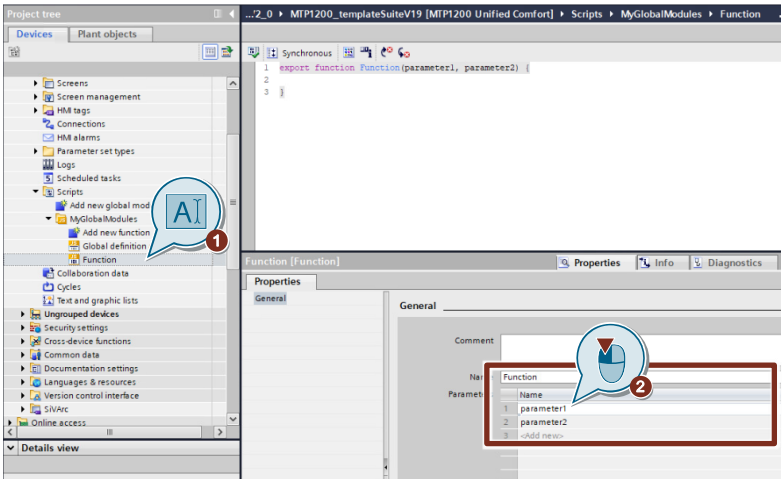
1. Click on "Add new function" in the project tree in your Global Module "MyGlobalModules".

Figure 3-12



2. If necessary, change the name of the function (1) and the number of transfer parameters ("Parameters") in the properties (2).

Figure 3-13



3. Enter the following code in the function:

```
let result;

result = myValue1 + parameter1 + parameter2;
return result;
```

Note The tag "myValue1" has been defined in the global definition area (see Section 3.2.2).

3.2.4. Import and Use Content from Global Modules

To be able to use the tags and functions of the global modules locally, you must first declare them with the import command in the global definition of the local script.

Requirement

In order to describe the import of the global module contents, the following elements were configured in advance:

- a screen ("Screen_Script")
- a button "Button_1" and an I/O field in the screen ("Screen_Script")
- an internal tag "ScriptResult" of the data type "Int".
- the I/O field is connected with the process tag "ScriptResult".

Example 1

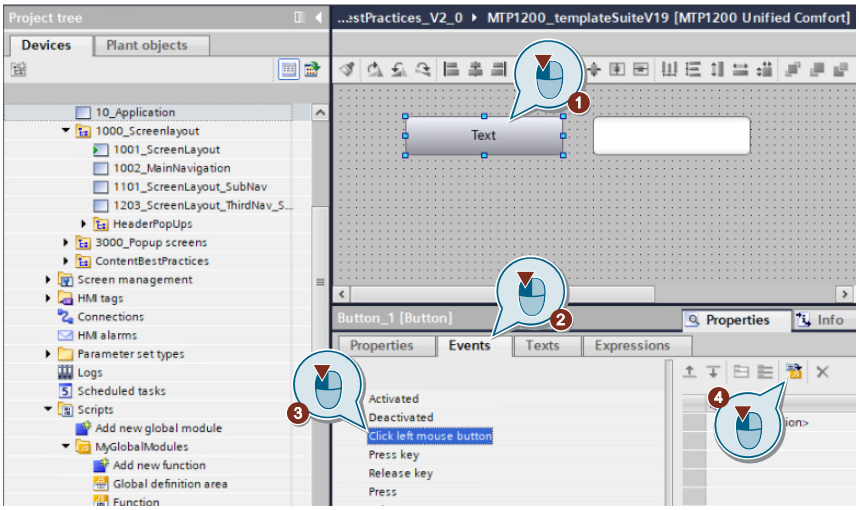
In this example, the value of the tag "myValue1" (as defined in Section 3.2.2), must be written from the global definition area of the global module "MyGlobalModule" into the internal tag "ScriptResult" when the button "Button_1" is pressed.

The value of the internal tag must then be output to the I/O field.

Configuration 1

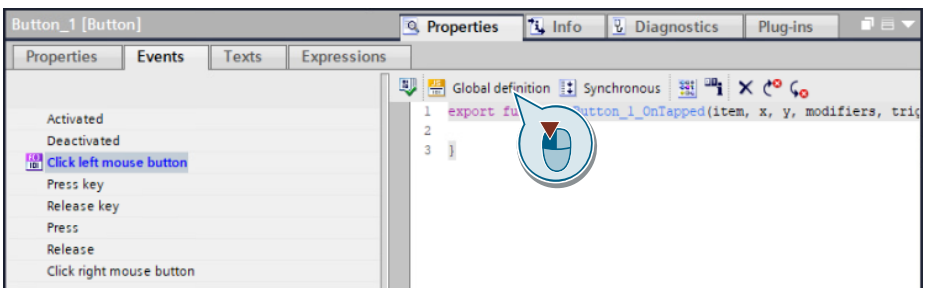
1. Creating a script
   - Select the button (1) in the "Screen_Script" screen.
   - In the properties, open the "Events" tab (2).
   - In the navigation area, select the trigger "Click left mouse button" (3) and then click the button "Convert to script" (4).

Figure 3-14



2. In the local script editor menu, click Global definition.

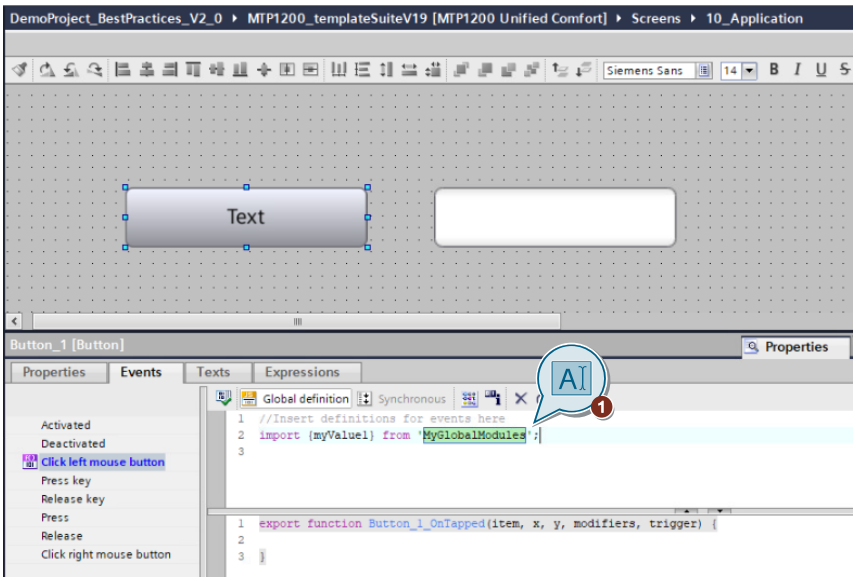
Figure 3-15



3. Import content from global definition area:
   - Insert the following code to import the tag "myValue1" from the global definition area of the global module (1).

```
import {myValue1} from 'MyGlobalModules';
```

Figure 3-16



Note

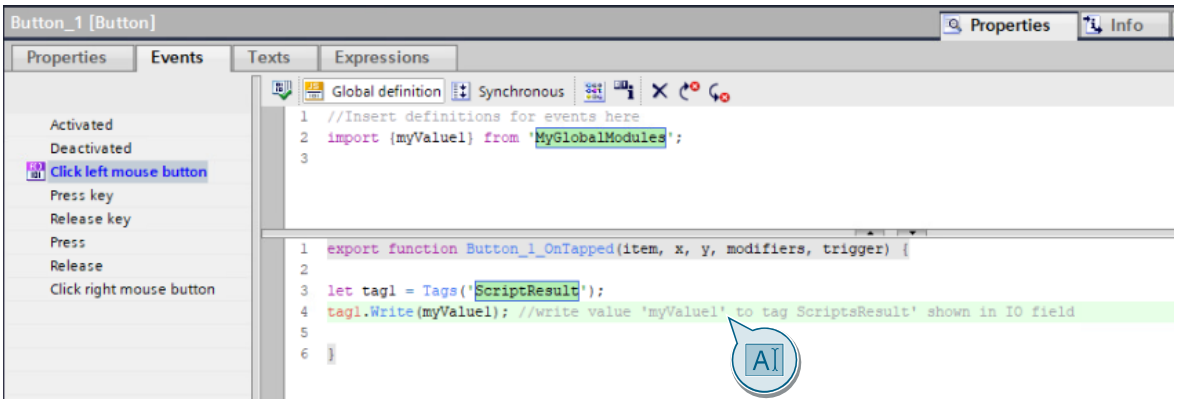
Once you have imported the tag in the global definition area (for example, in the screen properties), it is available in the entire screen properties area of the same screen.

To use the tag in the Events pane of the screen, you must re-import the tag. You therefore have to declare the tag in the global definition area of the events.

4. Insert the following code in the script editor of the event:

```
let tag1 = Tags('ScriptResult');
tag1.Write(myValue1); //write value 'myValue1' to tag 'ScriptResult' shown in IO field
```

Figure 3-17



Note

Optionally, you can also use the script snippet "HMI Runtime > Tag > Write tag" to generate the code in the script editor.

Further information can be found in Section 5.1.

5. Save your project.

Example 2

In the second example, the function "Function" of the global module "MyGlobalModules" is to be executed by pressing the button "Button_1".

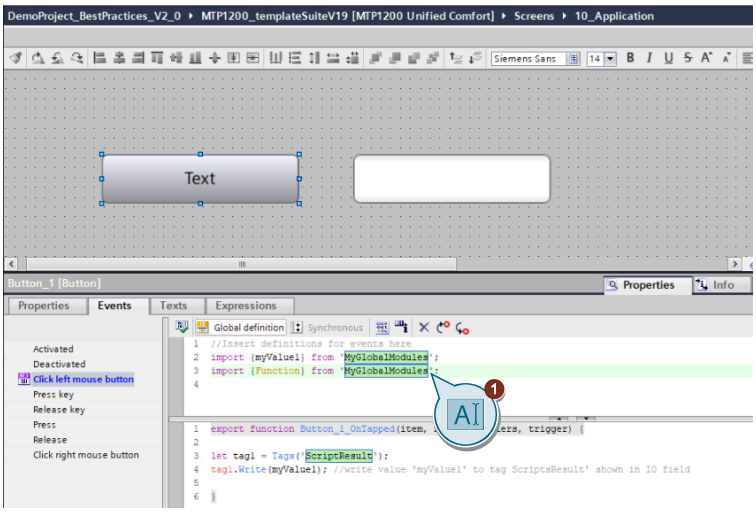
- The values 10 and 12 are to be passed to the function as transfer parameters.
- The return value of the function is then to be output as a trace message.

Configuration 2

1. Open the global definition area of the button "Button_1", as in step 1-3 with "Configuration 1" (page 24).
2. Import function from global module:
   - To additionally import the "Function" function, add the following code (2):

```
import {Function} from 'MyGlobalModules';
```

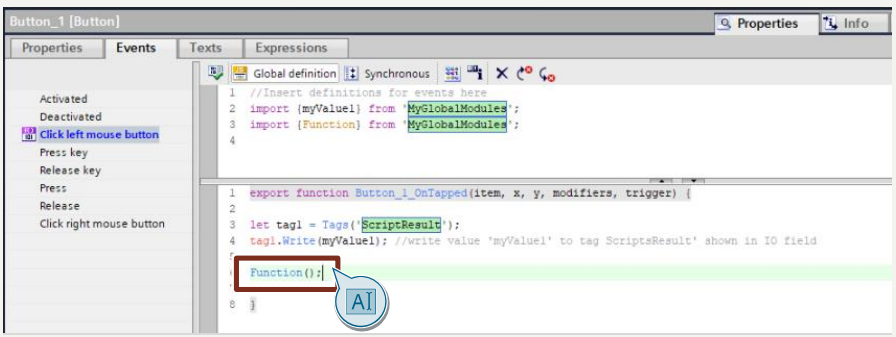
Figure 3-18



Note

If you do not want to pass values to a function of a global module, call the function "Function" with "()" in the workspace of the script editor.

Figure 3-19



Note

You can also import tags from the global definition area and functions from the same global module in one command:

```
import {myValue1; Function} from 'MyGlobalModules';
```

3. Transfer values

   - Define three additional tags in the local script of the button event as shown in the figure (1).

```
let myValueLocal1 = 10;
let myValueLocal2 = 12;
let resultFunction;
```

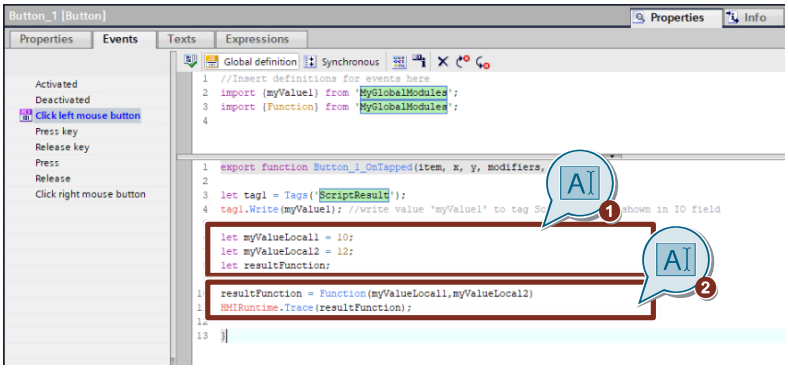
   - Assign the return value of the "Function" function to the "resultFunction" tag and pass the two tags ("myValueLocal1" and "myValueLocal2") (2).

```
resultFunction = Function(myValueLocal1, myValueLocal2);
```

   - Output the value of the tag "resultFunction" as a trace message (2).

```
HMIRuntime.Trace(resultFunction);
```

Figure 3-20



Note

Optionally, you can also use the script snippet "Trace" to generate the code for the trace message.

For additional information on script snippets, please refer to Section 5.1.

Note

If you have a large number of tags in the global definition area or functions in global modules, you can import them collectively with the following line of code:

```
import * as myGMs from 'MyGlobalModules';
```

You can call or use functions or tags from the global definition area of global modules as follows:

```
myGMs.Function();          //call function from global module
myGMs.myValue1;            /*call tag from global definition area in
                             */global module
```

4. Tips and Tricks for Creating Scripts (JavaScript in General)

4.1. Strings in JavaScript

In the following sections (sections 4.1.1-4.1.3), selected examples will show you how to process strings.

Note Further information on working with strings and JavaScript is available at the following link:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

4.1.1. Linking Strings by Script

If you wish to link several strings in a script, this is possible with the concatenation operator "+".

Example:

```
let Tag_01;
const Tag_Text_01 = 'Hello';
const Tag_Text_02 = 'my';
const Tag_Text_03 = 'world';

Tag_01 = Tag_Text_01 + Tag_Text_02 + Tag_Text_03;
// output Tag_01: 'Hellomyworld'
```

Note The Object.prototype.toString() function allows corresponding expressions to be converted to the "String" data type. This conversion before linking the strings allows you to avoid runtime errors.

You can find further information on the "*Object.prototype.toString()*" method at the following link:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString

4.1.2. Adding spaces to linked strings

If you link strings together, it may occur that they are output as one coherent word. This can consequently make the meaning and readability more difficult.

To separate the individual strings from each other, you can add additional separators (e.g. spaces). These are added to the string in quotation marks (" ") and via concatenation operator (+).

Example

```
let Tag_01, Tag_Text_01, Tag_Text_02, Tag_Text_03;
Tag_01 = Tag_Text_01 + " " + Tag_Text_02 + " " + Tag_Text_03;
```

Note The number of "spaces" determines the spaces between the quotation marks " ".

Other options

Alternatively, template strings can also be used. Back ticks (`) are used here instead of double or single quotation marks. The tags are then written with dollar sign and curly brackets "*${Variable}*".

Example


```
let Tag_01, Tag_Text_01, Tag_Text_02, Tag_Text_03;
Tag_01 = `${Tag_Text_01} ${Tag_Text_02} ${Tag_Text_03}`;
```

4.1.3. Determining the Length of a String

You can use the `length` method to determine the length of a sting and further edit it accordingly.

Example

```
const Tag_Text_01 = 'Hello';
HMIRuntime.Trace(Tag_Text_01.length);      //output: 5
```

This can typically be necessary if a string must be further edited which exceeds a determined minimum length.

4.1.4. Finding a Sub-Section of a String

The `substring()` method provides the option of determining and further processing a certain part of a string. Both the beginning and the end position of the partial string are defined in brackets.

Example

```
let Tag_Text_01 = 'WinCCUnified';
HMIRuntime.Trace(Tag_Text_01.substring(0,5));     //output: 'WinCC'
```

As a potential use case, it is conceivable that only strings having a certain prefix would be processed further.

4.1.5. Turning a String into an Array

The `split()` method uses the specified delimiter to split a string into child strings and returns them as an array.

Example

```
let TagText = 'SIMATIC_WinCC_Unified';  //define tag and assign text
let arrayOfString =[];                          //define array

arrayOfString = TagText.split('_');    //split string into array
// Trace output:: 'SIMATIC'
HMIRuntime.Trace(arrayOfString[0]);
```

This method can be applied if you have read a CSV file and then further edit the content of the individual columns separately.

4.2. Arrays in JavaScript

Arrays are a specific kind of data structure. This type of data structure helps when using scripts in SIMATIC WinCC Unified, among other things.

This section will show you some properties and methods which you can apply in connection with WinCC Unified.

Note

For a complete overview of which properties and methods are supported by the array object, please refer to the Mozilla Developer Network at the following link:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

4.2.1. Creating Arrays and Accessing Array Elements

Create array

Create an array in JavaScript by assigning multiple elements to the array name in brackets.

Example:

```
//Create array
let array = ['SIMATIC','WinCC','Unified'];

// TraceViewer output: "Trace Message Array SIMATIC,WinCC,Unified"
HMIRuntime.Trace("Trace Message Array: " + array);
```

Access an array element

If you wish to access a single array element, first enter the name of the arrays followed by the number of the array element that you want to access.

Example:

```
//TraceViewer output: "Trace Message 2. Array-Element: WinCC"
HMIRuntime.Trace("Trace Message 2. Array-Element: " + array[1]);
```

Determine the index of an array element

Using the `indexof()` method you can find which index the element first occurs at. If the element is not in the array, "-1" will be returned.

Example:

```
//Create array
let array = ['SIMATIC','WinCC','Unified'];

//define tag and assign index of WinCC to tag
let index = array.indexOf('WinCC');

// TraceViewer output: "Index of WinCC: 1"
HMIRuntime.Trace("Index of WinCC: " + index);
```

4.2.2. Extending and Truncating Arrays

Extend array

You can add one or more elements to the end of the array by using the `push()` method.

