Siemens
Industry
Online
Support

APPLICATION EXAMPLE

Integration of Custom Web Controls in WinCC Unified

SIMATIC WinCC Unified V19

SIEMENS

Legal information

Use of application examples

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

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

Disclaimer of liability

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

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

Other information

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

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

Security information

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

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

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

For additional information on industrial security measures that may be implemented, please visit https://www.siemens.com/industrialsecurity.

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

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under https://www.siemens.com/cert.

Table of Contents

1. General Information about Custom Web Controls

WinCC Unified offers an option for custom web controls (or CWC for short).

This means that your visualization is not limited to the standard control of WinCC Unified.

You have the option of creating your own custom controller using web technology or using existing controls and applying them to other WinCC Unified Runtime stations.

This application example explains how to integrate a "custom web control" created with Visual Studio Code into WinCC Unified.

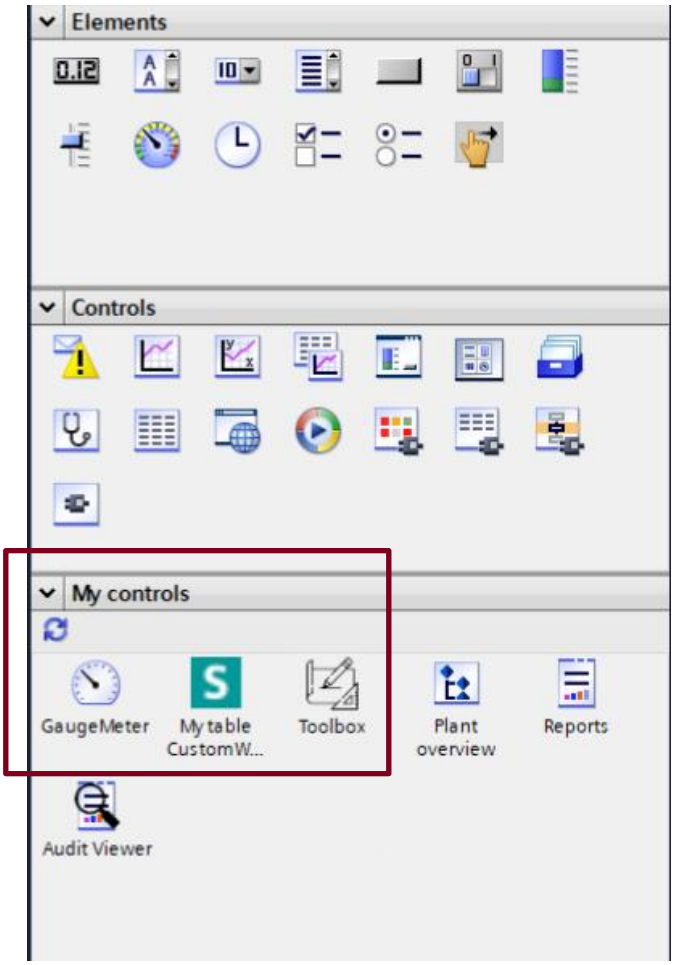
NOTE

Custom web controls and the following configuration steps work with the WinCC Unified PC Runtime and the Unified Comfort Panels.

Please note the limited performance of Unified Comfort Panels. Custom web controls with 3D representations are not recommended for Unified Control Panels, for example.

WinCC Unified Controller and elements

The following figure shows the elements, default control, and custom control ("My Controls") in WinCC Unified for Unified Comfort Panel. The "My Controls" area contains the custom web controls.



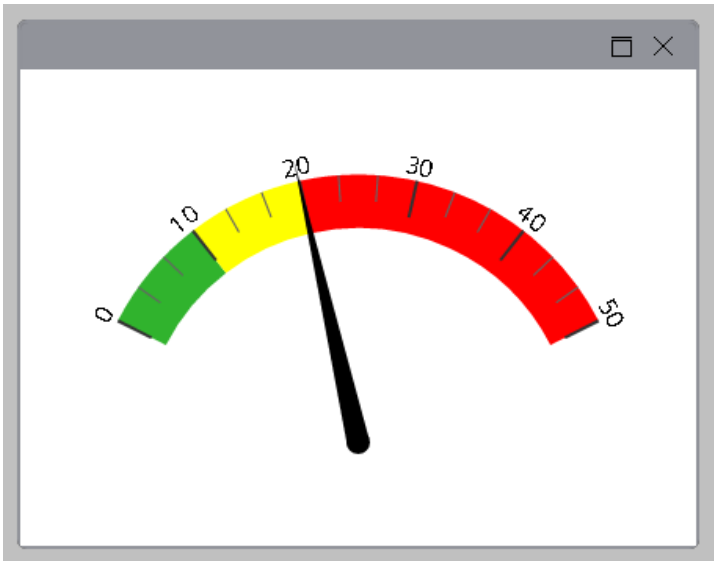
2. "Gauge" Custom Web Control

2.1. Introduction

2.1.1. Overview

The figure shows the "Gauge" custom web control, which is used as an example to demonstrate the individual configuration steps for integration in WinCC Unified.

The example control comes from the open source library "Gauge.js" from a third-party provider and is extended with code for use in WinCC Unified.



2.1.2. Principle of Operation

The complete custom web control has the following functions:

- Displaying a WinCC Unified tag value on a Gauge Control
- Defining upper and lower limits
- Define various style parameters for a better display
- Defining value ranges with different colors
- Method for flashing the area in which the value is currently located
- Event that occurs during the transition from one zone to another

Required knowledge

The application example assumes the following basic knowledge:

- Configuration with WinCC Unified
  The fundamentals are taught in the SITRAIN course "SIMATIC WinCC Unified 1 (TIA-UWCC1) - System Course".
  See article ID 109773211.
- Microsoft Visual Studio Code
- Web page programming with HTML5 and JavaScript

2.1.3. Components Used

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

Component	Number	Article number	Note
WinCC Unified V19	1	6AV2102-0AA02-3AA7	
Microsoft Visual Studio Code	1	-	See here
Gauge.js 1.3.8	1		See here \4\

The components listed here can be obtained from the Siemens Industry Mall.

2.2. Engineering

2.2.1. Configuration and Implementation of the Custom Web Control

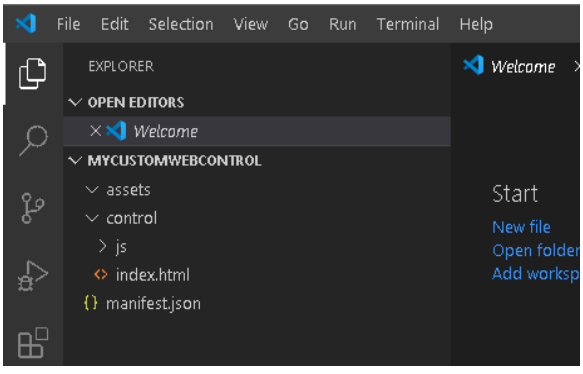
This subsection uses an example to show how to create a custom web control with Visual Studio Code.

NOTE Other text-based development environments can also be used.

If you're not very familiar with the topic, or if you're not familiar with Visual Studio Code software, for example, see Section 6, Helpful Information, for a brief introduction.

Create a custom web control

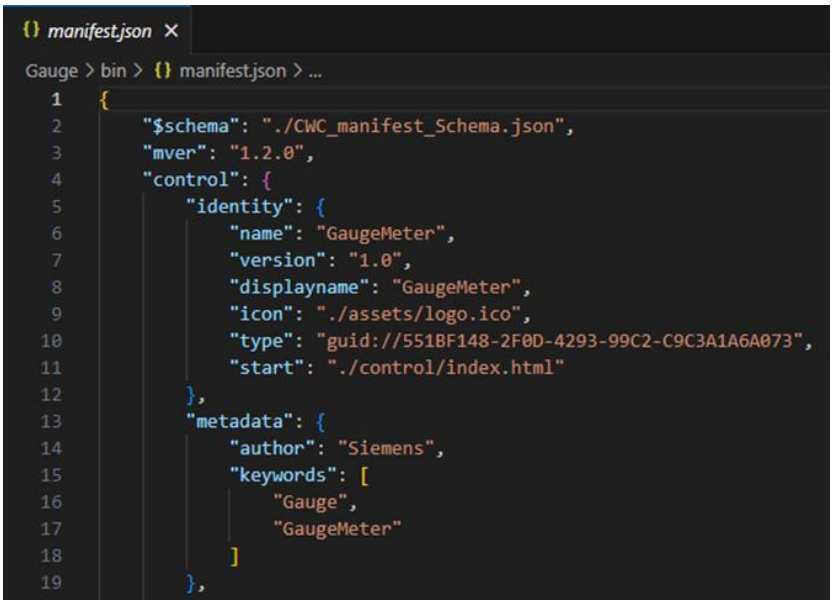
1. Create folder structure
   (See also the manual referred to in "General Configuration and Folder Structure.")
   Open Windows Explorer and create the following items in a working folder of your choice:
   a. File with the file name "manifest.json"
   b. Folder with the name "assets".
      Place an icon in any graphic format in this folder. It is displayed in the TIA Portal for this Control.
   c. Folder with the name "control".
      Create the files: "index.html", "code.js", "styles.css" and a folder "js" in which you store the file "webcc.min.js" from the attached files in this application example and the file "gauge.min.js" with the downloaded code (source: https://bernii.github.io/gauge.js/ \4\).
2. Open Visual Studio Code:
   a. Open Visual Studio Code (or a different editor).
   b. Select "File" > "Open Folder" to open the folder in which the folder structure you just created is stored.



3. Prepare the Manifest.json file:
   (See also the manual referred to in "Contract-Based Interaction and the Manifest File.")

   a. Open the Manifest.json file in Visual Studio Code (or in another editor). This file requires a certain structure, which is described in more detail in the manual. For this application example, it is sufficient to copy an existing file and make some adjustments.

   b. To create a new custom web control, copy the contents of the Gauge sample manifest.json file to your manifest.json.



   c. Give your custom web control a custom name under the "Name" attribute on lines 6 and 8.

   d. Customize the name of your logo on line 9 (all common graphic formats, such as JPG, PNG, BMP, etc. are possible).

   e. In line 10, assign a new, custom GUID. You can create one using an online generator like https://www.guidgenerator.com/.

   f. If necessary, adjust the metadata in line 13.

   g. If necessary, change the interface of your custom web controls from line 20. Possible data types include: "boolean", "number", "string", "array", or one of your own defined data types starting at line 80
   (See the included Manifest.json file for usage information.)

NOTE To ensure that you have created a valid JSON file, open it in Visual Studio Code or copy the content of the file into an online validation tool (such as https://jsonlint.com: Paste the content and click "Validate JSON" at the bottom left).

4. Index.html file (See also the manual referred to in "Interaction between Control and Container via API"):

   a. Open the Index.html file in Visual Studio Code (or in another editor). This file is the portal to your website. Here you also establish a connection to the WinCC Unified Runtime server for data exchange.

   b. The connection data for WinCC Unified can be found in the attached file "webcc.min.js". Move it to the "js" folder and reference the file as follows in index.html:
   `<script src='./js/webcc.min.js'></script>`
   This referencing is best done in your <Head> variable (see also line 11 of the attached example).

index.html

```
control > index.html > html
1   <!DOCTYPE html>
2   <html lang='en'>
3
4   <head>
5       <meta charset='utf-8'>
6       <title>GaugeMeter</title>
7       <meta name='viewport' content='width=device-width, initial-scale=1.0'>
8
9       <link rel='stylesheet' href='./styles.css' />
10
11      <script src='./js/webcc.min.js'></script>
12      <script src='./js/gauge.min.js'></script>
13
14  </head>
15
16  <body>
17      <canvas id='gauge'></canvas>
18
19      <script src='./code.js'></script>
20
21  </body>
22
23  </html>
```

c. The connection is made using the `WebCC.start()` function in file code.js (on line 241), which is included in the attached files.

```
239 ////////////////////////////////////////
240 // Initialize the custom control
241 WebCC.start(
242   // callback function; occurs when the connection is done or failed.
243   // "result" is a boolean defining if the connection was successfull or not.
244   function (result) {
245     if (result) {
246       console.log('connected successfully');
247       initializeGauge();
```

It is therefore important that the connection is established correctly when the page is accessed. This is best achieved if the function (as in the example) is located directly in the `<Script>` tag and not in a deeply nested function that may only be called at a later time (see previous Figure, line 19).

5. Data exchange between custom web control and WinCC
(See also the manual referred to in "Using the Control via WinCC"):

   a. You can now access the data defined in the Manifest.json file from anywhere in your application.

   b. Access is provided with the "`WebCC`" API object. You've already used it to make the connection. Now you have more options.

   c. For example, if you want to read or write properties (in the file Manifest.json under "properties" starting at line 42), you can access the "GaugeValue" property with write access as follows: `WebCC.Properties.GaugeValue = 5` sets the value to 5. The value of the linked WinCC variable is also set to 5.

   d. If you want to change the connected WinCC variables (perhaps a PLC variable), use the "`WebCC.onPropertyChanged.subscribe()`" function. You should call this function immediately after the connection is successfully established to get all the changes from the start (see lines 245 and 259). Use a function defined by you (callback function) as a transfer parameter. In this example, it is the function "setProperty" in the file code.js in line 125. With each data change, the "setProperty" function is called and a "data" object is passed that contains a "key" and a "value". The "key" is the name of the changed property and the "value" is the new value. It is therefore recommended to program branch points with switch-case for correct processing of the new value.


```
121  // This is a callback function that is called every time a contrac
122  // other functions so you can see the new value in the control.
123  // - data: object containing a key and a value property. The "key"
124  //         the "value" contains the new value.
125  function setProperty(data) {
126    // console.log('onPropertyChanged ' + data.key);  // uncomment t
127    switch (data.key) {
128      case 'GaugeValue':
129        updateValue(data.value);
130        break;
131      case 'GaugeBackColor':
132        document.body.style.backgroundColor = toColor(data.value);
```