Example:

```
//Create array
let array = ['SIMATIC','WinCC','Unified'];

// Add 'V19' to Array
array.push('V19');

// TraceViewer output: "New Array: SIMATIC,WinCC,Unified,V19"
```

```
HMIRuntime.Trace("New Array: " + array);
```

Truncate array

The `pop()` method is the inverse of the `push()` method. It removes the last element of the array and returns it.

Example:

```
//Create array
let array = ['SIMATIC','WinCC','Unified'];

// TraceViewer output: "Unified"
HMIRuntime.Trace("Return value: " + array.pop());

// TraceViewer output: "New Array after pop(): SIMATIC,WinCC"
HMIRuntime.Trace("New Array after pop(): " + array);
```

4.2.3. Sorting Arrays

You can use the `sort()` method to sort an array. This is the inverse function of the `split()` method.

Example:

```
//Create array
let array = ['SIMATIC','WinCC','Unified'];

//Sort array
array.sort();

// TraceViewer output: "Sorted Array: SIMATIC,Unified,WinCC"
HMIRuntime.Trace("Sorted Array: " + array);
```

4.2.4. Turn Arrays into Strings

The `join()` method is the inverse function to the `split()` method (see Section 4.1.5). It compounds individual array elements into a string and returns this as a value.

Example

```
//Create array and tag 'tagText' as string
let array = ['SIMATIC','WinCC','Unified'];
let tagText = '';

//convert array into string
tagText = array.join('-');

// TraceViewer output: 'SIMATIC-WinCC-Unified'
HMIRuntime.Trace(tagText);
```

This method is useful when you assemble the contents of an array into a string and then export it to CSV.

4.3. Math Object in JavaScript

The "Math" object in JavaScript enables you to perform mathematical operations on numbers. The following shows you how to:

- Add Constants
- Round off tag values
- Find Square Root
- Use Exponent Function
- Generate random numbers
- Determine minimum/maximum values

Note Besides the general JavaScript Math objects there is also a Math object specific to SIMATIC WinCC Unified (see Section 5.14).

4.3.1. Add Constants

You can access various constants in JavaScript via the "Math" object. The "Math" object provides corresponding properties for this.

Example:

In this example, the constant n is added.

```
let tag1 = Tags('HMI_Tag_PI');
tag1.Write(Math.PI);         //Write value '3.141592653589793...'
```

The table below also shows which constants you can access.

Table 4-1

Return value	Syntax
Euler's number	(Math.E)
Square root of "2"	(Math.SQRT2)
Square root of "0.5"	(Math.SQRT1_2)
Natural logarithm of "2"	(Math.LN2)
Natural logarithm of "10"	(Math.LN10)
Logarithm "e" to base "2"	(Math.LOG2E)
Logarithm "e" to base "10"	(Math.LOG10E)

4.3.2. Round Off Tag Values

Depending on requirements, it may be necessary to round off values of a tag. The "Math" objects provides the method `round()` for this.

Example:

```
let tag1 = Tags('HMI_Tag_Round');
let tagValue1 = tag1.Read();        //Read value
tag1.Write(Math.round(tagValue1)); //round value e.g. 4.7 --> 5
```

4.3.3. Find Square Root

The `sqrt()` method the "Math" object finds the square root of a number in JavaScript.

Example:

```
let tag1 = Tags('HMI_Tag_SQRT');
let tagValue1 = tag1.Read();                //Read value e.g. 9
tag1.Write(Math.sqrt(tagValue1));           //square root of "9" --> 3
```

4.3.4. Use Exponent Function

You can also calculate exponential functions in the script with the "Math" object. The `pow()` method is available for this.

Within the brackets you can pass the base and exponent parameters, "*Math.pow(Base, Exponent)*".

Example:

```
let tag1 = Tags('HMI_Tag_Exponent');
let tagValue1 = tag1.Read();        //Read Value1(Exponent) e.g. 3

let tag2 = Tags('HMI_Tag_Base');
let tagValue2 = tag2.Read();        //Read Value2(Base) e.g. 2

tag2.Write(Math.pow(tagValue2, tagValue1));
// Math.pow(x, y) returns the value of x to the power of y:
// e.g. 2³ = 8
```

4.3.5. Generating a Random Number

You can use the `random()` method of the "Math" object to generate random numbers between 0 and 1.

Example:

```
let tag1 = Tags('HMI_Tag_random');
tag1.Write(Math.random());          //write random value to tag12
```

4.3.6. Determine Minimum/Maximum Values

You can also determine the minimum and maximum values from a numerical sequence or multiple tags. You can use the `min()` and `max()` method of the "Math" object for this.

Example:

```
let tag1 = Tags('HMI_Tag_min');
let tag2 = Tags('HMI_Tag_max');

tag1.Write(Math.min(3,5,4,86,2));   //write min-value ("2") to tag1
tag2.Write(Math.max(3,5,4,86,2));   //write max-value ("86") to tag2
```

5. Tips and Tricks for Scripting (WinCC Unified Specific)

5.1. Script Snippets

To make scripting easier for you, SIMATIC WinCC Unified provides the option of adding frequently required code fragments, snippets.

These code snippets can be used to add preformulated, task-specific code fragments, which usually only have to be modified or supplemented slightly.

Example:

```
let tag1 = Tags('MyTag1');
tag1.Write(1234);            //Write value '1234' to tag 'MyTag1'
```

Snippet types

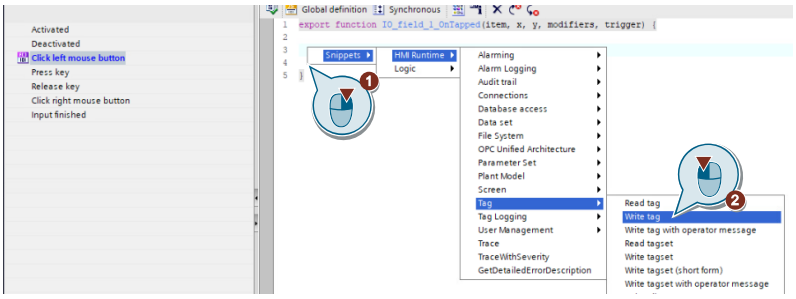
There are two types of snippets in SIMATIC WinCC Unified:

1. "HMI Runtime" - Contains snippets for accessing the object model
2. "Logic" - Contains snippets for branches or loops

Adding snippets

To add a snippet, right-click on the corresponding point in the Script Editor (1) and then select the corresponding snippet from the context menu (2).

Figure 5-1



5.2. Description of the "HMI Runtime" Snippets

The following table gives you an overview of the "HMI Runtime" snippets and their meaning.

Table 5-1

Snippet			Description
HMI Runtime	Alarming	Alarm subscription	Displays a selection of active alarms
		CreateSystemInformation with monolingual Alarm text	Creates an alarm of the message class "SystemInformation" in the message archive with a monolingual message text.
		CreateSystemInformation with monolingual Alarm text and Parameter value	Creates an alarm of the message class "SystemInformation" in the message archive with the alarm text from the specified text list
		CreateSystemInformation with monolingual Alarm text and embedded Text List	Creates an alarm of the message class "SystemInformation" in the message archive with the alarm text and the message parameter from the specified text lists
	Alarm Logging	Read log	Reads message archive
		Export alarm log as CSV	Exports message archive as CSV
	Audit trail	Read electronic records	Reads audit records
		Export electronic record as CSV	Exports audit records as CSV
	Connections	Set connection mode	Sets the connection mode
	Database access	Select Statement	Select database entry
		Create and Insert Statement	Create database entry and output TraceViewer message
	Data set	Screen DataSet	Generates and saves a dataset within a screen which can be accessed from anywhere in the screen. If a screen change is made, the data is deleted.
		Session DataSet	Generates and saves a dataset within a session (browser tab) for the logged-on user. The user can access the data within the session
		Dataset with Database Connection	Creates a database connection and saves it in the session (browser tab)
	File System	Read text file	Reads text file.
		Read binary file (using Int32Array)	Reads the contents of a binary file from the file system (as an array in Int 32 format) [1)]
		Read binary file (using DataView)	Reads the contents of a binary file from the file system. (as DataView) [2)]
		Write text file	Writes text to a new file in the file system.
		Write binary file	Adds binary data to the end of a binary file (*.bin) in the file system
		Append to text file	Adds text to the end of a text file in the file system
		Append to binary file	Appends value to a binary file (*.bin).
		Create directory	Creates a new folder in the file system
		Delete directory	Deletes a folder in the file system with all subfolders and data therein
		Delete file	Deletes a file in the file system
		Write temporary file	Writes a temporary file
		Browse text files recursively in home directory	Searches text files recursively in the home directory
	OPC Unified Architecture	Call a method via OPC UA	Reads or writes tags from the OPC UA Server. Complex function sequences can also be started via OPC UA
	Parameter set	Load Parameter Set from storage and write to PLC	Load parameter set from device storage and write to the PLC.
		Read Parameter Set from PLC and save to storage	Read parameter set from the PLC and store in the device storage.
	Plant model	Read PlantObject properties	Reads property of a plant object.
		Write PlantObject properties	Writes property of a plant object.
		Get parent of PlantObject	Finds higher-level plant object.

Snippet			Description
		Get child of PlantObject	Finds lower-level plant object.
		Get all PlantObjects of specific type	Finds all plant objects of a certain type.
		Get instance screens	Determines the screen name
	Screen	Change base screen	Changes base screen.
		Change screen in screen window of current screen	Changes the screen in screen window which is in the screen being displayed.
		Change screen item property in current screen	Changes property of a screen object which is in the current screen.
		Enable flashing of an item property in current screen	Enables flashing of an object property in the current screen
		Change flashing settings of an item property in current screen	Changes the flashing settings of an object property in the current screen
		Change style of many screen items upon authorization	Changes the style of many screen elements after authorization
		Get names of screen items	Gets screen item names
		Get all alarm properties of the selected alarm from the alarm control	Reads all alarm properties of the selected alarm from the alarm control
		Open faceplate in popup	Opens faceplate as a popup screen.
		Set Language	Sets the language
	Day	Read tag	Reads tag.
		Write tag	Writes tag.
		Write tag with operator message	Writes tag with message to each tag to the user.
		Read tagset	Reads tagset.
		Write tagset	Writes tagset.
		Write tagset (short form)	Writes tagset (short form).
		Write tagset with operator message	Writes tagset with message to the user.
		Subscribe to Tags	The "TagSubscription" object returned enables HMI tags to be monitored for changes
		Linear scaling	Scale tag value linearly.
		Inverse linear scaling	Inverted tag scaling.
	Tag Logging	Read log	Reads tag log.
		Export tag log as CSV	Exports tag archive as CSV.
		Add Comment to log tag	Adds comment to logged tag.
		Correct logged tag values	Corrects logged tag value.
	User management	Check user role	The function checks whether the current user has the selected role
		List all roles	Retrieves the associated roles from the current user
	Trace		Creates TraceViewer messages
	TraceWithSeverity		Used by HMIRuntime's Trace method to indicate severity
	GetDetailedErrorDescribtion		Returns a detailed error description of the error code passed

[1] More information about "Int32Array" can be found at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Int32Array

[2] You can find additional information on the "DataView" view at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView

5.3. Performance-Optimized Configuration

If you want to focus on performance-optimized configuration in your WinCC Unified project, then it is best to observe a few rules.

- Always use the system functions and dialogs provided by TIA Portal before you create a script. The functions that are implemented on the system side are optimized for performance and thus place less of a load on your project's performance.
- Read multiple tags with TagSet If you wish to read multiple tag values in a script, it is recommended to employ the "TagSet" object. You can use the code snippet "HMI Runtime > Tag > Read tagset" for this purpose.

Reading multiple tag values with the TagSet object also increases the script performance and thus places less of a load on your project.

Recommended

```
//Read values via TagSet
let ts = Tags.CreateTagSet(["MyTag1", "MyTag2"]);
ts.Read();
```

Not recommended

```
//Read values single
let tag1 = Tags("MyTag1");
let tagValue1 = tag1.Read();
```

- As far as possible, avoid scripts with cyclic triggers, as this burdens the performance of the entire project. If you nevertheless need cyclic scripts and the use case permits it (e.g. when synchronizing data or for data exchange with databases), the 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.

Note For additional information on "Performance-optimized configuration", please refer to the Engineering guideline at: https://support.industry.siemens.com/cs/us/en/view/109827603

5.4. Screens and Screen Objects

5.4.1. Finding Objects in Screen Windows with Object Paths

Depending on the use case, it may occur that several screen windows are nested within one another. The `FindItem()` method can be used to reference objects within the screen window and change their properties dynamically.

Absolute and relative object paths

In the `FindItem()` method, the object path of the object to be changed is expressed as an argument. This can be specified both relatively and absolutely.

- Relative object path
- The relative object path is indicated based on the screen in which the script is called up. The following table shows the required syntax for relative path specification.

  Table 5-2

Prefixes	Description
".."	References the superordinate screen window (Parent) from the viewpoint of the current screen window
"."	References the own screen window (Self)
""	Without prefix, a screen object of the current screen window is referenced

- Absolute object path
- The absolute object path is indicated based on the "RootScreenWindow". The following table shows the required syntax for absolute path specification.

  Table 5-3

Prefixes	Description
"/"	References a screen window on the highest level, followed by its name Note The "RootScreenWindow" does not have a name in the SIMATIC WinCC Unified V19, therefore two slashes can follow one another.
"~"	References the screen window on the highest level in the own window screen object hierarchy.

Note The object path consists of the name of screen windows (Screen Windows) and screen objects (Screen Items). The names are linked with a slash ("/") according to the hierarchical order. Screens (Screens) and their names are not used in the formulation.

Supported objects

The method is supported by the following objects:

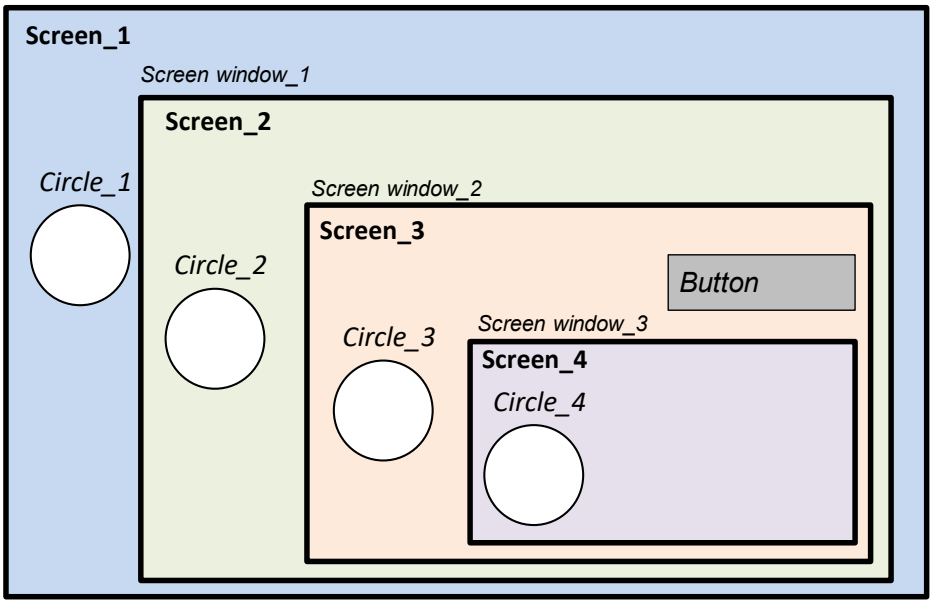
- "UI" (user interface of the graphic runtime system)
- "Screen" (screen in runtime)

Note Only "Window" objects (screen windows) have the property "Path", which returns the absolute path to the "Window" in the total runtime.

Example

In the example, the color of the circle (Circle_1 - Circle_4) over the button "Button" should be changed to yellow. Several screen windows ("Screen windows") are used here, these thus representing a window screen object hierarchy.

Figure 5-2



"FindItem()" with relative object paths

In this example, the color must be changed with the `FindItem()` method and specification of relative object paths.

```
//Change Background Color of 'Circle_3'
Screen.FindItem('Circle_3').BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_2'
Screen.FindItem('../Circle_2').BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_1'
Screen.FindItem('../../Circle_1').BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_4'
Screen.FindItem('./Screen window_3/Circle_4').BackColor = 0xFFFFFF00;
```

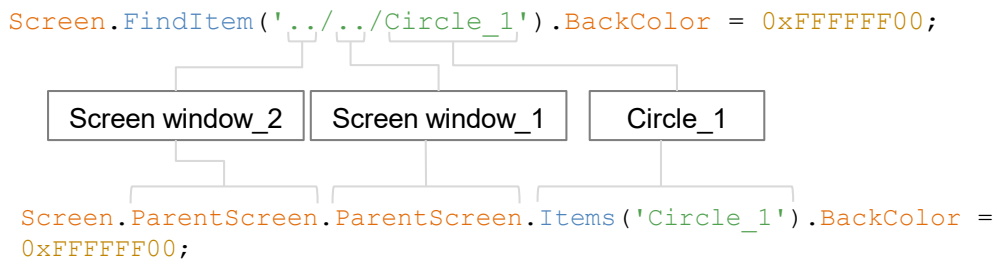
As an alternative to the `FindItem()` method, you can use the "ParentScreen" property to navigate to screens that are located in the screen hierarchy over the screen with the script execution.

```
//Change Background Color of 'Circle_2'
Screen.ParentScreen.Items('Circle_2').BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_1'
Screen.ParentScreen.ParentScreen.Items('Circle_1').BackColor = 0xFFFFFF00;
```

Screens that are located under the screen with the script execution in the window screen object hierarchy cannot be referenced via "ParentScreen".

The relative object path is therefore composed as follows based on the button:

Figure 5-3



Further examples for indicating relative object paths:

Table 5-4

Object Path	Description
'ItemX'	Object ItemX in same screen
'./ItemX'	Object ItemX in same screen
'./ScreenWindow1/ItemY'	Object ItemY from the child screen window "ScreenWindow1"
'../ScreenWindow1/ItemY'	Object ItemY from the neighboring screen window "ScreenWindow1"
'../ItemZ'	Object ItemZ from the superordinate screen window Note Only functions if the script is called up from a screen window.