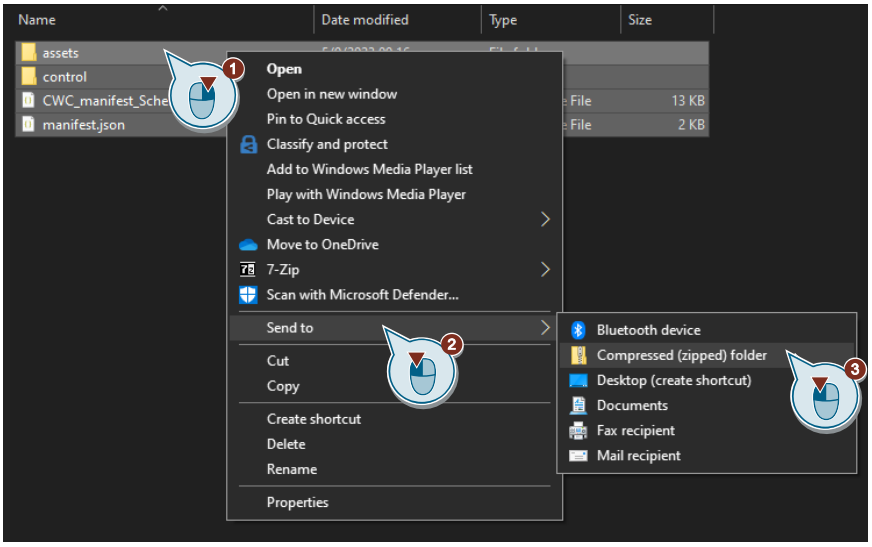
e. If you have declared methods in the Manifest.json file (line 22 in the "manifest.json" file), WinCC can call these methods and you can react to them in the custom web control. WinCC can call these methods at any time. This means that you must always define what should happen before making the connection. You define a function with the exact name you specified in the Manifest.json file and also the same parameters. You can find an example of this in the attached code.js file on line 268.

f. If you have defined an event in the file Manifest.json (see line 32), you can fire that event anywhere in your code with WebCC.Events.fire() so that WinCC is notified. In this case, the first transfer parameter is always the name of the event that you want to trigger, followed by all transfer parameters in the correct order that you have specified in the Manifest.json file. You can find an example of this in the attached code.js file in line 69.

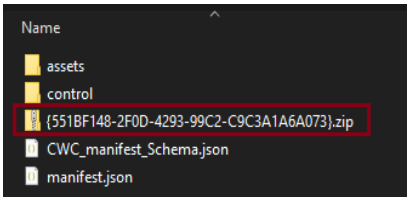
6. Deployment process for using the custom web control in the TIA Portal (see also the manual referred to in "Creating the ZIP file"):

   a. When you have finished programming, you still need to package the code so that the TIA Portal recognizes your custom web control correctly.

   b. To do this, open Windows Explorer and go to your project folder. Select the originally created folders "assets", "control, the file 'CWC_manifest_Schema.json' (for this file see Section 2.2.4) and the file 'manifest.json' and archive them by right-clicking (1) > "Send to" (2) > "Compressed (zipped) folder" (3).



   c. Use your GUID file from the Manifest.json file to change the name of your created .zip file as follows (the Xs are your GUID):
   `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip`
   Please note the curly brackets before and after the GUID.



d. Your custom web control is now complete and ready for use in the TIA Portal.

2.2.2. Installation and Integration into the User Project

Installation in the TIA Portal

Before you can use the custom web control in the TIA Portal, you have two procedures for installing it (see also the official manual under "Installing a custom web control"):

1. Only make available for a specific project
   Add your custom web control to your project folder:
   "...Project_1\UserFiles\CustomControls\ {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip"
2. Make available for all projects
   Add your custom web control to the TIA Portal installation path:
   "C:\Program Files\Siemens\Automation\Portal Vxx\Data\Hmi\CustomControls\ {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip"

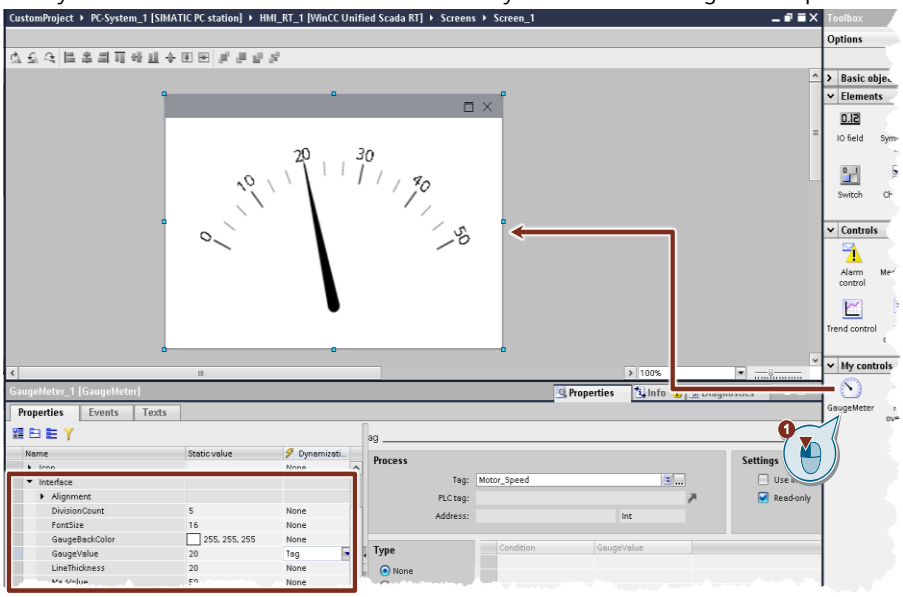
NOTE If you copy the project to another PC, the custom web controls will not be submitted, and a compilation error will occur in the TIA Portal project. You will also need to copy the custom web controls to the new computer and to the directory specified above.

NOTE When you upload a project to the Operator Panel, the TIA Portal also transmits your custom web control. There is no installation on the Runtime server.

Integration into the user's project

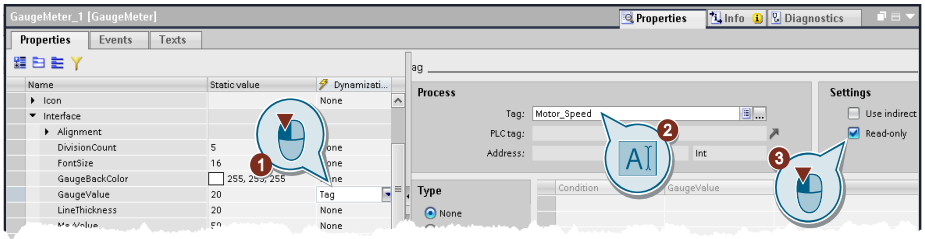
To configure your custom web control in the TIA Portal project, proceed as follows:

1. Open an screen for your Unified Comfort Panel or PC station.
2. Click your custom web control under "Tools > My Controls" and drag and drop it onto the screen.



In the properties of your custom web controls, under "Interface", you will find all the properties that you have defined in the Manifest.json file. As with all other screen object properties, you can assign a static value to these or define dynamization.

3. Dynamize custom web control
   a. Create a dynamization "tag" for "GaugeValue".
   b. Select a compatible PLC or HMI tag.
   c. If your custom web control only has read access to the tag, the box should remain checked. However, if your implementation provides for the custom web control to change your tags, then deactivate the checkbox. In this case, the "GaugeMeter" only displays the tag (read).



2.2.3. Debugging

In this subsection, you will learn how to debug custom web controls in Runtime.

Debugging is not possible in the TIA Portal. If you cannot transfer the custom web control to Runtime because you have previously encountered problems, read Section 6 Common Error Patterns for options.