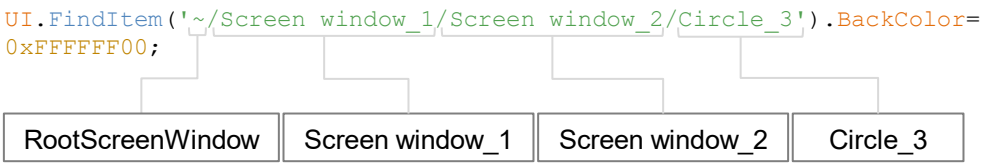
"FindItem()" with absolute object paths

The description below also changes the color of the circles, but the `FindItem()` method is used with absolute object paths.

```
//Change Background Color of 'Circle_3'
UI.FindItem('Circle_3').BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_2'
UI.FindItem('~/Screen window_1/Circle_2').BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_1'
UI.FindItem('~/Screen window_1/Screen window_2/Circle_1'). BackColor = 0xFFFFFF00;
//Change Background Color of 'Circle_4'
UI.FindItem('~/Screen window_1/Screen window_2/Screen window_3/ Circle_4').BackColor =
0xFFFFFF00;
```

The absolute object path is therefore composed as follows based on the button "Button":

Figure 5-4



Further examples for specifying absolute object paths:

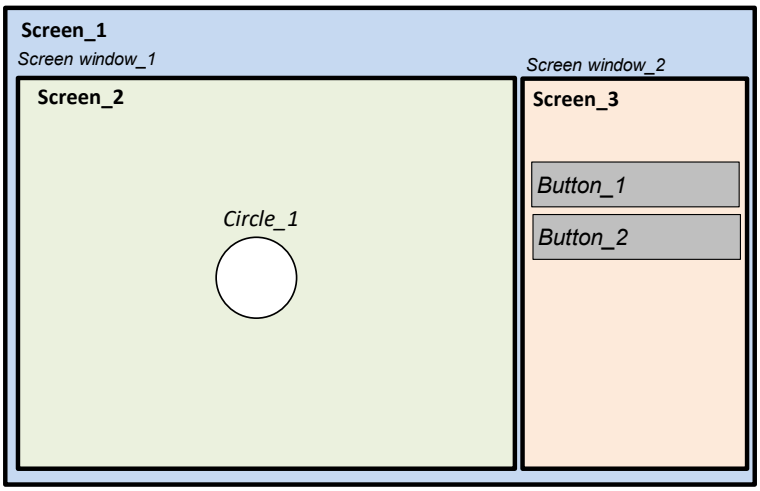
Table 5-5

Syntax	Description
'~/ItemX'	ItemX of the highest screen currently visible ("Screen")
'//ItemX'	ItemX of the highest screen currently visible ("Screen")

5.4.2. Screen Change Across Multiple Screen Windows

Oftentimes screen layouts are used which resemble the figure below.

Figure 5-5



In this case, two screen windows are placed side-by-side in the screen "Screen_1". The screen window "Screen window_2" in turn contains two buttons with the following functions:

- Button "Button_1" should show the screen "Screen_2" in the screen window "Screen window_1".
- Button "Button_2" should change the color of the circle "Circle_1" to yellow in the displayed screen "Screen_2".

Solution with "FindItem()" and relative object paths

Below you will find the JavaScript code for the specified example along with the `FindItem()` method and relative object paths.

```
//Press "Button_1" to change Screen of 'Screen window_1'
Screen.FindItem('../Screen window_1').Screen = 'Screen_2';
// Press "Button_2" to change background color of 'Circle_1' in 'Screen window_1'
Screen.FindItem('../ Screen window_1/Circle_1').BackColor = 0xFFFFFF00;
```

Solution with "FindItem()" and absolute object paths

The following JavaScript code also shows you the solution with the `FindItem()` method, but this time by specifying the absolute object paths.

```
//Press "Button_1" to change Screen of 'Screen window_1'
UI.FindItem('~/Screen window_1').Screen = 'Screen_2';
// Press "Button_2" to change background color of 'Circle_1' in 'Screen window_1'
UI.FindItem('~/ Screen window_1/Circle_1').BackColor = 0xFFFFFF00;
```

5.4.3. Displaying Screens as Pop-Ups

In WinCC Unified, pop-up images are created in the same manner as normal process screens and called up on the screen as "PopupScreenWindow".

This section will show you how to open and close screens using the script system functions "OpenScreenInPopup" and "ClosePopup". Please refer to Section 5.5 on how to open faceplates as pop-ups.

Note

For further information, refer to the "System manual – SIMATIC WinCC WinCC Engineering V19 – Runtime Unified" at:

"OpenScreenInPopup" system functions
https://support.industry.siemens.com/cs/ww/en/view/109896132/170652811403

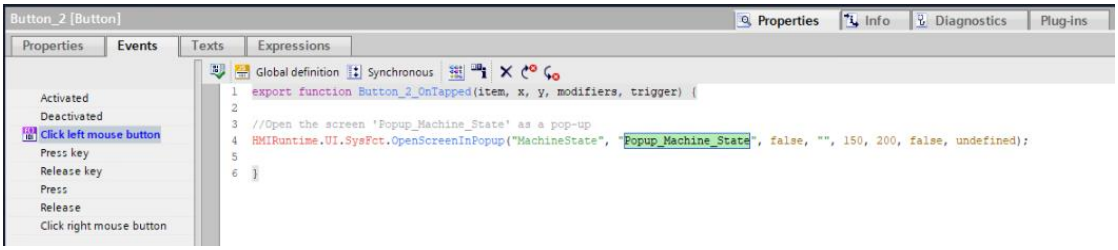
"ClosePopup" system functions:
https://support.industry.siemens.com/cs/ww/en/view/109896132/159281950347

Open screen as pop-up with a script

In order to open the process screen "Popup_Machine_State" as a pop-up via a script, enter the following JavaScript code:

```
//Open the screen 'Popup_Machine_State' as a pop-up
HMIRuntime.UI.SysFct.OpenScreenInPopup("MachineState", "Popup_Machine_State", false, "", 150,
200, false, undefined);
```

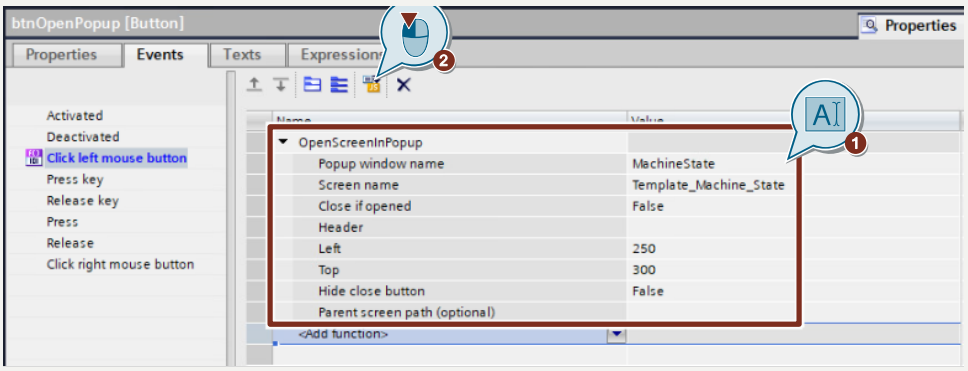
Figure 5-6



Note

Optionally, you can also use the system function "OpenScreenInPopup" and then convert it to JavaScript.

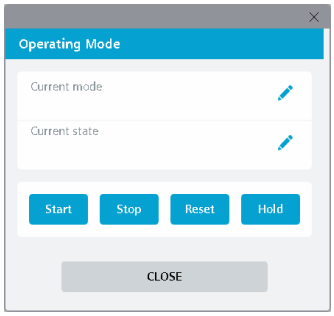
Figure 5-7



Showing pop-up screen in runtime

The screen appears as a pop-up in the runtime. By default, each "PopupScreenWindow" has a gray border and a header.

Figure 5-8, Pop-up in the runtime

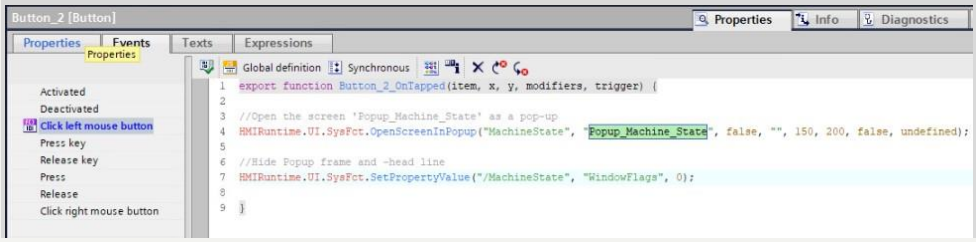


Note

If you want to show the "PopupScreenWindow" without a frame or header, you must set the properties of "Window settings" to "0" using the system function "SetPropertyValue".

```
HMIRuntime.UI.SysFct.SetPropertyValue("/MachineState", "WindowFlags", 0);
```

Figure 5-9



The parameter for "Popup window path" is composed of the "/" (reference to the user's window) and the name of the pop-up window that you specify when opening the pop-up with "OpenScreenInPopup" (see also Section 5.4.1).

Note

For further information on "Pop-up screen window" and the "SetPropertyValue" system function, refer to the "System manual – SIMATIC WinCC WinCC Engineering V19 – Runtime Unified" at:

"SetPropertyValue" system functions:
https://support.industry.siemens.com/cs/ww/en/view/109896132/159282054539

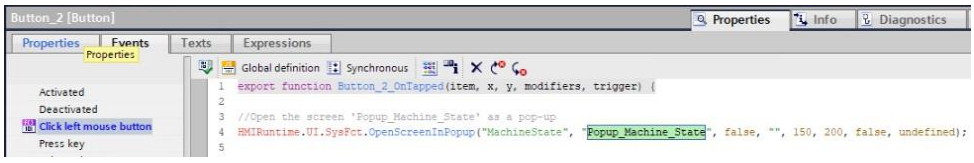
Close pop-up externally

In order to close the pop-up via script (i.e. via a button that is not part of the pop-up), you can use the system function "OpenScreenInPopup".

However, you must set the function's "toggleOpen" parameter to "True".

```
//Close the popup if it´s open
HMIRuntime.UI.SysFct.OpenScreenInPopup("MachineState", "Popup_Machine_State", true, "", 150, 200,
false, undefined);
```

Figure 5-10



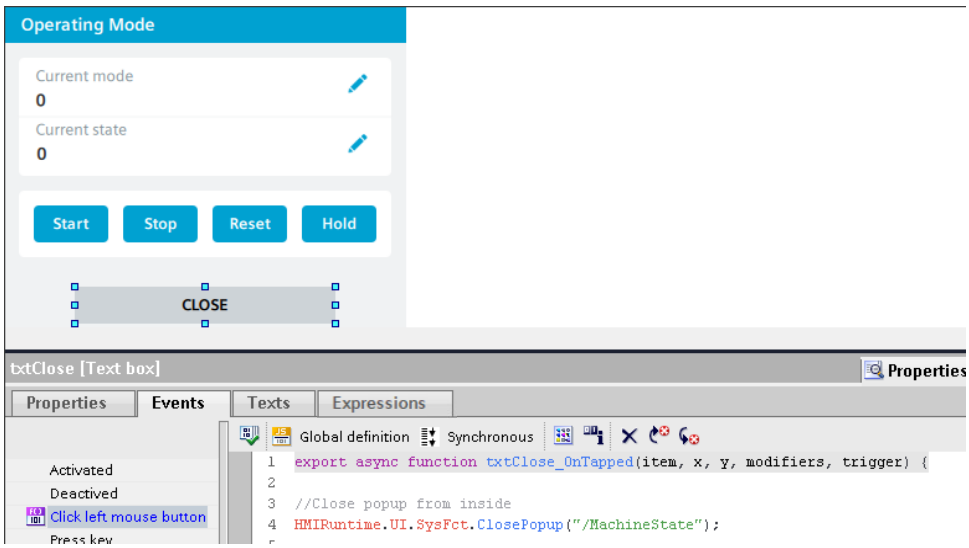
Close pop-up internally

If the pop-up is open and you want to close it via a configured button (in the figure, the "CLOSE" button), you can implement this with the system function "ClosePopup".

For the parameter, specify the path of the pop-up window.

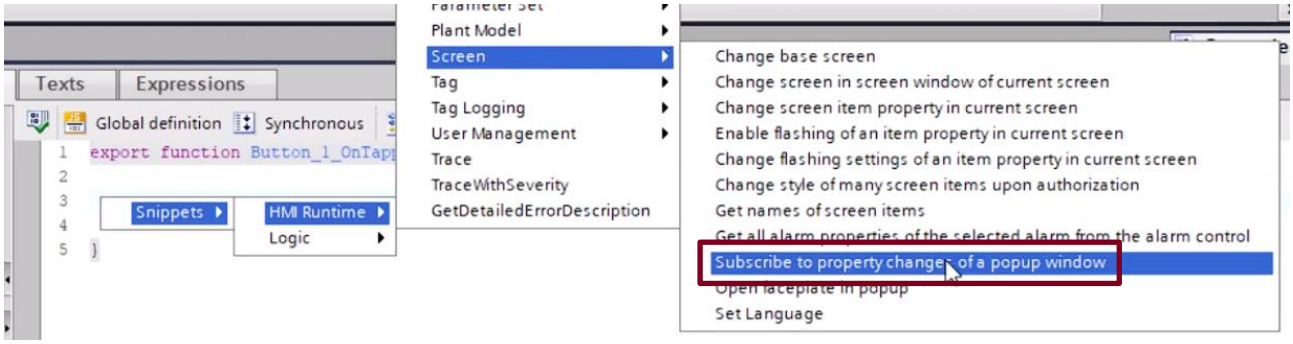
```
//Close popup from inside
HMIRuntime.UI.SysFct.ClosePopup("/MachineState");
```

Figure 5-11



5.4.4. Subscribing to Screen Pop-Up Events

Respond to pop-up events. For example, changes to the positioning or zoom level of the pop-up. A new code snippet is available for this.



The code snippet has the following functions:

1. It creates a subscription that monitors changes to properties in the user interface. If a property of a monitored element changes, a log entry is created containing the name of the element, the changed property and its new value.
2. A pop-up window with the name "PopupWindow" opens, in which any screen is displayed. The pop-up window has a title and a specific position. The changes configured in this popup can be read from the RTIL TraceViewer.

```
// create subscription
let sub = HMIRuntime.UI.Events.CreateSubscription();
sub.OnPropertyChanged = (Item, PropertyName, Value) =>
{
  HMIRuntime.Trace(`Property changed: '${PropertyName}' Item: '${Item.Name}' Value: '${Value}'`);
};
sub.Start();
```

```
// Open a popup window
HMIRuntime.UI.SysFct.OpenScreenInPopup("PopupWindow", "Screen_2", true, "CaptionText", 0, 0,
false, "./Screen");
```

5.4.5. Determining the Screen Name

Determining the Screen Name

To determine the name of a screen, you can output this with the following code line as a message in TraceViewer.

```
HMIRuntime.Trace(Screen.Name);
```

Consecutively numbered screen names

If you name the screens correspondingly, e.g. Screen_01, Screen_02, you can extract the number from the screen name using the `split()` method.

```
// screen name = "Screen_01"
HMIRuntime.Trace(Screen.Name.split('_')[1]);
```

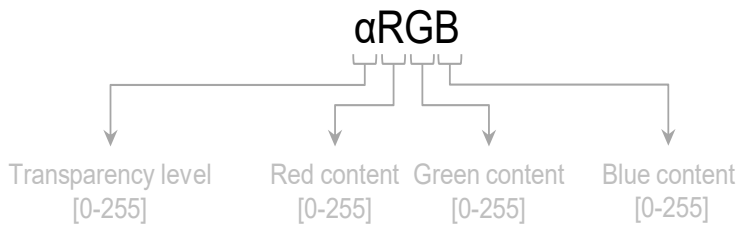
For a screen name "Screen_01" in TraceViewer, "01" is output as a message.

5.4.6. Change Colors

SIMATIC WinCC Unified lets you change colors (e.g. background color, text color, frame color) of objects via JavaScript.

The color that you assigned to the object is made up as follows:

Figure 5-12



Both the transparency value as well as the RGB value are indicated as a separate value in hexadecimal form in the script.

```
//αRGB – α opacity[0-255] Red[0-255] Green[0-255] Blue[0-255];
Screen.Items('Rectangle1').BackColor = 0xFF00FF00;
```

Note

If you want to specify the RGB value instead of the hexadecimal value, you can do this with the "RGB" method of the WinCC Unified "Math" object.

```
HMIRuntime.Math.RGB(0,255,0);
```

Note

If you specify less than eight digits instead of the color code, the missing digits are treated as "0" and added at the beginning.

e.g.: 0xFF00FF → 0x00FF00FF

The can lead to the color being displayed as transparent, as the transparency level is "00".

5.4.7. Counting Screen Objects and Finding Screen Object Names

In this section you will find code examples for how to count screen objects and output their names.

Counting screen objects

The following example will output the number of screen objects in the TraceViewer which are configured in the current screen.

Note

If a screen object has been switched to invisible, it will still be counted as a configured screen object.

Example

```
// amount of all screen items on the screen:
const items = Screen.Items;
HMIRuntime.Trace(items.Count);
```

Outputting names of the screen objects

The example below outputs each screen object name in a separate trace message in the TraceViewer.

Example

```
// names of all screen items on the screen:
for(let i in Screen.Items) {
  HMIRuntime.Trace(Screen.Items(Number(i)).Name);
}
```

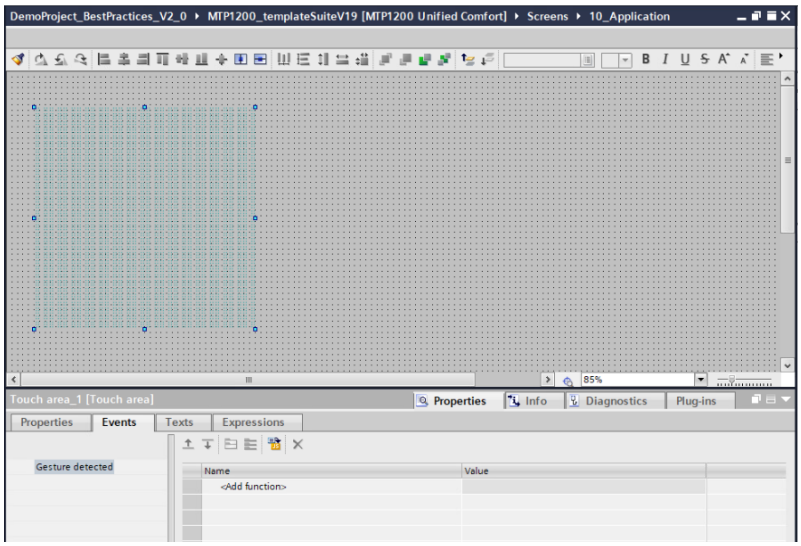
5.4.8. Reading Out Touch Area Direction

The screen object "Touch area" recognizes swipe gestures in the runtime. Therefore, if you swipe over the configured area in the runtime, it will be detected and you can execute an action.