Debugging in Google Chrome

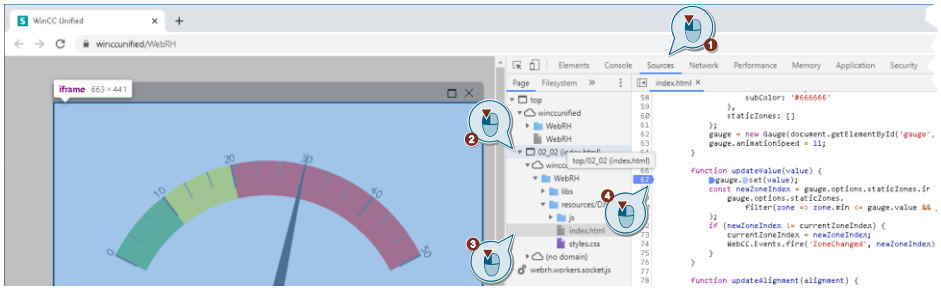
The WinCC Unified Runtime sends the custom web control directly to the web client when it is used, so that you can also debug it on the web client using the browser's standard developer console.

Open the Unified Runtime screen containing the custom web control in Google Chrome and press F12 to open the developer console.

Step-by-step instructions

In the figure below you can see how debugging works after opening the developer console with F12.

1. Select the "Sources" tab in the developer console.
2. Move the mouse pointer slowly over the folders under the "top" node on the left-hand side of the developer console. Your custom web control does not have a descriptive name here, but it will be highlighted in blue by your browser if you have selected it. (here you will find a separate folder for each custom web control instance).
3. Open the corresponding folder and go to the file you want to debug. (all code is contained in the index.html file).
4. Scroll to the corresponding position in the permission window and click the line number to insert a breakpoint. When the code reaches this position again, it stops and you receive detailed information about the program sequence. In the figure, there is a breakpoint at the beginning of the "updateValue" function. The script will now always stop here when you update the linked value. Remove the breakpoint by clicking the line number again.



NOTE For more information about working with the developer console and the information you can read, please refer to Google's official documentation: https://developers.google.com/web/tools/chrome-devtools/javascript#sources-ui

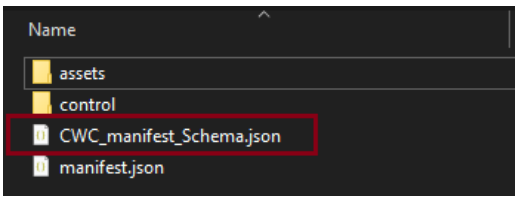
NOTE Note You can also carry out debugging with any other browser, simply read the documentation for the respective browser.

2.2.4. Auto-Complete During Programming

In this Section you will learn how to activate auto-complete.

JSON Manifest Schema

To support the creation of the "manifest.json", you need a file that specifies the schema to enable auto-completion (see step 3 in Section 2.2.1). This is then placed at the same file level as the "manifest.json" file. To do this, copy the file "CWC_manifest_Schema.json" to the "manifest.json"



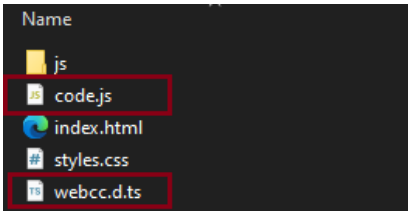
Now you need to specify the path of the template in the "manifest.json". It is sufficient to specify the path to the template immediately after starting the file. The parameter "$schema" is used for this purpose. In this case: insert the following line after the first opening curly bracket:

```
"$schema": "./CWC_manifest_Schema.json"
```

Save the changes. You can now use the auto complete.

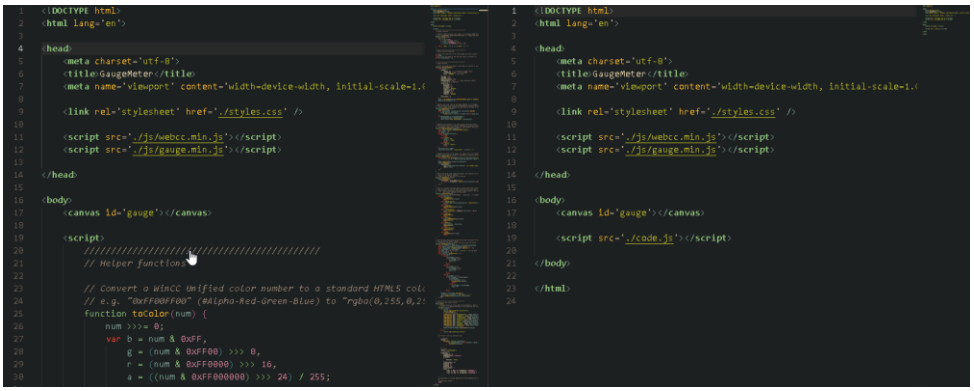
Scripting support with webcc.d.ts

For the auto-completion in the script part, you need the webcc.d.ts file (see steps 4 and 5 of Section 2.2.1). It contains the relevant information for the WebCC object and must be in the same folder as the index.html file. This part of the script is included in the code.js file because Visual Studio Code's scripting support only works in a .js file, not in the index.html.



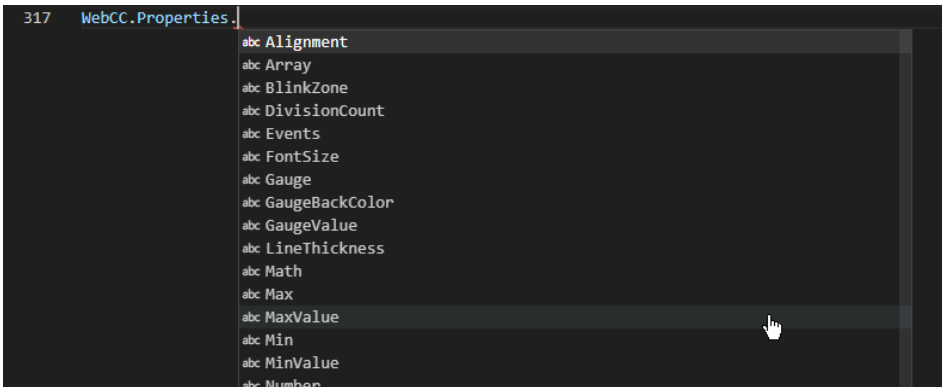
The CWC must be referenced in order to be able to locate scripts in the "code.js". You can do this with the following code snippet:

```
<script src='./code.js'></script>
```



Result

As a result, you get auto complete and a clear separation of the scripts.

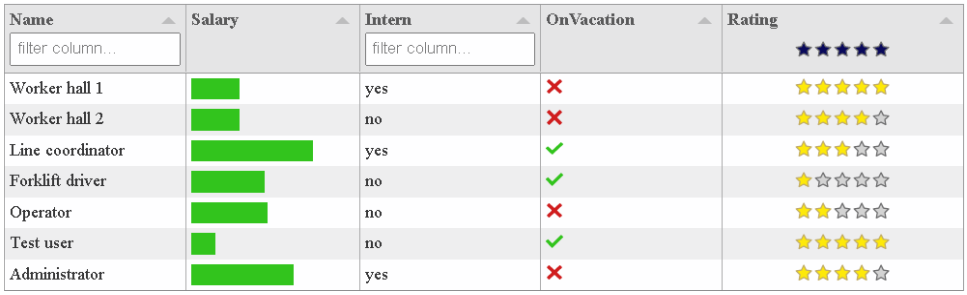


3. "Table" Custom Web Control

3.1. Introduction

3.1.1. Overview

The following is another application example for creating custom web controls. The custom web control "Table" is a tabular display of values. You can not only display the values, but also format them graphically (see figure below).