Direction-independent action (default setting)

By default, the screen object has the event "Gesture detected". In this setting, any arbitrary direction is detected and the configured action will be executed.

Figure 5-13

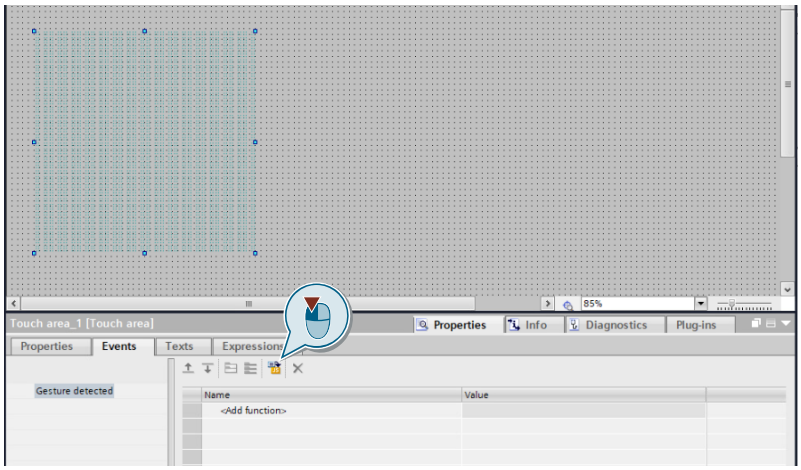


Direction-dependent action

However, if you want different actions to be run depending on the direction of the swipe, you can implement this using JavaScript.

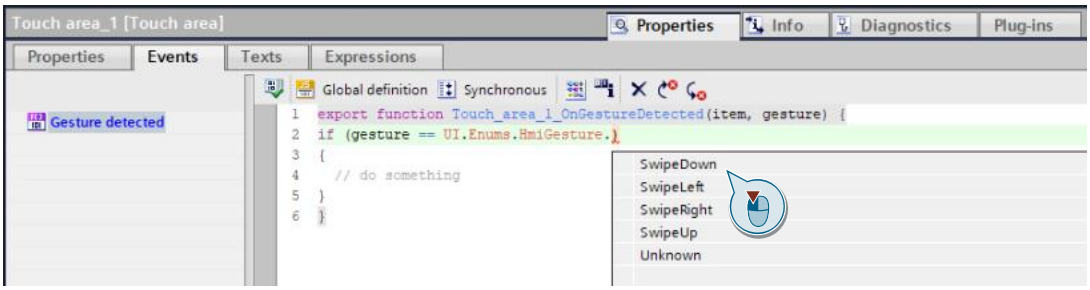
1. Click on the button to go to the "Script editor".

Figure 5-14



2. Select direction
   - Press the key combination "CTRL + space bar".
   - Then select the direction from the selection list that is to be recognized as a swipe gesture and the action is to be executed.

Figure 5-15



Note As an alternative, you can also select the direction of the gesture via the object model.

e.g., UI.Enums.HmiGesture.SwipRight;

Note For each touch area, you can add more If commands and thereby define actions for all the other directions.

5.5. Interconnecting Faceplates via Scripts

Faceplates can be configured centrally in TIA Portal; you can later re-use them with various process tags.

Overview

This section describes how you can:

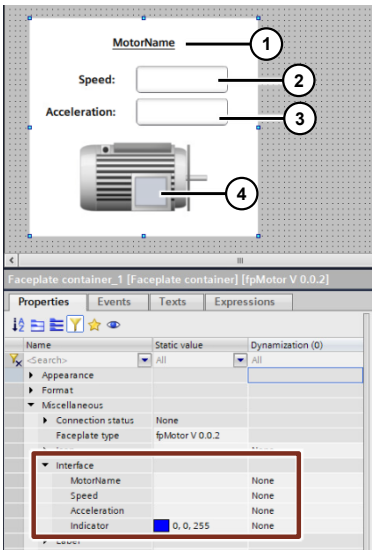
- Open a faceplate as a pop-up and interconnect via script → see Section 0
- Configure a faceplate in the screen and modify the interconnection in the runtime → see Section 5.5.2
- Open a faceplate from a faceplate → see Section 5.5.3

Example

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

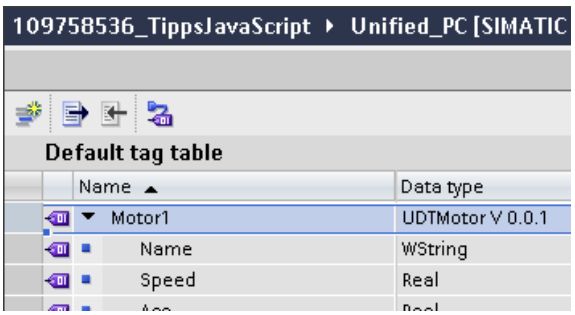
- three tag interfaces ("MotorName" (1), "Speed" (2) and "Acceleration" (3)) and
- a properties interface ("Indicator" (4)).

Figure 5-16



This faceplate should be called up as a pop-up window and connected to the UDT "UDTMotor".

Figure 5-17, "UDTMotor"



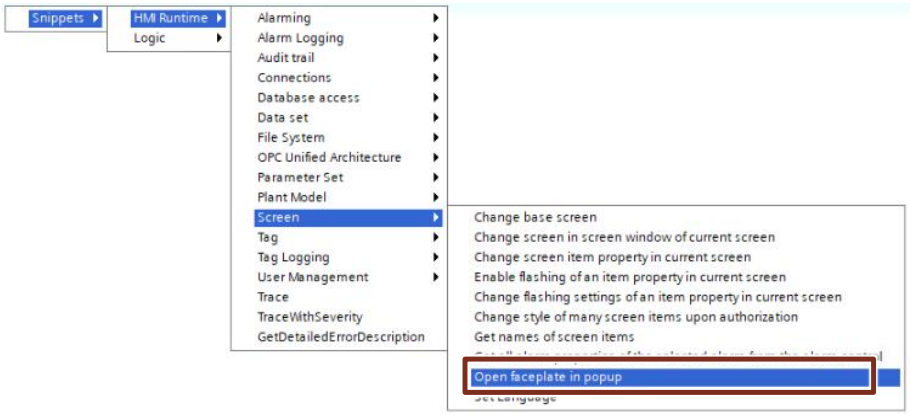
5.5.1. Opening Faceplates as a Pop-Up

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

Code snippet

The snippet "Open faceplate in popup" will help you call the faceplates.

Figure 5-18

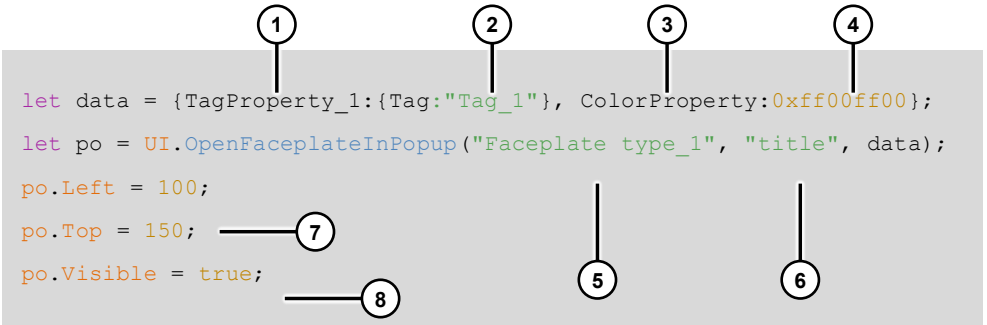



```
let data = {TagProperty_1:{Tag:"Tag_1"}, ColorProperty:0xff00ff00};
let po = UI.OpenFaceplateInPopup("Faceplate type_1", "title", data);
po.Left = 100;
po.Top = 150;
po.Visible = true;
```

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

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

Figure 5-19



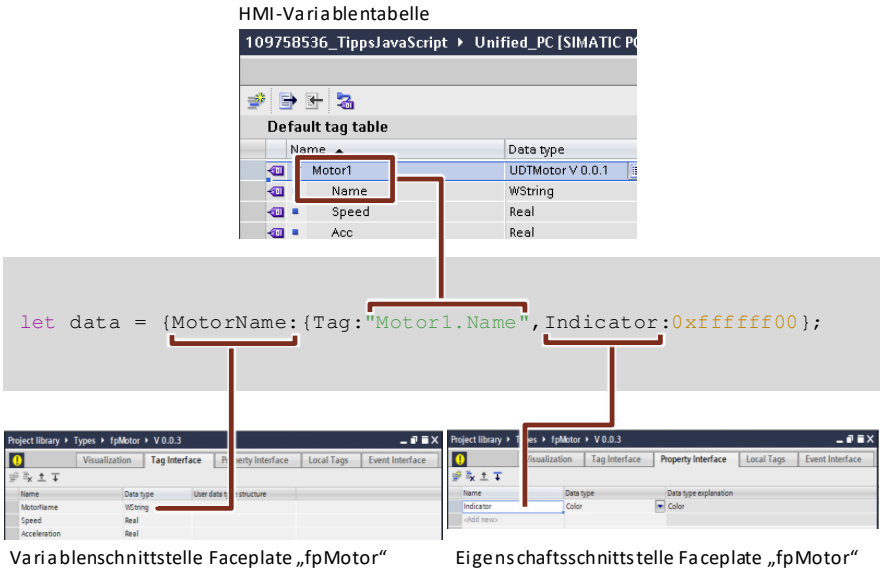
Note

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

https://support.industry.siemens.com/cs/ww/en/view/109896132/170651517835?dl=de

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

Figure 5-20



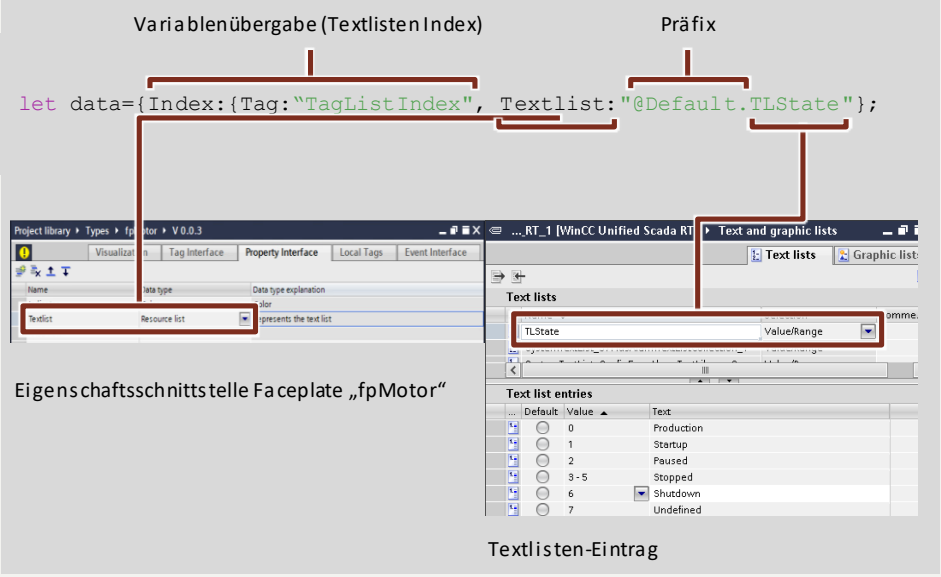
Note

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

The transfer parameter for a text list itself is made up of the following two specifications:

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

Figure 5-21



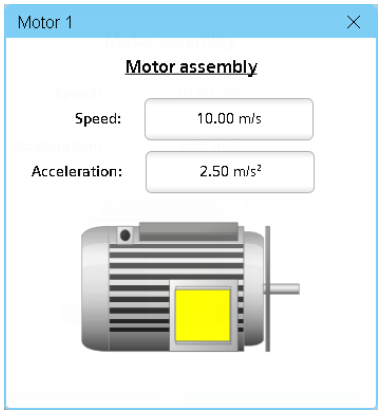
Result

The following overview shows the fully configured code snippet.

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

After the button has been pressed in Runtime, the faceplate appears as a pop-up window with the transferred data.

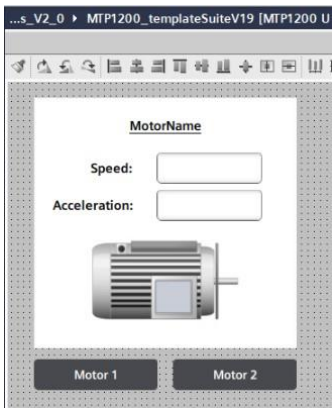
Figure 5-22



5.5.2. Modifying Faceplate Interconnection in the Screen

In the second use case, the intent is to configure the faceplate "fpMotor" in the screen. Using two buttons, the interface in the runtime must be switched from UDT Motor 1 to UDT Motor 2.

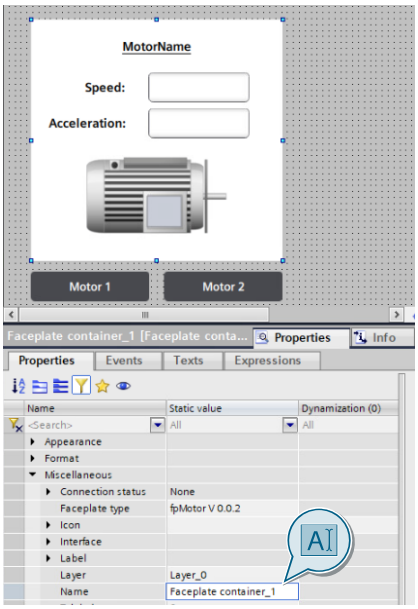
Figure 5-23



Configuration

1. To do this, first place the faceplate "fpMotor" in the screen by dragging and dropping.
2. Name the automatically generated faceplate container as "fpContainer".

Figure 5-24



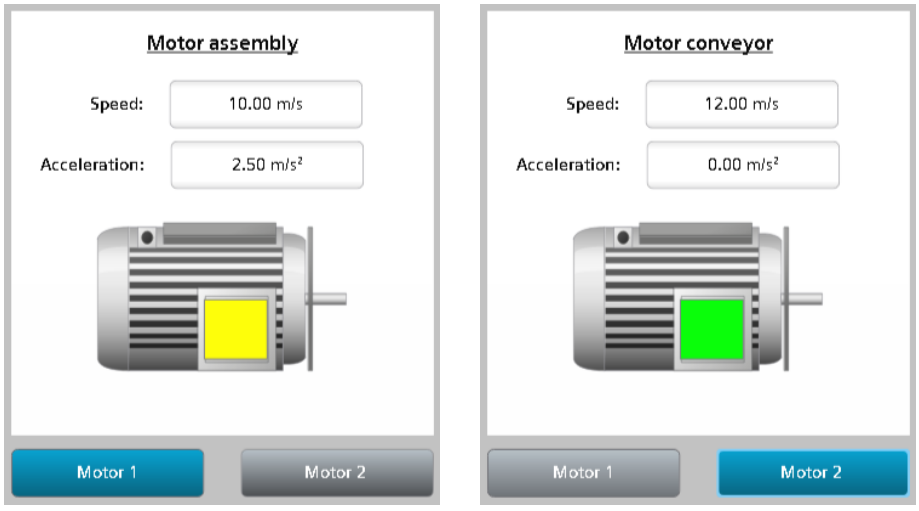
3. Add the button for Motor 1 with the following script.

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

4. Then repeat step 3 on the button for motor 2 and replace "Motor1" with "Motor2".

Result in Runtime

Figure 5-25



5.5.3. Open a Faceplate from a Faceplate

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

Code snippet

To achieve this, you also have the option within a faceplate of opening a second faceplate by using the snippet "Open faceplate in popup".

Figure 5-26


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

Note If you launch or open a second faceplate (e.g. "Faceplate_2") from the current faceplate (e.g. "Faceplate_1"), the linked interface tags will be passed automatically to the faceplate that is being called ("Faceplate_2"). No additional configuration steps are necessary.

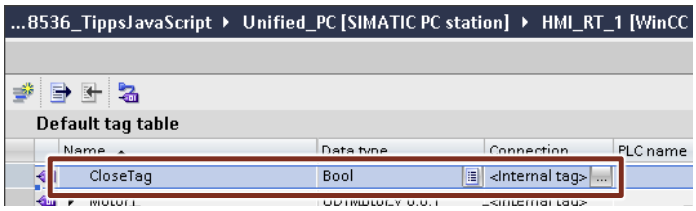
5.5.4. Closing a Faceplate

This section will show you how you can close a faceplate by pressing a button in the faceplate.

Add HMI tag

In the first step, add an HMI tag "CloseTag" of data type "Bool".

Figure 5-27

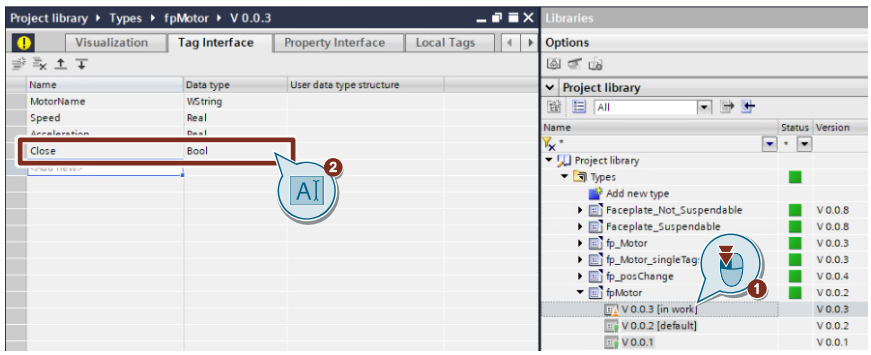


Note If you have multiple faceplates that you wish to close in this way, you can also create an "Array of Bool" with the number of faceplates that you want to close.