For this example, the JavaScript library "Tabulator" (see http://tabulator.info) was used and supplemented with code so that it can be used in WinCC Unified.

3.1.2. Range of Functions

With the "Table" custom web control you have the following options:

- Create a table based on submitted WinCC Unified variable values (e.g., process values, parameter setpoints)
- Save all table values in a string tag
- Customize table columns individually (width, display format of the values, filter options for the values)

3.1.3. Components Used

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

Component	Number	Article number	Note
WinCC Unified V19	1	6AV2102-0AA02-3AA7	
Microsoft Visual Studio Code	1	-	See here
Tab v6.2	1		See here \5\

The components listed here can be obtained from the Siemens Industry Mall.

3.1.4. Application Examples

The tables of the custom web control are suitable for the following purposes, among others:

- Output of the values of a CSV file as a table in WinCC Unified Runtime.
- Graphical representation of the process values within a table.
- Exporting the process values as a CSV file.

3.2. Engineering

3.2.1. Configuration and Implementation of the Custom Web Control

In this subsection you will learn how to create and implement the custom web control "Table" with Visual Studio Code.

Creating a custom web control

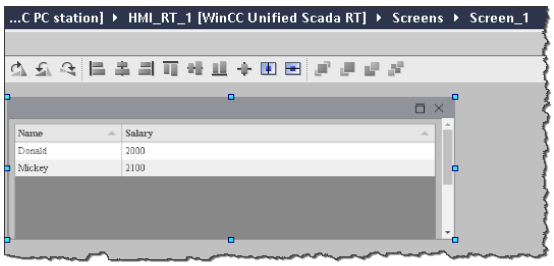
1. Create a folder structure (see also in the manual under: "General layout and folder structure").
   Open Windows Explorer and create the following items in a working folder of your choice:
   a. File with the file name "manifest.json"
      (This file requires a certain structure, which is described in more detail in the manual. For this application example, it is sufficient to copy an existing file and configure adjustments as required).
   b. Folder with the name "assets".
      Place an icon in any graphic format in this folder. It is displayed in the TIA Portal for this Control.
   c. Folder with the name "control".
   d. In the "control" folder, create the "index.html" and "code.js" files and add the "webcc.min.js" file from the attached files to this directory. Unzip the downloaded "Tabulator" library (source: https://github.com/olifolkerd/tabulator) into the "control" folder.
2. Visual Studio Code:
   a. Open Visual Studio Code or another editor of your choice.
   b. Select "File" > "Open Folder" to open the folder in which the folder structure you just created is stored.

3. Create the "Manifest.json" file:
   a. Open the Manifest.json file.
   b. Set up your manifest file as described in Section 2.2.1.
   c. Do not make any changes to the TableDataString property. The table contents are written back here.
4. Configure Index.js
   Reference the required scripts.

```
<script src='webcc.min.js'></script> <!-- mandatory dependency -->
<script type="text/javascript" src="./dist/js/tabulator.min.js"></script>
<link href="./dist/css/tabulator.css" rel="stylesheet">
```

5. Configure code.js:
   a. Start the connection setup with the function "WebCC.start();".
   b. As soon as the connection is established, the current value hidden behind "TableDataString" is written to the "ArrayData" array.
   c. Finally, you will see the "subscribe" function. It checks whether the value "TableDataString" changes. This happens automatically when the page is loaded. When the page is reloaded, the value is set to the "static" value from the TIA Portal project. As soon as this happens, the value is overwritten again with the current values of "ArrayData".

6. Work with "Tabulator"
   a. The "ArrayData" field now contains the data points from the table. You can use these to create tables with the "Tabulator" function. (More information is available at http://tabulator.info/)
   b. The interface tag "ColumnStyleString" defines the properties of the table columns.
7. Deployment process and installation:
   a. When you have finished programming, create a *.zip file as shown in the previous "Gauge" example.
   The .zip file should contain the "assets" and "control" folders, the "CWC_manifest_Schema.json" file (see Sec. 2.2.4 for more information on this file) and the "manifest.json" file. Give the *.zip file its own GUID name (`{xxxxxxxx-xxxx-xxxx-xxxxxxxx}.zip`).
   b. Now make this file available to your user project by copying it to the "CustomControls" folder of your project.
   `…Project_1\UserFiles\CustomControls\{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip`
   Open your TIA Portal project and drag your custom web control onto your desired screen.



   c. Then transfer the corresponding data via script to the "ColumnStyleString" and "TableDataString" properties of the custom web control.
   Further information on this can be found in the Necessary Data section and in the "Table" Custom Web Control section.

3.2.2. Installation and Integration into the User Project

The subsections Installation and Integration into the User Project describe how to install the custom web control in the TIA Portal and integrate it into your user program.

3.2.3. Debugging

In this subsection, you will learn how to debug custom web controls in Runtime.

Debugging is not possible in the TIA Portal. If you cannot transfer the custom web control to Runtime because you have previously encountered problems, read Section 6 Common Error Patterns for options.

3.2.4. Necessary Data

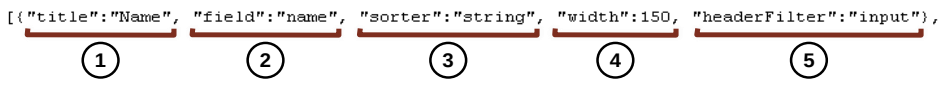
To create a table, two strings must be passed to the following two properties of the "Table" custom web control:

- "ColumnStyleString": String for formatting the columns
- TableDataString String with data entries (table content)

ColumnStyleString

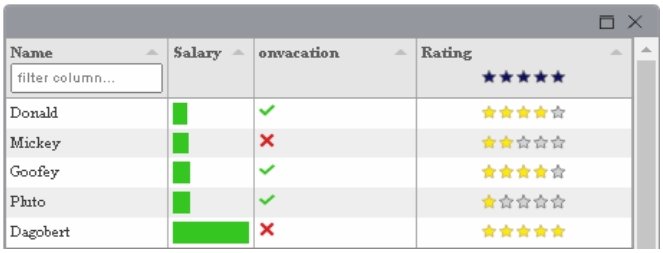
"ColumnStyleString" (string in JSON format) defines the layout of the table columns.

An example of a "ColumnStyleString" structure for the "Name" column is shown in the following figure.



No.	Parameter	Description
1.	title	Defines the name of the column in the custom web control
2.	field	Used to assign the values (from the "TableDataString") when filling the table. **Note:** Make sure that the spelling of the values is correct, otherwise they cannot be assigned to the "title" column.
3.	sorter	Specifies the attribute by which the list is sorted.
4.	width	Defines the column width.
5.	headerFilter	Optional: Creates a filter in the column header.

You can not only simply display the values, but also format them graphically. If necessary, specify this using additional parameters, such as "hozAlign", "formatter" and "formatterParams".



NOTE Note See the following link for a detailed description of the formatting options: http://tabulator.info/docs/4.9/format

TableDataString

The "TableDataString" (string in JSON format) contains the values with which the individual columns are to be filled. Its structure consists of the "field" name of the "ColumnDataString" and the value that is to be inserted in the corresponding table row.

NOTE Note In Section 4.1 "Gauge" Custom Web Control usage, the build of "ColumnStyleString" and "TableDataString" in the application is explained again using the sample project.

You can find more information about the properties at http://tabulator.info/.

NOTE Note If a parameter is empty or a control is not connected correctly, the custom web control does not display the table correctly (see Section 6.2).

4. "Toolbox" Custom Web Control

4.1. Introduction

4.1.1. Overview

The figure shows the icon of the "Toolbox" custom web control. This becomes visible as soon as the control has been successfully added to the TIA project.

The control can be downloaded via the GitHub link of the article page in SIOS. It is constantly maintained and developed.

4.1.2. Principle of Operation

The full custom web control has the following methods:

- Download: Downloads the content to the user in the specified file name
- switchUrl: Changes the current URL to change the Unified website to another website
- takeScreenshot: Takes a screenshot of the website and downloads it as a PNG to the client device
- openNewWindow: opens a new window with the given URL and the specified width and height
- closeWindow: Closes the oldest window opened by openNewWindow
- openScreenOnMonitorX: opens new windows with the screens specified in the interface values
- checkFullscreen: checks the URL and the specified interface values for the transition to fullscreen mode, only works together with openScreenOnMonitorX
- playSound: plays a sound file in the client browser, e.g., as an alarm horn
- openNewTab: opens the specified URL in a new tab

Required knowledge

The application example assumes the following basic knowledge:

- Configuration with WinCC Unified
  The fundamentals are taught in the SITRAIN course "SIMATIC WinCC Unified 1 (TIA-UWCC1) - System Course".
  See article ID 109773211.
- Microsoft Visual Studio Code
- Web page programming with HTML5 and JavaScript

4.1.3. Components Used

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

Component	Number	Article number	Note
WinCC Unified V19	1	6AV2102-0AA02-3AA7	
Microsoft Visual Studio Code	1	-	See here
Toolbox	1		See github \6\

The components listed here can be obtained from the Siemens Industry Mall.

4.2. Engineering

4.2.1. Configuration and Implementation of the Custom Web Control

This subsection uses an example to show how to create a custom web control with Visual Studio Code.

NOTE Other text-based development environments can also be used.

If you are not very familiar with the topic, or if you are not familiar with the Visual Studio Code software, for example, see Section 7 Helpful Information, for a brief introduction.

Create a custom web control

1. Create folder structure

   (See also the manual referred to in "General Configuration and Folder Structure.")

   a. Open Windows Explorer and create the following items in a working folder of your choice:

   b. File with the file name "manifest.json"

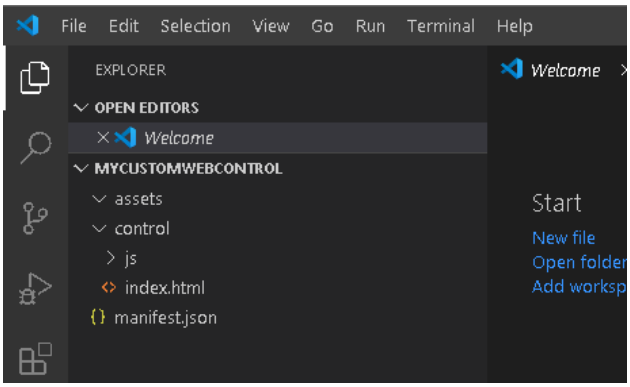
   c. Folder with the name "assets".
   Place an icon in any graphic format in this folder. It is displayed in the TIA Portal for this Control.

   d. Folder with the name "control".
   Create the files: "index.html", "code.js", "styles.css", and a folder "js" in which you will place the file "webcc.min.js" from the attached files in this application example.

2. Open Visual Studio Code:

   a. Open Visual Studio Code (or a different editor).

   b. Select "File" > "Open Folder" to open the folder in which the folder structure you just created is stored.



3. Prepare the Manifest.json file:
   (See also the manual referred to in "Contract-Based Interaction and the Manifest File")

   a. Open the Manifest.json file in Visual Studio Code (or in another editor)
   This file requires a certain structure, which is described in more detail in the manual. For this application example, it is sufficient to copy an existing file and make some adjustments.

   b. To create a new custom web control, copy the content of the manifest.json file of the toolbox example into your manifest.json.

   c. Give your custom web control a custom name under the "Name" attribute on lines 6 and 8.

   d. Customize the name of your logo on line 9 (all common graphic formats, such as JPG, PNG, BMP, etc. are possible).

   e. In line 10, assign a new, custom GUID. You can create one using an online generator such as https://www.guidgenerator.com/.

   f. If necessary, adjust the metadata in line 13.

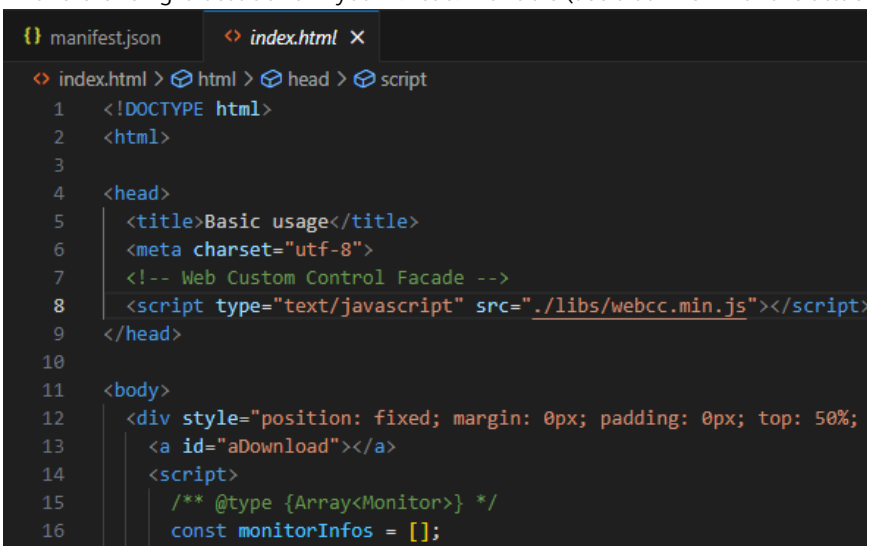
   g. If necessary, change the interface of your custom web controls from line 20. Possible data types include: "boolean", "number", "string", "array", or one of your own defined data types starting at line 80
   (See the included Manifest.json file for usage information.)

NOTE To ensure that you have created a valid JSON file, open it in Visual Studio Code or copy the content of the file into an online validation tool (such as https://jsonlint.com: Paste the content and click "Validate JSON" at the bottom left).