Modify faceplate

1. Open the faceplate that you wish to close with a configured button, in this example, "fpMotor".
2. In the tag interface, add the tag "Close" of data type "Bool".

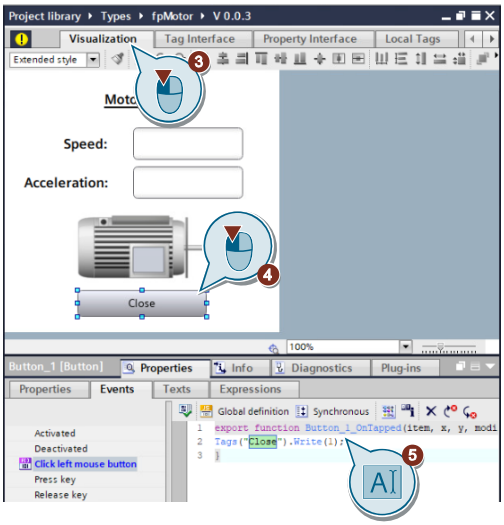
Figure 5-28



3. Then switch to the visualization of the faceplate.
4. Configure a button.
5. with the following code on the "Click left mouse button" event.

```
Tags("Close").Write(1);
```

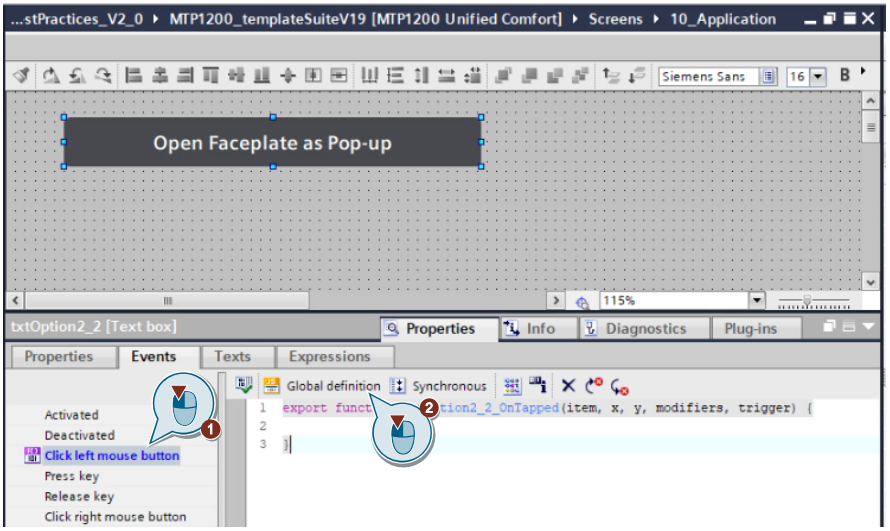
Figure 5-29



Modify faceplate call at the button

1. Select the button via which the "fpMotor" faceplate should be opened and add a script to the "click left mouse button" event. Then open the global definition area.

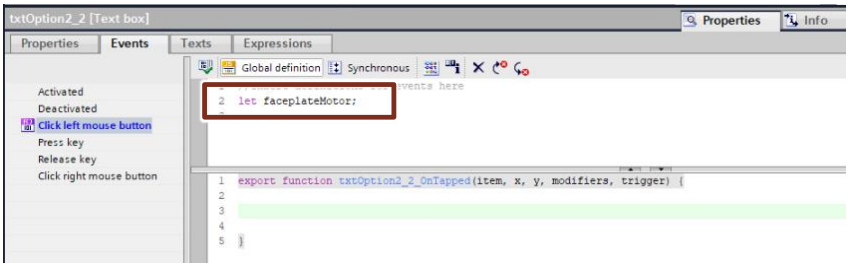
Figure 5-30



2. Add the following code to the definition area and then switch back to the script function.

```
let faceplateMotor;
```

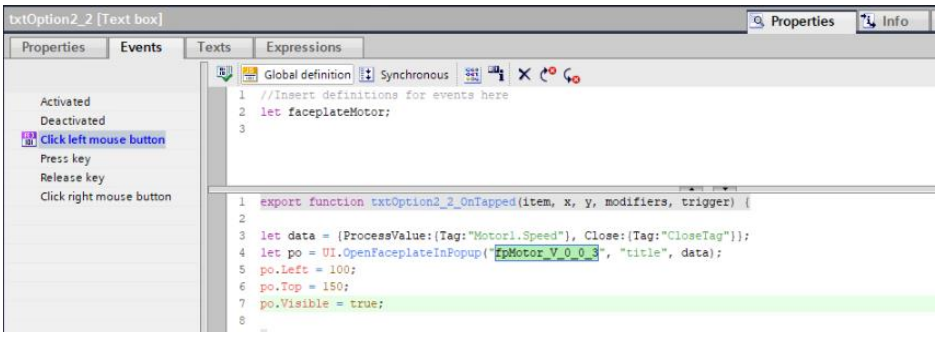
Figure 5-31



The reference of the pop-up window is stored in the global definition area via this step and you can access it via another button.

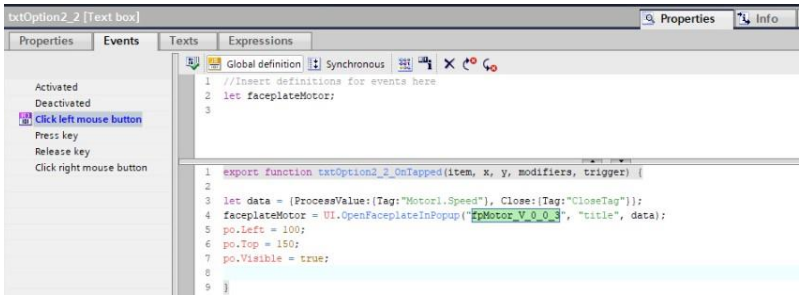
3. Add the snippet "Open faceplate in popup" and link the two interfaces (see also Section 0), as follows:
   - "ProcessValue" with "Motor1.Speed"
   - "Close" with "CloseTag"

Figure 5-32



4. Then replace "po" with "faceplateMotor" (from the global definition area) and remove "let" in line 4.

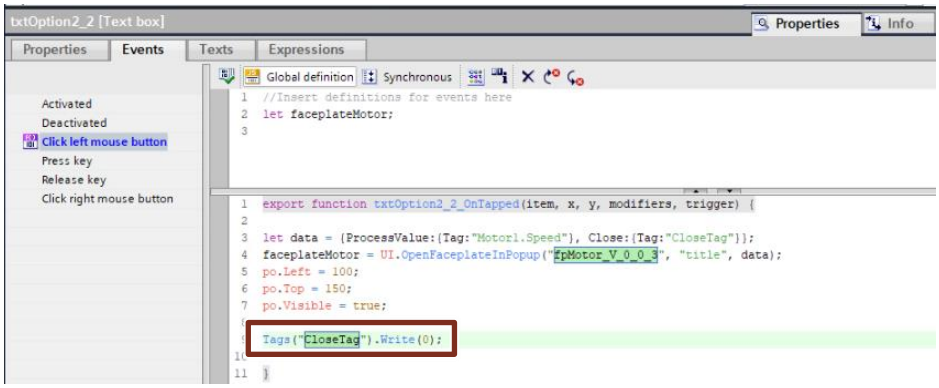
Figure 5-33



5. Finally, add the following code in order to reset the "CloseTag".

```
Tags("CloseTag").Write(0);
```

Figure 5-34



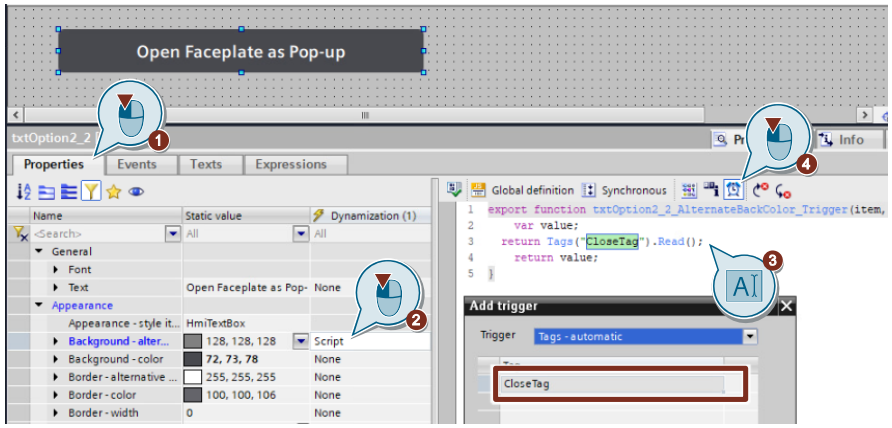
Close faceplate when tag value "CloseTag" is set

1. query "CloseTag"
   - Switch to the "Properties" of the button (1).
   - As dynamization, select "Script" for the property "Alternative background color" (2).
   - Enter the following code (3).

```
return Tags("CloseTag").Read();
```

- For the trigger for the script, select "Tags" and select the tag "CloseTags" (4).

Figure 5-35

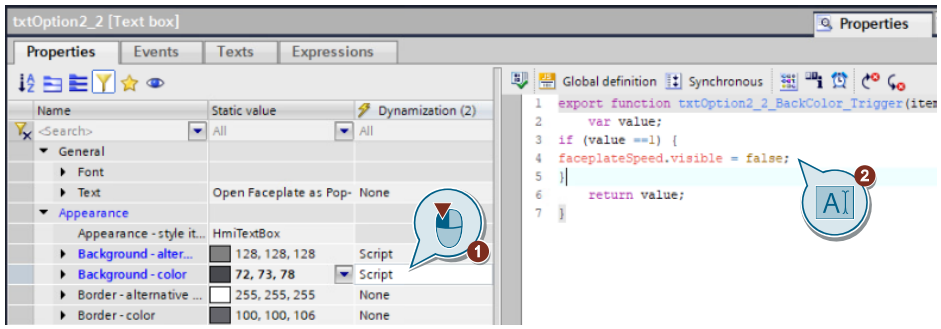


Note In the place of the property "Alternative background color" you can also dynamize a different unused property.

2. Closing a Faceplate
   - In the next step, for the property "Alternative background color", add "Script" as dynamization for "Change" (1).
   - Paste in the following code.

```
if (value==1)  {
  faceplateSpeed.Visible = false;
  }
```

Figure 5-36



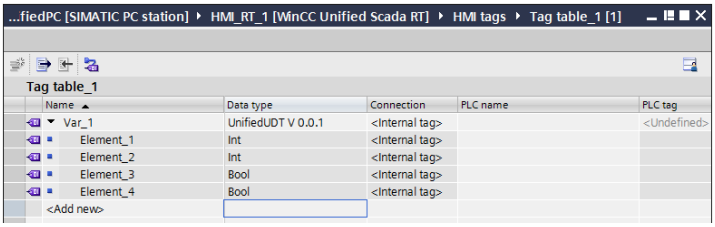
3. Save the changes and start the runtime.

5.6. Tags and UDTs

5.6.1. Access to HMI UDT Elements

SIMATIC WinCC Unified lets you use HMI UDTs in tag tables. You can also access the individual elements of the UDT via JavaScript with the keyword "Tags".

Figure 5-37



To be able to access the individual elements of the HMI_UDTs, the indication in JavaScript is made with a prefix (name of the UDT and separation point, e.g. "Let_1.") and suffix (name of the element, e.g. "Element_1").

Example:

```
// read value of "Element_1" in UDT "Let_1"
let tag1 = Tags('Let_1.Element_1').Read()
```

Note This applies to the instances of the library type HMI-UDT that were created in the project. The library type cannot be accessed directly.

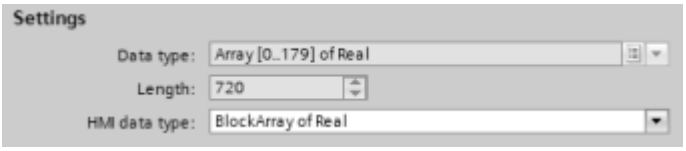
5.6.2. Reading Arrays as Individual Tags

Arrays can be read out as individual tags with a single function. With the "hmiReadDirect" method, it is possible to read the entire array at once. In this case, the unified array does not need to be converted to a Javascript array to directly determine the length of the array, for example.

```
const arrayValues = Tags("WString_Array").Read(Tags.Enums.hmiReadType.hmiReadDirect);
HMIRuntime.Trace(arrayValues[0]);
```

In this case, "WString Array" is a block array. In contrast to the standard unified array, this array can be read out in a single pass. It can also be configured flexibly in the tag table.

Figure 5-38



Notes:

The data block is always treated as a whole.

Simplified scripting thanks to a dynamic array size.

Can also be used for parameter control, trend control and logging.

Can be used in RT Openness and ODK.

5.6.3. Loop Breakers

You can add scripts to internal tags which will be executed when a value changes. Infinite loops can easily occur in the process, as shown by the example in the following table.

Example:

Table 5-6

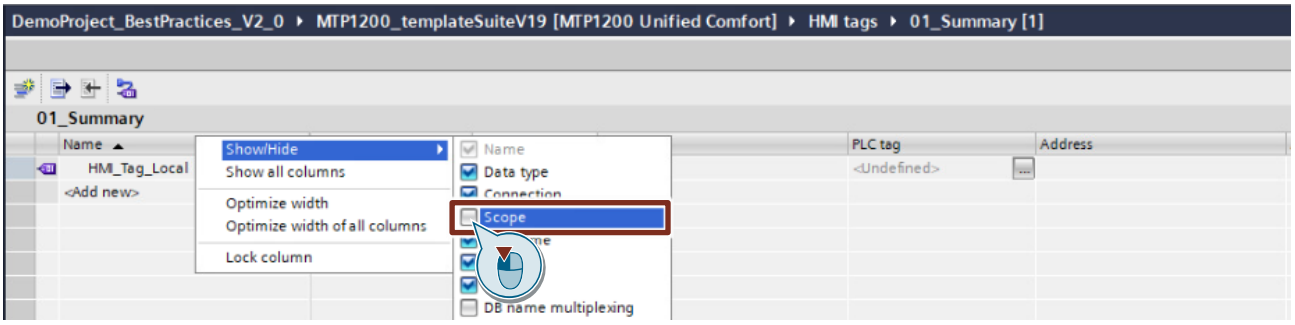
Tag	Data type	Goal of the task: Script function after value changes
"Tag_1"	Internal tag	Increment tag "Tag_2" by 2
"Tag_2"	Internal tag	Increment tag "Tag_1" by 2

If there are loops, SIMATIC WinCC Unified will detect and interrupt them.

5.6.4. Using Client-Internal Tags

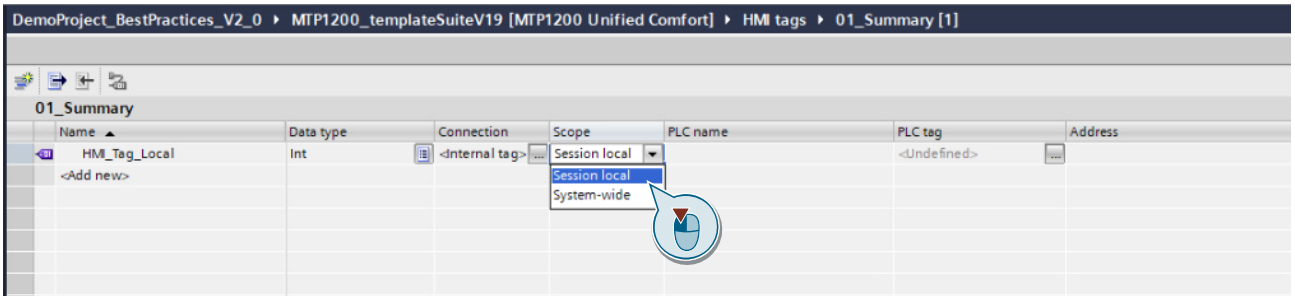
In WinCC Unified you can declare tags as "Session-local". This means that if you change the tag value on a client, it is not changed on the server, but only for the client session. To define an internal tag as "Session-local", you must first make the scope attribute visible in the tag table view.

Figure 5-39



Under "Scope" you can switch between System-wide and Session local.

Figure 5-40



As an alternative solution, you can use the "DataSet" object (for more on this, see the description of the DataSet snippet in Section 5.2). There are two different types here:

- "DataSet" local to the session (browser tab)
- "DataSet" local to the screen

You can find the complete code for both variants at the end of this section.

Restriction

The following constraints apply when using DataSets:

- Only simple types (Numbers, Strings, Bools) can be saved. JavaScript objects or classes cannot be saved.
- Triggers on changes are not possible.
- DataSets are only valid in the script context, while the internal tags can be used anywhere in the project. E.g., for direct dynamizations.

Code for session-local "DataSet"

```
/* Use the session local Dataset to store any kind of data (numbers, strings, Booleans, structs …) within a session of a browsertab (dataset is kept after changing basescreen, but deleted after logout/login).*/

const ui = HMIRuntime.UI;
ui.DataSet.Add('UI_Tag_1',123);
ui.DataSet.Add('UI_Tag_2',1234);

for(let Index in ui.DataSet)
 {
  HMIRuntime.Trace("Key = " + ui.DataSet[Index].Name + " Value =" +
  ui.DataSet[Index].Value);
 }

for(let Data of ui.DataSet)
 {
  HMIRuntime.Trace("Key = " +Data.Name + " Value =" + Data.Value);
 }

if(ui.DataSet.Exists('UI_Tag_2'))
 {
  ui.DataSet.Remove('UI_Tag_2');
  HMIRuntime.Trace('UI_Tag_2 Removed');
 }

ui.DataSet.Clear();
```

Code for local "DataSet" screen

```
/*Use the screen local Dataset to store any kind of data (numbers, strings, Booleans, structs, etc.) within a screen session (dataset is deleted after changing basescreen).*/
const screen = Screen;
screen.DataSet.Add('Screen_Tag_1',123);
screen.DataSet.Add('Screen_Tag_2',1234);

for(let Index in screen.DataSet)
 {
  HMIRuntime.Trace("Key = " + screen.DataSet[Index].Name + " Value
  =" + screen.DataSet[Index].Value);
 }

for(let Data of screen.DataSet)
 {
  HMIRuntime.Trace("Key = " + Data.Name + " Value =" + Data.Value);
 }

if(screen.DataSet.Exists('Screen_Tag_2'))
 {
  screen.DataSet.Remove('Screen_Tag_2');
```

```
  HMIRuntime.Trace('Screen_Tag_2 Removed');
 }

screen.DataSet.Clear();
```

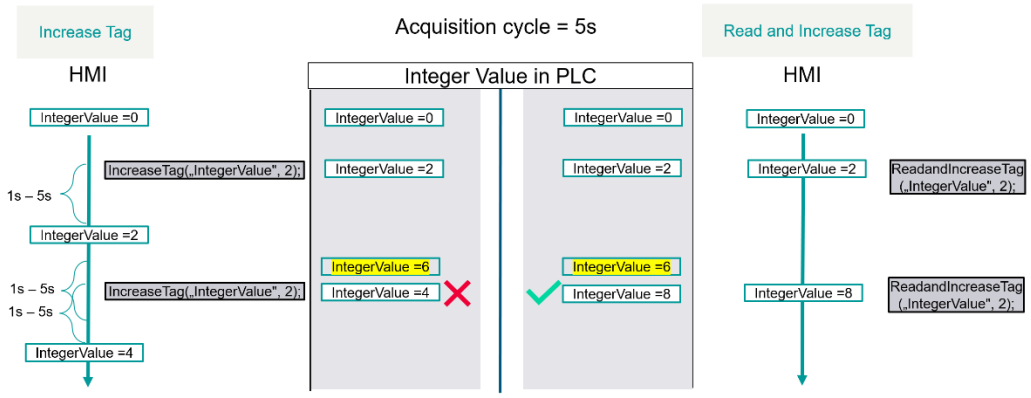
5.6.5. Inching

WinCC Unified works in such a way that new values become visible with each new data entry cycle. The value of the last synchronization is always used for system functions. If the tag has changed but the recording time has not yet expired, the PLC works with an old value. To avoid possible time-related complications in such cases, the "Inching" system function can be used.