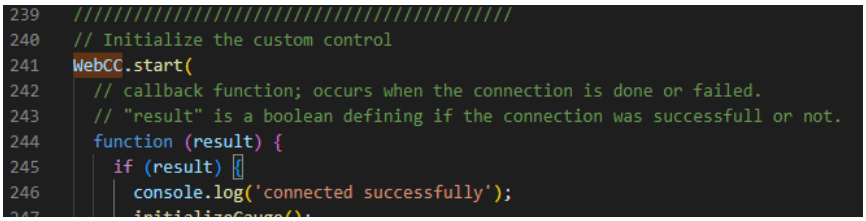
4. Index.html file (See also the manual referred to in "Interaction between Control and Container via API"):

   a. Open the Index.html file in Visual Studio Code (or in another editor).
   This file is the portal to your website. Here you also establish a connection to the WinCC Unified Runtime server for data exchange.

   b. The connection data for WinCC Unified can be found in the attached file "webcc.min.js". Move it to the "js" folder and reference the file as follows in index.html:
   `<script src='./js/webcc.min.js'></script>`
   This referencing is best done in your <Head> variable (see also line 11 of the attached example).



   c. The connection is made using the `WebCC.start()` function in file code.js (on line 241), which is included in the attached files.



   d. It is therefore important that the connection is established correctly when the page is accessed. This is best achieved if the function (as in the example) is located directly in the <Script> tag and not in a deeply nested function that may only be called at a later time (see previous Figure, line 19).

5. Data exchange between custom web control and WinCC
   (See also the manual referred to in "Using the Control via WinCC"):

   a. You can now access the data defined in the Manifest.json file from anywhere in your application.

   b. Access is provided with the "`WebCC`" API object. You've already used it to make the connection. Now you have more options.

   c. For example, if you want to read or write properties (in the file Manifest.json under "properties" starting at line 34), you can access the "TabTitle" property with write access as follows: `WebCC.Properties.TabTitle = ""` sets the value to `""`. The value of the linked WinCC variable is also set to `""`.

   d. If you want to change the connected WinCC variables (perhaps a PLC variable), use the "`WebCC.onPropertyChanged.subscribe()`" function. You should call this function immediately after the connection is successfully established to get all the changes from the start. Use a function defined by you (callback function) as a transfer parameter. In this example, it is the function "setProperty" in the file code.js in line 125. With each data change, the "setProperty" function is called and a "data" object is passed that contains a "key" and a "value". The "key" is the name of the changed property and the "value" is the new value. It is therefore recommended to program branch points with switch-case for correct processing of the new value.

```
121  // This is a callback function that is called every time a contrac
122  // other functions so you can see the new value in the control.
123  // - data: object containing a key and a value property. The "key"
124  //         the "value" contains the new value.
125  function setProperty(data) {
126    // console.log('onPropertyChanged ' + data.key);  // uncomment t
127    switch (data.key) {
128      case 'GaugeValue':
129        updateValue(data.value);
130        break;
131      case 'GaugeBackColor':
132        document.body.style.backgroundColor = toColor(data.value);
```

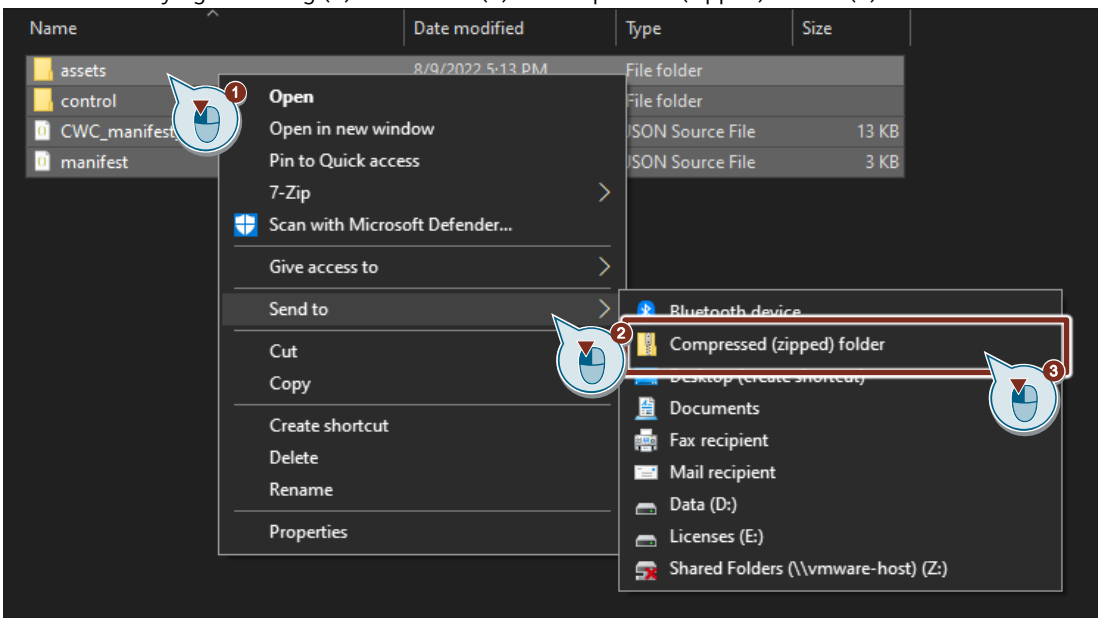
   e. If you have declared methods in the Manifest.json file (line 35 in the "manifest.json" file), WinCC can call these methods and you can react to them in the Custom Web Control. WinCC can call these methods at any time. This means that you must always define what should happen before making the connection. You define a function with the exact name you specified in the Manifest.json file and also the same parameters.

   f. If you have defined an event in the file Manifest.json (see line 116), you can fire that event anywhere in your code with WebCC.Events.fire`()` so that WinCC is notified. In this case, the first transfer parameter is always the name of the event that you want to trigger, followed by all transfer parameters in the correct order that you have specified in the Manifest.json file. You will find an example of this in the attached index.html file in line 422.

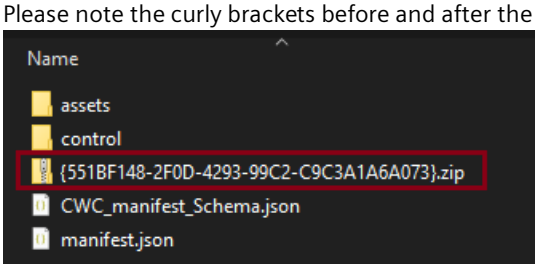
6. Provisioning process for using the custom web control in the TIA Portal
   (See also the manual referred to in "Creating the ZIP file"):

   a. When you have finished programming, you still need to package the code so that the TIA Portal recognizes your custom web control correctly.

   b. To do this, open Windows Explorer and go to your project folder. Select the originally created folders "assets", "control, the file 'CWC_manifest_Schema.json' (for this file see Section 2.2.4) and the file "manifest.json" and archive them by right-clicking (1) > "Send to" (2) > "Compressed (zipped) folder" (3).



   c. Use your GUID file from the Manifest.json file to change the name of your created .zip file as follows (the Xs are your GUID):
   `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip`
   Please note the curly brackets before and after the GUID.



   d. Your custom web control is now complete and ready for use in the TIA Portal.