With Inching, new values in the IO field become visible immediately after the system function is executed. This means that the latest value from the PLC is always used for the system function. Before the action, the PLC reads the tag again to ensure an updated value.

- Enables the synchronization of tag values between HMI and PLC before and after the write process, thus increasing reliability.

Figure 5-41



Example: Syntax for the ReadAndSetTagValue method:

```
[HMIRuntime.]Tags.Inching.SysFct.ReadAndSetTagValue(Tag,value);
```

Note

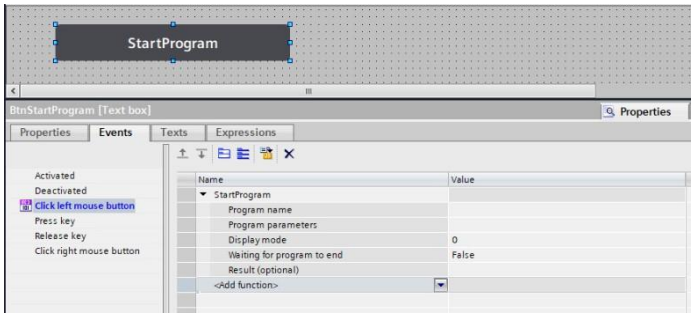
Further inching system functions can be found in the "WinCC Unified V19 – Programming reference": https://support.industry.siemens.com/cs/us/en/view/109896132/168730302219?dl=en

5.7. Starting Programs from the Runtime

You can start programs from the runtime by using the function "StartProgram". You can employ both of the following methods to do this:

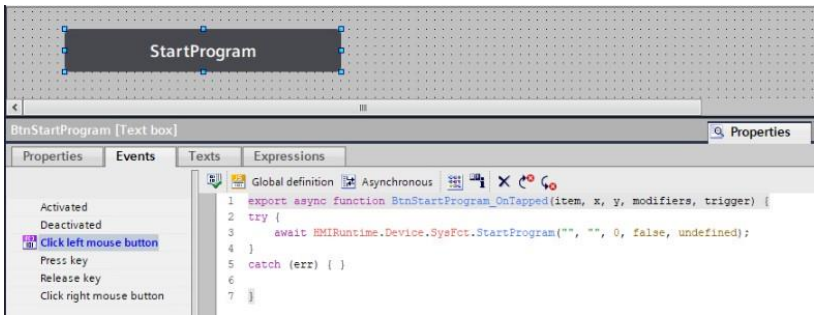
- "StartProgram" via the system function

Figure 5-42



- "StartProgram" via script

Figure 5-43



Note

General information on the "StartProgram" function can be found in the "SIMATIC WinCC Engineering V19 – Runtime Unified" system manual in the "StartProgram" section.

https://support.industry.siemens.com/cs/ww/en/view/109896132/156350910219

The functional scope of "StartProgram" differs, however, between the PC runtime and the runtime on the Unified Comfort Panel.

5.7.1. StartProgram in the Unified PC Runtime

If you execute the "StartProgram" function in the WinCC Unified PC Runtime, the system function itself can only be used to start programs that do not have a user interface.

If you want to start programs with user interfaces, you can only do so with the runtime Open Development Kit (ODK) or Open Pipe.

Note

Additional information about the runtime ODK can be found in the "Documentation" folder in the WinCC Unified installation folder:

C:\Program Files\Siemens\Automation\WinCCUnified\Documentation

5.7.2. StartProgram in the Unified Comfort Panel

You can start programs that have user interfaces on the Unified Comfort Panel using the "StartProgram" function. In this manner, you can typically launch the pre-installed programs (such as Doc Viewer, PDF Viewer).

Program name

The following table shows you which parameter "Program name" is used to start which program.

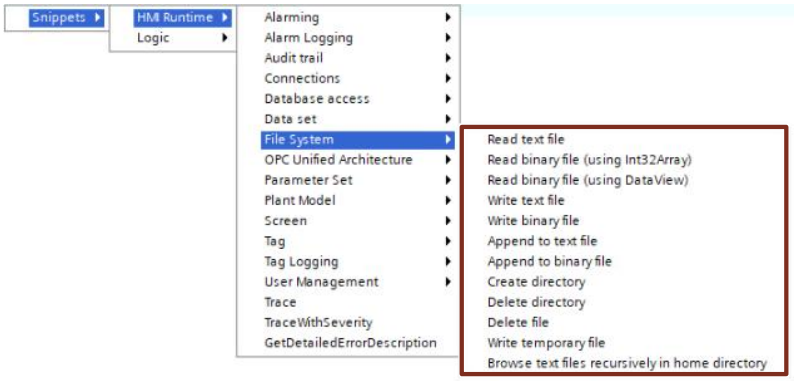
Table 5-7

Program to be launched	Input string for the parameter "Program name"
Doc Viewer	/opt/siemens/App_Restriction/runLibreoffice.sh
Email client	/opt/siemens/App_Restriction/runEvolution.sh
File Browser	/opt/siemens/App_Restriction/runThunar.sh
Media Player	/opt/siemens/App_Restriction/runVLC.sh
PDF Viewer	/opt/siemens/App_Restriction/runOkular.sh
Web Browser	/opt/siemens/App_Restriction/runChromium.sh

5.8. File Handling

You can also access your computer's file system with WinCC Unified. There are snippets in "HMI Runtime > File system" which can help you do this.

Figure 5-44



Note

All scripts are executed on the server in SIMATIC WinCC Unified.
When working with the file system (Snippet "File System"), you can only access files on the server and not on the file system of the client on which the script was triggered.

5.8.1. Creating a Folder

Using the snippet "HMIRuntime > File System > Create Directory", you can create a folder including subfolder in an already existing folder.

Creating folders on Unified PC

Based on the example of the snippet, the folder "mydatadir" is created in the folder "C:\Users\Public" along with the subfolder "mysubdir". Once the folder has been created, a trace message will be output.

```
HMIRuntime.FileSystem.CreateDirectory("C:\\Users\\Public\\mydatadir\\mysubdir").then(
  function() {
    HMIRuntime.Trace("Directory successfully created");
});
```

Note

When specifying the directory, pay attention to the double backslash ("\\") between the individual folders.

If you want to create a folder under "C:\\...", this may be rejected due to a lack of access permissions.

Creating folder on Unified Comfort Panel

If you want to create a folder on your Unified Comfort Panel, you must change the specified path to suit the Linux operating system.

```
HMIRuntime.FileSystem.CreateDirectory("/home/industrial/mydatadir/ mysubdir").then(
  function() {
    HMIRuntime.Trace("Directory successfully created");
});
```

Note

In your Unified Comfort Panel, you only have permission to create folders and files in the directory "/home/industrial/".

However, you can also directly access external storage devices (USB drive or SD card) and create a folder there.

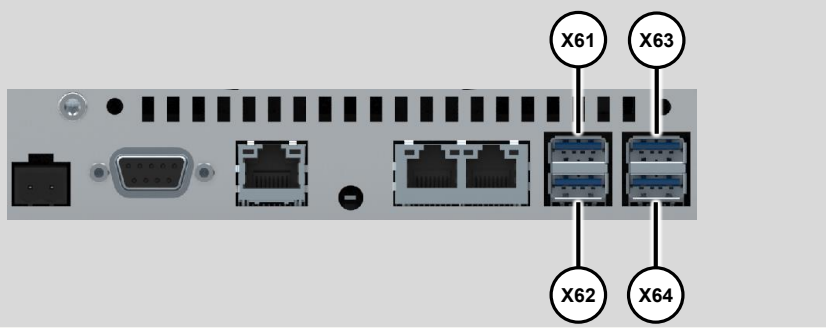
To do so, you must specify the path as follows:

```
SD card: "media/simatic/X51/…"
USB:     "media/simatic/X61/…"
```

Note

The specified path for USB drives depends on which USB port the USB drive is connected to. The overview below explains the difference between the ports.

Figure 5-45



5.8.2. Writing Values to a File and Create File

Using the method "WriteFile", you can create a file, "textfile.txt", in the specified path (here, "C:\Users\Public\"). The "Process value:" is then written to this.

Finally, one more message is output in TraceViewer as to whether the write job was successful or if an error occurred.

```
HMIRuntime.FileSystem.WriteFile("C:\\Users\\Public\\textfile.txt", "Process value: ",
"utf8").then(
function() {
  HMIRuntime.Trace("Write file finished successfully");
}).catch(function(errorCode) {
  HMIRuntime.Trace("Write failed errorcode=" + errorCode);
});
```

Note You can find further information in the system manual "SIMATIC WinCC WinCC Engineering V19 – Runtime Unified" in the section entitled "'WriteFile' method (FileSystem.WriteFile)":

https://support.industry.siemens.com/cs/ww/en/view/109896132/162204099851

Adding values to an existing file

If a file has already been created, you can add values to the end of the file with the method "AppendFile".

Example

```
HMIRuntime.FileSystem.AppendFile("C:\\Users\\Public\\textfile.txt", "Added Process value: ",
"utf8").then(
  function() {
    HMIRuntime.Trace("Write file finished successfully");
});
```

If you previously read the tag value, you can then add it to the text file in a similar manner.

Example

```
//Read process value
let tag1 = Tags("MyTag1").Read();

//Append process value to text file
HMIRuntime.FileSystem.AppendFile("C:\\Users\\Public\\textfile.txt", "Added Process value: " +
tag1 + "\n", "utf8").then(
  function() {
    HMIRuntime.Trace("Write file finished successfully");
});
```

Note Start a new line with the command "\n". More commands can be found at the following link:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

5.8.3. Reading Values from a File

You can also read values from a text file with the "ReadFile" method.

In this example, the content of the file "textfile.txt" in the directory "C:\Users\Puplic" is read and then output in TraceViewer and in the text field "Textbox".

Example

```
HMIRuntime.FileSystem.ReadFile("C:\\Users\\Public\\textfile.txt", "utf8").then(
  function(text) {
    HMIRuntime.Trace("Text=" + text);
    Screen.Items('Textbox').Text = text;
});
```

Note For security reasons, reading the contents of an entire folder is not supported.

Read binary file

You can read a binary file (".bin") in WinCC Unified with the method "ReadFileBinary". The type of file to be read is divided into the following two classes:

- Int32Array

  More information about "Int32Array" can be found at:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Int32Array
- DataView
  You can find additional information on the "DataView" view at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView

Note General information and a code example of how you can read binary data can be found in the system manual "SIMATIC WinCC Engineering V19 – Runtime Unified" in the section:

Method "ReadFileBinary"
https://support.industry.siemens.com/cs/ww/en/view/109896132/162203612299

"Reading and writing binary files"
https://support.industry.siemens.com/cs/ww/en/view/109828368/152137197323

5.8.4. Subscribe to Tags

The "TagSubscription" object enables HMI tags to be monitored for changes. The object is initialized via the "CreateSubscription" method of the "Tags" object. The object has the following methods:

- Start() – Activates the monitoring of HMI tags of the "TagSubscription" object.
- Stop() – Stops the monitoring of HMI tags of the "TagSubscription" object.

Example code:

```
const subs = HMIRuntime.Tags.CreateSubscription(['HMI_Tag_1', 'HMI_Tag_2'], (changedTags)
=> {
 for (const tag of changedTags) {
 HMIRuntime.Trace(`Tag ${tag.Name} value updated to ${tag.Value}`);
 }
 subs.Stop();
});
subs.Start();
```

A snippet is already available for the "TagSubscribtion". Especially if the same script logic is implemented in the scripts (e.g. if-else), you save a lot of script execution through monitoring.

Figure 5-46

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

let subs = HMIRuntime.Tags.CreateSubscription(['HMI_Tag_1', 'HMI_Tag_2', 'HMI_Tag_3'], function(TagArray) {
  for (let index in TagArray)
  {
    HMIRuntime.Trace("Tag Name_" + (index+1) + " = " + TagArray[index].Name);
    HMIRuntime.Trace("Tag Value_" + (index+1) + "= " + TagArray[index].Value);
    HMIRuntime.Trace("Tag QualityCode_" + (index+1) + "= " + TagArray[index].QualityCode);
    HMIRuntime.Trace("Tag TimeStamp_" + (index+1) + "= " + new Date(TagArray[index].TimeStamp));
    HMIRuntime.Trace("Tag LastError_" + (index+1) + "= " + TagArray[index].LastError);
    HMIRuntime.Trace("Tag ErrorDescription_" + (index+1) + "= " + TagArray[index].ErrorDescription);
  }
});
subs.Start();

```


5.8.5. Read Tag Array

To read the array length of an HMI tag of type array, you can use the following code:

```
let esc = false;
let name = "HMI_Tag_1";
let i = 0;

do{
  let tag1 = Tags(name+"["+i+"]");
  let tagValue1 = tag1.Read();

  if(tagValue1 == undefined){
    esc = true;
  }else {
    HMIRuntime.Trace("value of MyTag1: " + tagValue1);
    i++;
  }
}
while(!esc)

HMIRuntime.Trace("The length of the array is: " + i);
```

The loop reads value from the array until an entry has the value "undefined". At this point, the loop is interrupted and the result is output with a trace message.

5.9. Configuring Time Delays in a Script

You can configure time delays in a script by using the "Timer" object.

Note

For further information, refer to the system manual "SIMATIC WinCC Engineering V19 – Runtime Unified" in the section entitled "'Timers' object":

https://support.industry.siemens.com/cs/ww/en/view/109896132/157858368651

An example of the implementation of timers in WinCC Unified is the operation of setting a signal while a button is pressed.

Example

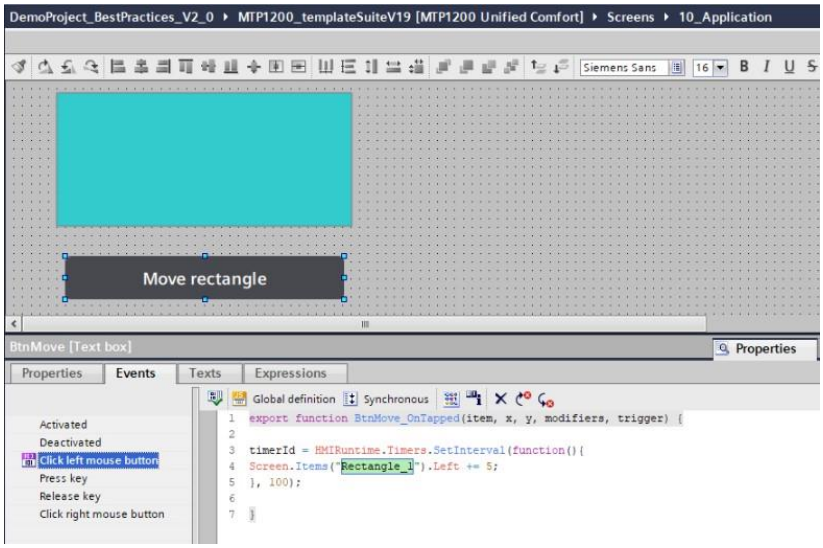
In the following example, a button called "btnMove" is supposed to be pressed. As long as the button is held down, a rectangle "Rectangle_1" should move horizontally.

To do this, proceed as follows:

1. Add the following script to the button at the "Click left mouse button" event:

```
timerId = HMIRuntime.Timers.SetInterval(function() {
   Screen.Items('Rectangle_1').Left += 5;
  }, 100);
```

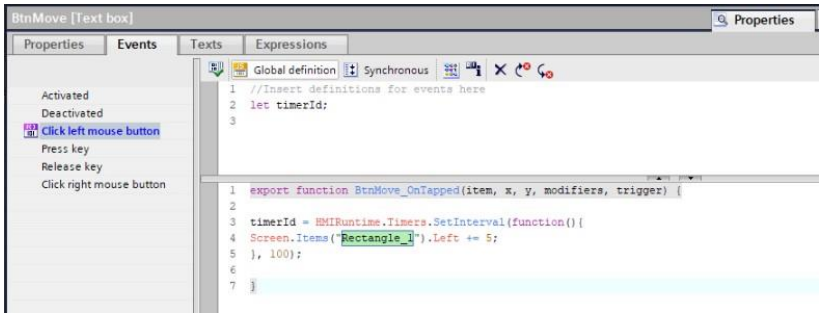
Figure 5-47



2. Then add the following code line to the global definition area:

```
// Global definition area
let timerId;
```

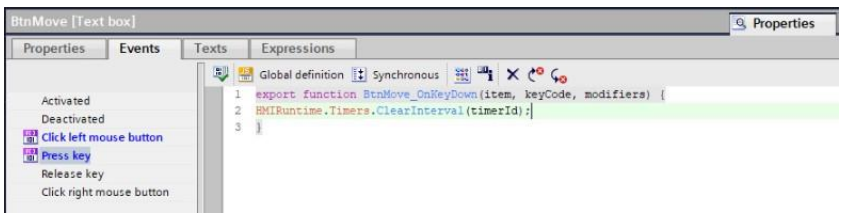
Figure 5-48



3. In the last step, add the following script to the same button at the "Release" event:

```
HMIRuntime.Timers.ClearInterval(timerId);
```

Figure 5-49

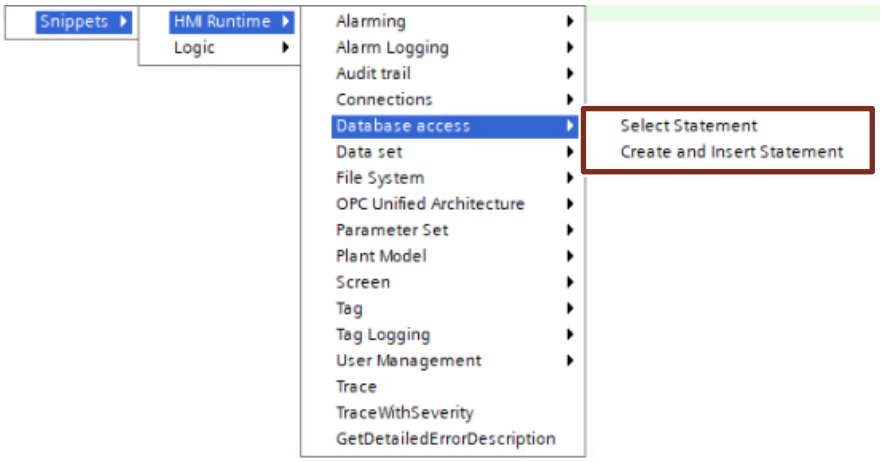


Note	Ensure that you do not configure scripts for the "Press key" and "Release key" events, as these triggers are not executed during touch operation

5.10. Configuring Access to Databases

SIMATIC WinCC Unified gives you the ability to connect with databases and exchange data. Two snippets will help you do this.

Figure 5-50



For how to access your SQL or SQLite database from WinCC Unified, please refer to the FAQ "How do you access an SQLite / Microsoft SQL database via the WinCC Unified PC Runtime?".

https://support.industry.siemens.com/cs/ww/en/view/109781074

NOTICE	Under no circumstances should you make manual changes to the database from WinCC Unified, otherwise inconsistencies can occur and you will have to completely reinstall your system.

5.10.1. Installation und and Configuration of SQL and SQLite

The following section describes the installation process for the two variants, *Microsoft SQL Server* and *SQLite*. The installation files can be found on the installation DVD.

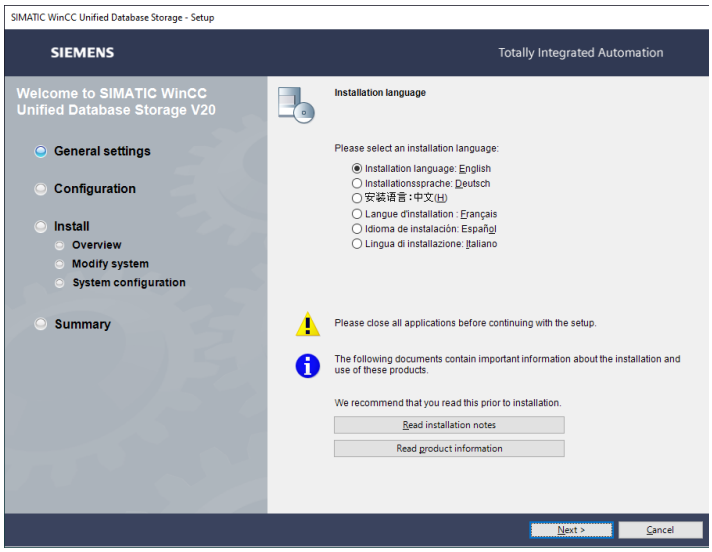
ACHTUNG

Unified PCs use **SQLite** as the default database type. To use **Microsoft SQL**, the system provides an installation option with a setup package. After installing Microsoft SQL, logging with SQLite is no longer possible.
Existing SQLite files are retained but can no longer be used at runtime.

Microsoft SQL:

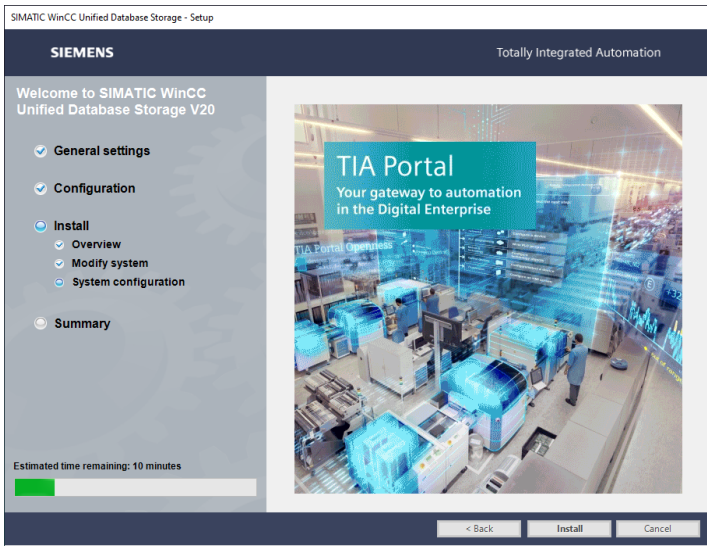
Run the downloaded ISO file. This will open the installation wizard. In the following setup window, various settings can be configured, such as specifying the installation location.

Figure 5-51



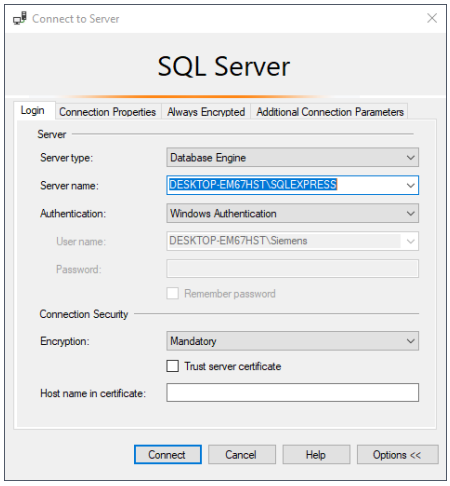
Click through the installation wizard until the installation begins. A restart may be required to complete the installation.

Figure 5-52



Microsoft SQL Server Management Studio can be downloaded from the official Microsoft website. After downloading, run the installation file and follow the instructions of the installation wizard. Once the installation is complete, a system restart is required. After restarting, the Management Studio can be opened from the Start menu. When launched, a connection dialog will appear.

Figura 5-53



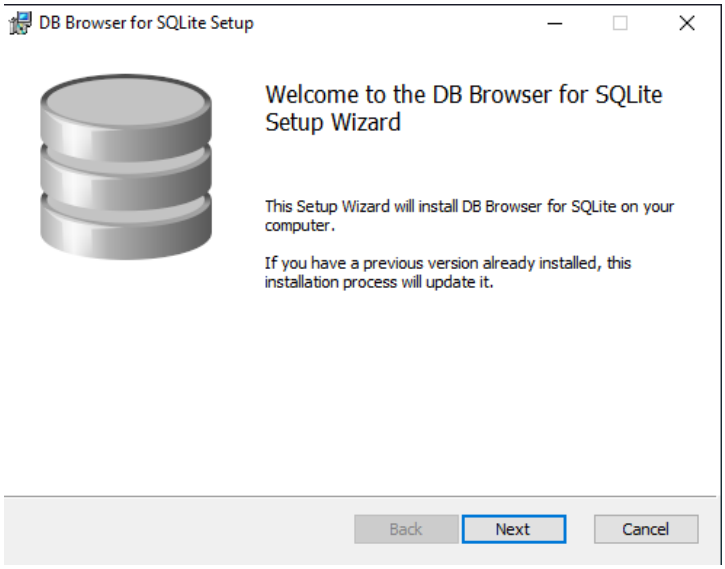
In the connection window, select the appropriate SQL Server, e.g., the WinCC server. You can then connect to the server and begin managing the SQL databases. At this stage, it is no longer necessary to select a specific database.

SQLite:

To connect with Microsoft SQL, a current ODBC driver must be installed. This driver is essential for access via scripting and is installed by default.

In addition to the driver, a tool can be used to view the entries. There are many tools available; in this example, **DB Browser** was used. This tool allows you to open and view SQLite databases. Unlike Microsoft Management Studio, the user must manually navigate to the database storage location in order to open them. The storage location can either be defined in the Runtime Settings or centrally managed through the Unified configuration.

Figure 5-54



After the installation, the databases can be accessed without any further configuration.

5.10.2. Establishing Database Connections Once

1. Snippet

If you want your WinCC Unified Runtime to access the same database multiple times, it is recommended to establish the database connection once and store it within the session.
For this purpose, you can use the code snippet **"HMI Runtime > Data set > Dataset with Database Connection"** (see also Chapter 5.2).

With repeated database access, this allows you to access the connection more quickly without requiring an additional connection setup, which in turn reduces the load on your project.

Code Snippet:

```
(async function() {
 try{
  let connectionstring = "Driver={SQL Server}; Server=localhost;
Database=MyDB;trusted_connection=yes;";
  let conn = await HMIRuntime.Database.CreateConnection(connectionstring);
  HMIRuntime.UI.DataSet.Add("DatabaseConnection",conn);
  let con = HMIRuntime.UI.DataSet("DatabaseConnection");
  let query = "Select * from MyTable;";
  let results = await con.Execute(query);
   if(results !== undefined && results !== null)
  {
    let statements = results.Results;
    for(let statement in statements)
    {
      let rows = statements[statement].Rows;
      for (let i in rows)
      {
        let row = rows[i];
        for(let key in row)
        {
           HMIRuntime.Trace(key+":"+row[key]);
        }
      }
    }
  }
  conn.Close();
}
catch(e)
{
  let res = e.Results;
  for(let statement in res)
  {
    let errors = res[statement].Errors;
    for (let i in errors)
      {
        let detailed = errors[i];
        HMIRuntime.Trace("Errors state : "+detailed.State);
        HMIRuntime.Trace("Errors Message : "+detailed.Message);
      }
    }
  }
```

Note	Note that the connection is only available to the logged-in user within a session (browser tab). If the user is changed or the session is ended, the stored database connection is lost.

2. Configuration Options

From this point, you have two configuration options:

1. Variant with SQLite
2. Variant with Microsoft SQL

Each variant requires a different configuration string and a different SELECT statement. The inner part, i.e., the IF statement and the FOR loop, is the same for both variants since the result of the statement has the same structure.

Hinweis	It is also important that the asynchronous function is secured with a catch block. This is essential; otherwise, other logging functions such as alarm logging or tag persistency could fail.

SQLite:

For the configuration of SQLite, the connection string is required by default. This consists of the driver name and the driver version. Unlike SQL, the database must then be referenced directly with its path. The specification of whether the connection is established via a *"Trusted Connection"* indicates that no transfer of login credentials is required. A *"Trusted Connection"* can only be used if the Runtime (RT) and the database are running on the same PC.

```
let connectionstring = "Driver=SQLite3 ODBC Driver;
Database=C:\\ProgramData\\SCADAProjects\\HMI_RT_1\\RecipeService_59331018-A581-45D8-B7A8-
DB9D276827E7\\HMI_RT_1_RecipeStorage.db3;trusted_connection=yes;";
```

The query is then executed using the SQLite statement. In this example, the statement is composed as follows:

- **SELECT Name** – This command returns the "Name" column from the "Recipe" table. Only the record names are queried.
- **FROM 'Recipe'** – This specifies the "Recipe" table as the data source. The table contains the records from which the "Name" column is retrieved.
- **WHERE LifeCycleState = 4** – This condition filters the results so that only the records from the "Recipe" table are returned where the value in the "LifeCycleState" column equals 4.

```
let query = "SELECT Name FROM 'Recipe' WHERE LifeCycleState = 4;";
```

Microsoft SQL:

The configuration of Microsoft SQL begins with the connection string. Here, the name of the driver and its version are required. Next, the server must be specified. This is achieved by using the runtime station name and the SQL server name, which are separated by a double "\" as shown in the example below. Finally, the name of the database is needed, along with an indication that it is a *"Trusted Connection"*.

```
let connectionstring = "Driver=ODBC Driver 17 for SQL Server; Server=UNIFIEDDEVICE\\WINCCUNIFIED;
Database=HMI_RT_1_Recipe;trusted_connection=yes;";
```

The SQL statement should then be adjusted. It is also composed of several components:

- **SELECT** – Specifies the columns to be retrieved (* for all columns).

- **FROM** – Specifies the table and, if applicable, the schema or database.
- **WHERE** – Filters the results based on conditions.
- **GROUP BY** – Groups the data, usually in combination with aggregate functions (COUNT, SUM, AVG, etc.).
- **HAVING** – Filters groups based on a condition (after GROUP BY).
- **ORDER BY** – Sorts the result (ASC for ascending, DESC for descending).
- **LIMIT / TOP** – Limits the number of returned rows.

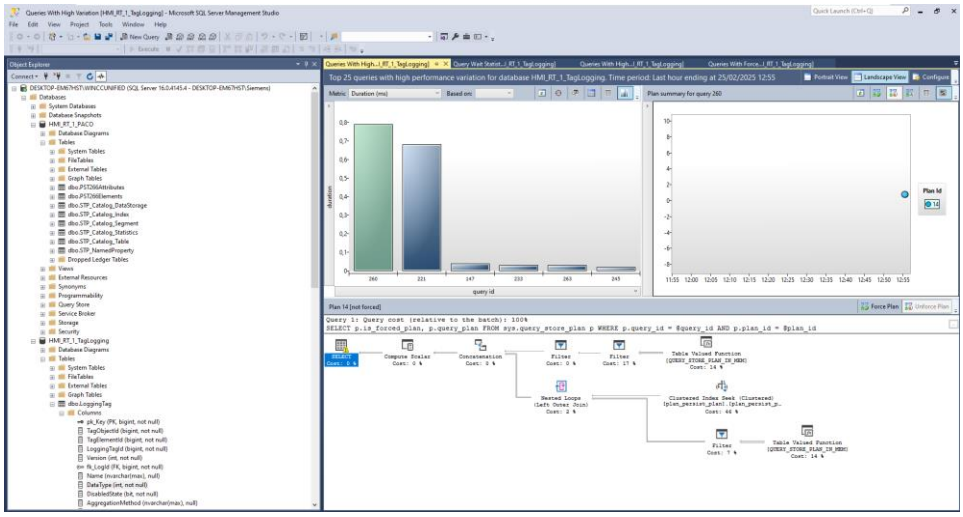
In this example, in addition to the name of the database, the desired column is also selected. A WHERE statement can also be used to filter the results according to specific criteria.

```
let query = "SELECT Name FROM [HMI_RT_1_Recipe].[dbo].[Recipe] WHERE LifeCycleState = 4; ";
```

3. SQL Server Management Studio & DB Browser

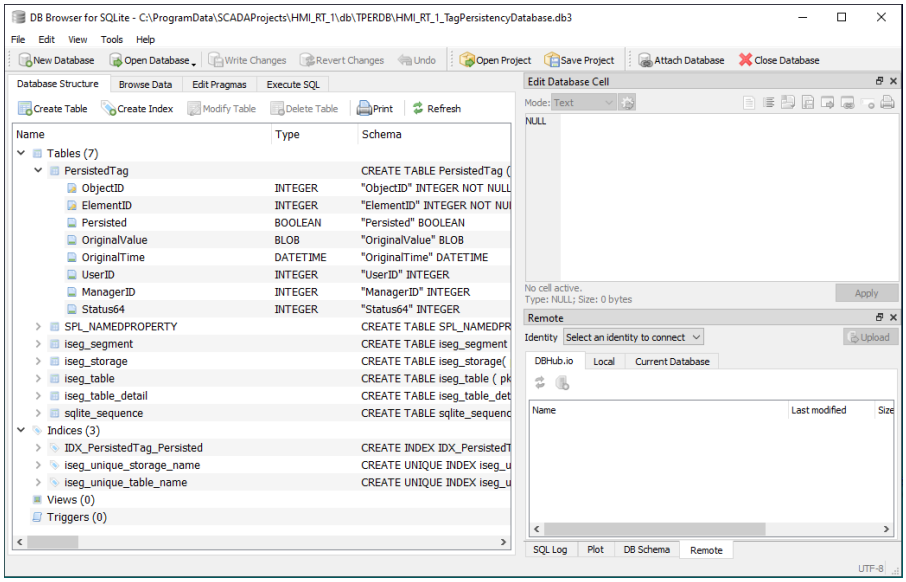
SQL Server Management Studio (SSMS) is an administration tool for SQL Server databases that provides a graphical interface for creating, managing, and querying databases. It can be downloaded from the Microsoft website.

Figure 5-55



DB Browser for SQLite is an open-source tool for managing SQLite databases that provides a user-friendly interface for creating, editing, and querying databases.

Figure 5-56



5.11. Configuring Access to Internet Resources

For security reasons, access to internet resources (e.g. REST API, Json files) via JavaScript is not supported from the runtime.

If you still want to share data from the WinCC Unified runtime with other applications, you have access to the runtime ODK functions or OpenPipe.

Note

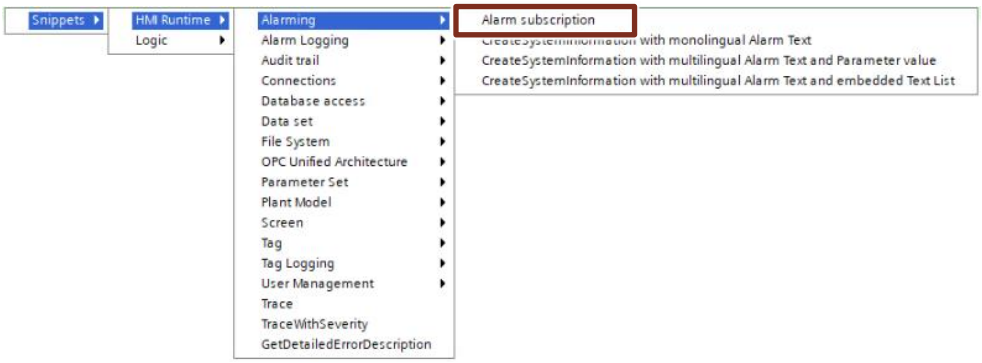
Further information is available in:

- in Section "Runtime API" in the "SIMATIC WinCC Engineering V19 – Runtime Unified" system manual:
  https://support.industry.siemens.com/cs/ww/de/view/109828368/160915757835
- in the operating manual - "SIMATIC HMI WinCC Unified Open Pipe"
  https://support.industry.siemens.com/cs/ww/de/view/109828368/160914252683

5.12. Filtering Alarms and Messages

In WinCC Unified, you can use JavaScript to display a selection of pending alarms via the "Alarm subscription". The snippet of the same name will help you do this.

Figure 5-57



Using the "Filter" property, you can specify filter criteria to filter the active alarms, in this example the alarm class Warning.

Example

```
//Snippet "HMI Runtime > Alarming > Alarm subscription"
let subs = HMIRuntime.Alarming.CreateSubscription();
subs.Filter = "AlarmClassName=\"Warning\"";
subs.Language = 1033;
subs.OnAlarm = function(Errorcode, SystemNames, ResultSet) {
  for (let index in ResultSet)
  {
    HMIRuntime.Trace("Alarm Name_" + (index+1) + " = " +
    ResultSet[index].Name);
    HMIRuntime.Trace(" Alarm State_" + (index+1) + "= " +
    ResultSet[index].State);
  }
};
subs.Start();
```

Note

Further information about the object "Alarm subscription" can be found in the system manual "SIMATIC WinCC WinCC Engineering V19 – Runtime Unified" in the section titled "Object 'AlarmSubscription'":

https://support.industry.siemens.com/cs/ww/en/view/109896132/159274234507

Note

You can find more options for filtering alarms and messages in the application example "Filtering messages and alarms in SIMATIC WinCC Unified".

https://support.industry.siemens.com/cs/ww/en/view/109760056

5.13. Switching the Runtime Language

You can configure a language change in SIMATIC WinCC Unified in two different ways as follows:

- with the system function "ToggleLanguage"
- with the snippet "Set Language"

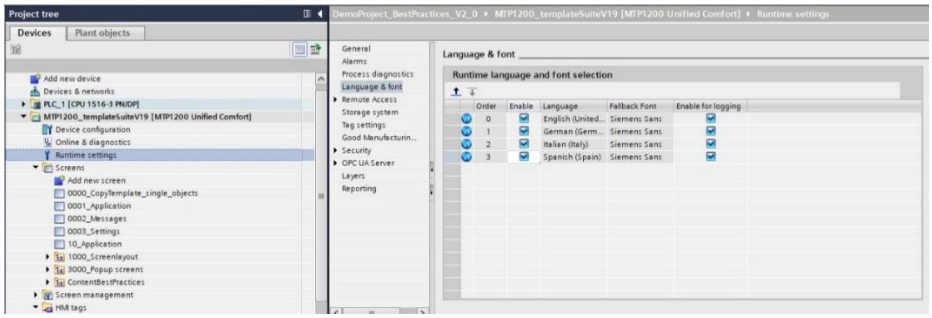
Changing language with the system function

Use the system function "ToggleLanguage" to switch between languages which are configured in the runtime settings.

Note

The "ToggleLanguage" function is also available as the system function "UmschalteSprache" in German.

Figure 5-58



Languages will be switched in the order in which they are configured in the runtime settings in the "Order" column.

Setting the language with the snippet

You can set the runtime language with the snippet "HMI Runtime > Language > Set Language". In the process you will enter a decimal ID for the respective language in the script editor.

Note

The snippet is also available as a system function, "SetLanguage" ("SetzeSprache" in German).

The table below contains a selection of languages with the associated language ID.

Table 5-8

Language	Language/region tag	Decimal language ID
German	de_DE	1031
English	en_US	1033
French	fr_FR	1036
Spanish	es_ES	3082
Italian	it_IT	1040
Chinese	zh-CN	2052

Note

Further language IDs are available at the following link:

https://msdn.microsoft.com/en-US/library/windows/hardware/dn898488(v=vs.85).aspx

5.14. "Math" Object

By default, JavaScript does not allow editing of 64-bit data types. To operate on these data types anyway WinCC Unified has the "Math" object.

The "Math" area has the following objects, properties and methods:

- "Int64" (property and object)
- "Int64Base" (property and object)
- "Uint64" (property and object)
- "DatePrecise" (property and object) → see also Section 5.15.3
- "RGB" (method) → see also section 0
- "RGBWeb" (method)

Note Further information about the "Math" object and the associated properties and methods can be found in the system manual "SIMATIC WinCC Engineering V19 – Runtime Unified":

https://support.industry.siemens.com/cs/ww/en/view/109896132/160001878027

5.15. Configuring the Date and Time

5.15.1. Working with Local Date/Time

The JavaScript "Date" object allows you to work with date and time in SIMATIC WinCC Unified.

To crate a new object of the type Date, use "new Date ()".

Example:

```
let myDate = new Date();     // create a Date object
let localTime, localDate, localDateTime;

//get local date and time to script internal tag
localDateTime = myDate.toLocaleString();

//get local date to script internal tag
localDate = myDate.toLocaleDateString ();

//get local time to script internal tag
localTime = myDate.toLocaleTimeString();
```

Note You can also indicate the two arguments "locales" and "options" to the "Date" object. You can find further information on the toLocaleDateString() method and arguments at the following link:
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Date/toLocaleDateString

5.15.2. Editing User-Defined Date/Time

You can also assign an individual date to the "Date" object. For this, enter the time in brackets when creating the date object.

Your can express the argument for the date and time indication as a string here, in the following format.

- new Date ("Day month year hour:second:millisecond")

Example:

```
let myDate = new Date("02 03 2016 1:10"); // create a Date object
let strTime, strDate;

//write date (20.03.2016) to script internal tag
strDate = myDate.toLocaleDateString ();

//write time (1:10:00 AM) to script internal tag
strTime = myDate.toLocaleTimeString();
```

Alternatively, you can express the arguments for the date and time indication separately.

- new Date (year, month, day, hour, second, millisecond)

Example:

```
let myDate = new Date(2016,2,20,1,10);
// create the date object 20.03.2016 1:10:00 GMT
```

Note

When indicating the month, note that the entry comprises 0 to 11. This results in:

- 0 = January
- 1 = February

  ...
- 11 = December

Note

If you only indicate year, month and day, the open time indications (hour, minute...) are completed with 0.

Besides the options described here, there are further options for expressing the date object argument.

5.15.3. Working with Timestamps on a Nanosecond Basis

If you need a timestamp down to the nanosecond, then you can implement this with the object "DatePrecise".

This object represents a highly granular time stamp with a resolution of 100 ns as a 64-bit integer value. This is particularly useful when archiving data. Furthermore, the SIMATIC controllers (e.g. S7-1516) also work on a nanosecond time basis, whereby a consistent time basis can be established.

The object also contains methods for converting between various timestamp formats.

Note

Further information about the "DatePrecise" object and the associated properties and methods can be found in the system manual "SIMATIC WinCC Engineering V19 – Runtime Unified":

https://support.industry.siemens.com/cs/ww/en/view/109896132/160000905867

5.16. Script Diagnostics

5.16.1. "Alert()" Notification Window

The `alert()` method is available in JavaScript in order to output message in a notice window. This is usually used for script diagnostics.

However, this function is not available in SIMATIC WinCC Unified for security reasons. The following scenarios show by way of example what effects the `alert()` method could have:

- The "Alert" notice window appears on top of operating elements and thus prevents rapid operation of the system.
- The "Alert" notice window blocks the script and thus prevents further execution of the script.

Alternative RTIL TraceViewer

To nevertheless output messages and check the correct execution of scrips, you can create entries in the RTIL TraceViewer.

The following section shows you how to compose a message by script in the RTIL TraceViewer.

5.16.2. Diagnostics via RTIL TraceViewer

To quickly rectify errors that have occurred in the configuration and scripts, effective diagnostics are required. The TraceViewer for SIMATIC WinCC Unified provides a diagnostic option for this.

The following description shows you how to use JavaScript to create an entry in the TraceViewer with corresponding information text.

Creating a message by script

If you wish to output a message via TraceViewer in the script, you need to enter the following as syntax:

```
HMIRuntime.Trace('This is the text for the TraceViewer!');
```

If you wish to output the process value of a tag in addition to the notice text, this must first be read.

Example:

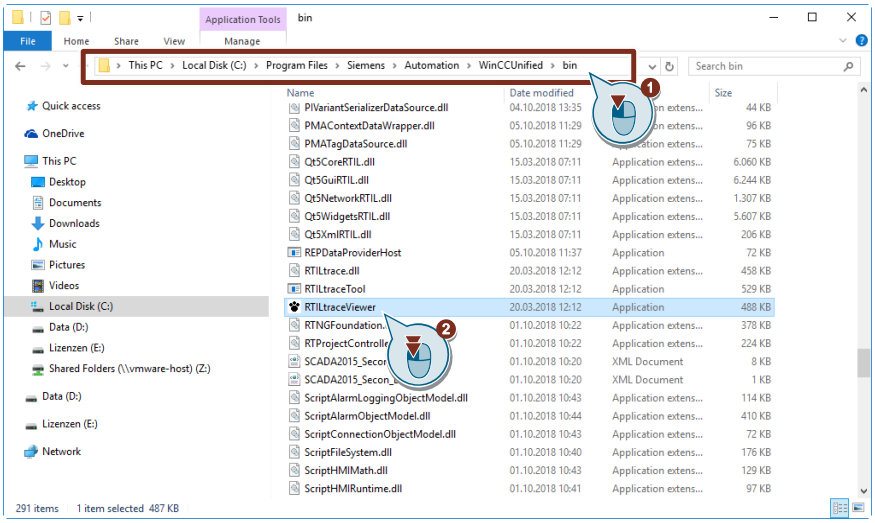
```
let tag1 = Tags('MyTag1');
let tagValue1 = tag1.Read();    //read value of 'MyTag1' e.g. 9.87

// create TraceViewer notification: 'value of MyTag1:  9.87'
HMIRuntime.Trace('value of MyTag1: ' + tagValue1);
```

Note	If you use the snippet for reading a tag, a TraceViewer notice will be included.

Opening TraceViewer

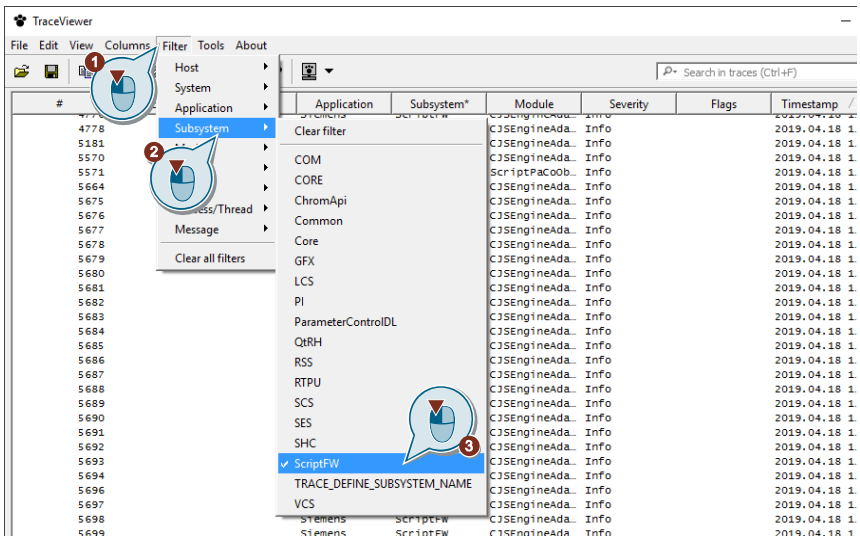
1. Open "Windows-Explorer" ("File Explorer").
2. Open TraceViewer
   - Open the path "C:/Program Files/Siemens/Automation/WinCCUnified/bin" (1).
   - Then double-click the application "RTILtraceViewer" to open the TraceViewer.

Figure 5-59



3. TraceViewer is opened.
4. Filter according to messages
   - Open the "Filter" menu (1) in the TraceViewer menu bar.
   - Then open the category "Subsystem" (2).
   - Select the ScriptFW module (3).

Figure 5-60



Only messages that you have created by script will now be displayed in TraceViewer.

Note Trace messages must be present in order to configure a filter.

Note

You can also integrate TraceViewer into TIA Portal as an external application and thus access it more quickly. Consult the section entitled "Integrate RTIL Trace Viewer as an external application" in the manual "SIMATIC WinCC Engineering V19 – Runtime Unified" for how to integrate TraceViewer.

https://support.industry.siemens.com/cs/ww/en/view/109828368/112955745419

TraceViewer and Unified Comfort Panel

If you have deployed a SIMATIC HMI Unified Comfort Panel, you can similarly use the RTIL TraceViewer. To do so, activate the "Trace forwarder" function in the Unified Comfort Panel.

Further information on this topic can be found in the FAQ "How do you use the Trace Viewer with the Unified Comfort Panel?"

https://support.industry.siemens.com/cs/ww/en/view/109777593

5.16.3. Debugging Scripts in Chrome

As an additional diagnostic option (e.g. setting break points), you can use the WinCC Unified script debugger together with the "Chrome" browser.

Additional information is available in the FAQ "How do you use the Script Debugger with WinCC Unified and the Chrome browser?"

https://support.industry.siemens.com/cs/ww/en/view/109779192

Script debugger for Unified Comfort Panel

The script debugger function is available to you only for the WinCC Unified runtime on a PC station.

In order to also use the script debugger function for the Unified Comfort Panel, you must simulate your Unified Comfort Panel project on a PC station.

5.16.4. Planning for Responses in Case of Error

Using the "try...catch" command, you can program an command (*try*) and the associated response (*catch*) to be executed in case of an error.

Note

You can find further information on the "try...catch" command at the following link:

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/try...catch

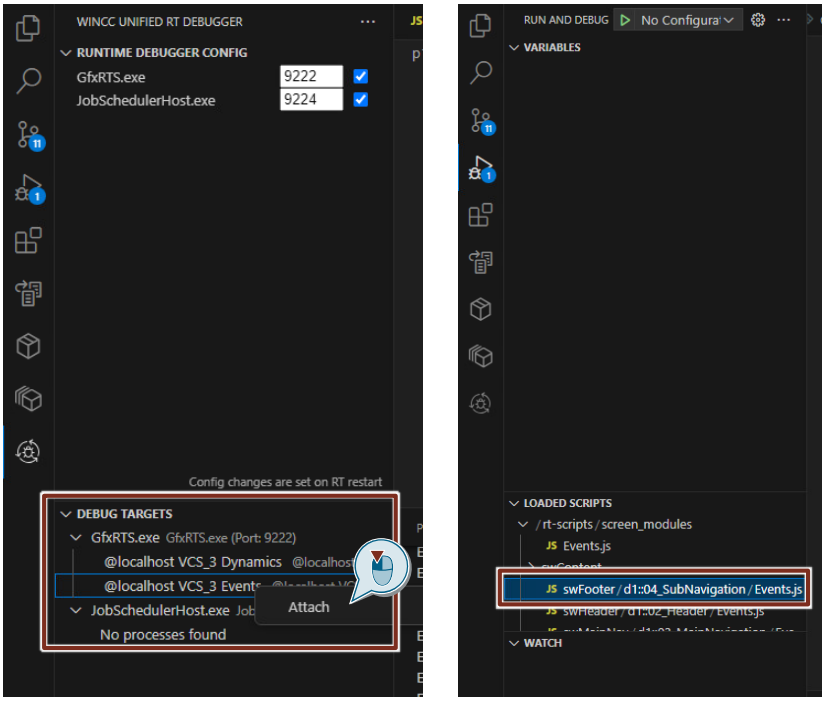
Snippet

In WinCC Unified, you can insert the "try...catch" and the "try...catch...finally" command into your code with the respective snippets. You can find them under "Snippets > Logic > try...catch" or "Snippets > Logic > try...catch...finally".

5.16.5. RT Debugger – Visual Studio Code Extension

With the RT Debugger Extension, it is possible to adapt the script code during debugging without having to download it via the TIA Portal. The scripts can be examined in Runtime in the Visual Studio Code Editor. The installation file is installed as a VSIX file. The script debugger must be activated in the WinCC Runtime Manager. After installation, we can select a script context and establish a connection to it.

Figure 5-61



After establishing the connection, proceed as follows:

1. Select the screen position of the script to be debugged
2. Define breakpoints and optional observers
3. Check the debugging

Restriction:

- Conditional breakpoints are not supported
- Minimized VS code is not automatically displayed when a breakpoint is reached.

Note

For additional information on the VS Code Extension, click here:

https://support.industry.siemens.com/cs/us/en/view/109827603

6. Useful Information

You can find further general information on JavaScript on the following websites, among others:

- Mozilla Developer Network https://developer.mozilla.org/en/docs/Web/JavaScript
- Introduction to JavaScript (SelfHTML)
  https://wiki.selfhtml.org/wiki/JavaScript
- JavaScript collection (German resource)
  http://www.jswelt.de/index.php
- JavaScript Tutorial
  https://javascript.info/

7. Appendix

7.1. 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

Industry Online Support

Industry Online Support is the previous address for information on our products, solutions and services.

Product information, manuals, downloads, FAQs and application examples - all information is available with just a few mouse clicks: support.industry.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts.
Please send queries to Technical Support via Web form: support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.
For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page: siemens.com/sitrain

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:

7.2. Application Support

Siemens AG
Digital Factory Division
Factory Automation
Production Machines
DF FA PMA APC
Frauenauracher Str. 80
91056 Erlangen, Germany

mailto: tech.team.motioncontrol@siemens.com

7.3. Links and literature

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to this entry page of this application example https://support.industry.siemens.com/cs/ww/en/view/xxx
\3\	System manual "SIMATIC WinCC Engineering V19 – Runtime Unified" https://support.industry.siemens.com/cs/ww/en/view/109828368/177361948043
\4\	Working with strings in JavaScript https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/toString
\5\	Operating manual "SIMATIC HMI WinCC Unified Open Pipe" https://support.industry.siemens.com/cs/ww/en/view/109826705

Table 7-1

7.4. Change documentation

Version	Date	Modification
V1.0	09/2018	First version
V2.0	05/2019	Description added to global modules Update to WinCC Unified V15.1 Small spelling corrections
V3.0	09/2020	Update to WinCC Unified V16 Expanded use cases
V4.0	08/2024	Update to WinCC Unified V19
V5.0	09/2025	Update to WinCC Unified V20

Table 7-2