4.2.2. Installation and Integration into the User Project

Installation in the TIA Portal

Before you can use the custom web control in the TIA Portal, you have two procedures for installing it (see also the official manual under "Installing a custom web control"):

1. Only make available for a specific project
   Add your custom web control to your project folder:

   "...Project_1\UserFiles\CustomControls\ {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip"

2. Make available for all projects
   Add your custom web control to the TIA Portal installation path:

   "C:\Program Files\Siemens\Automation\Portal Vxx\Data\Hmi\CustomControls\ {xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip"

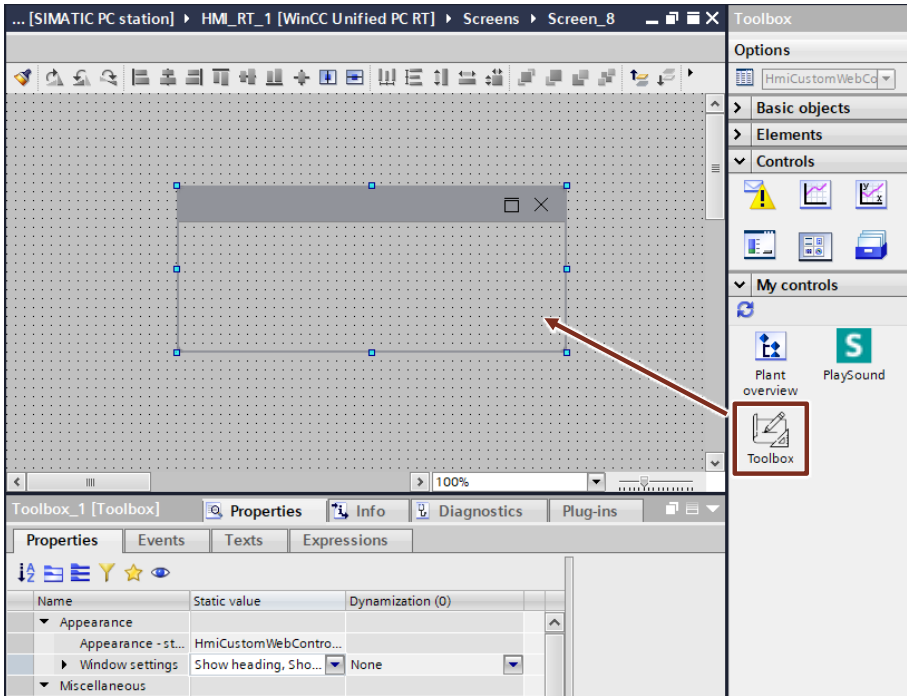
NOTE If you copy the project to another PC, the custom web controls will not be submitted, and a compilation error will occur in the TIA Portal project. You will also need to copy the custom web controls to the new computer and to the directory specified above.

NOTE When you upload a project to the Operator Panel, the TIA Portal also transmits your custom web control. There is no installation on the Runtime server.

Integration into the user's project

To configure your custom web control in the TIA Portal project, proceed as follows:

1. Open a screen for your Unified Comfort Panel or PC station.
2. Click your custom web control under "Tools > My Controls" and drag and drop it onto the screen.



In the properties of your custom web controls, under "Interface", you will find all the properties that you have defined in the Manifest.json file. As with all other screen object properties, you can assign a static value to these or define dynamization.

4.2.3. Debugging

In this subsection, you will learn how to debug custom web controls in Runtime.

Debugging is not possible in the TIA Portal. If you cannot transfer the custom web control to Runtime because you have previously encountered problems, read Section 6 Common Error Patterns for options.

Debugging in Google Chrome

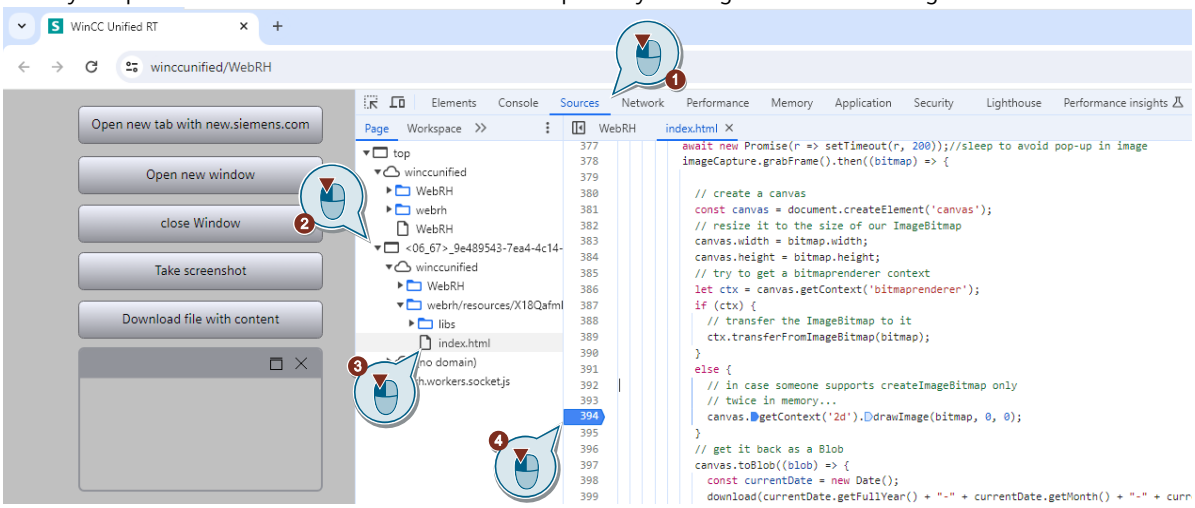
WinCC Unified Runtime sends the custom web control directly to the Web Client when in use. This way, you can also debug it on the Web Client using the standard developer console of the browser.

Open the Unified Runtime screen containing the custom web control in Google Chrome and press F12 to open the developer console.

Step-by-step instructions

Figure 2-11 shows how debugging works after opening the developer console with F12.

1. Select the "Sources" tab in the developer console.
2. Move the mouse pointer slowly over the folders under the "top" node on the left-hand side of the developer console. Your custom web control does not have a descriptive name here, but it will be highlighted in blue by your browser if you have selected it. (here you will find a separate folder for each custom web control instance).
3. Open the corresponding folder and go to the file you want to debug. (all code is contained in the index.html file).
4. Scroll to the corresponding position in the permission window and click the line number to insert a breakpoint. When the code reaches this position again, it stops and you receive detailed information about the program sequence. In the figure, there is a breakpoint at the beginning of the "updateValue" function. The script will now always stop here when you update the linked value. Remove the breakpoint by clicking the line number again.



NOTE

For more information about working with the developer console and the information you can read, please refer to Google's official documentation: https://developers.google.com/web/tools/chrome-devtools/javascript#sources-ui

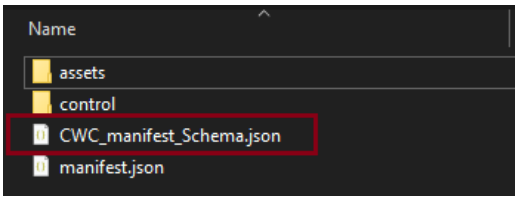
4.2.4. Auto-Complete During Programming

NOTE You can also carry out debugging with any other browser, simply read the documentation for the respective browser.

In this Section you will learn how to activate auto-complete.

JSON Manifest Schema

To support the creation of the "manifest.json", you need a file that specifies the schema to enable auto-completion (see step 3 in Section 4.2.1). This is then placed at the same file level as the "manifest.json" file. To do this, copy the file "CWC_manifest_Schema.json" to the "manifest.json".



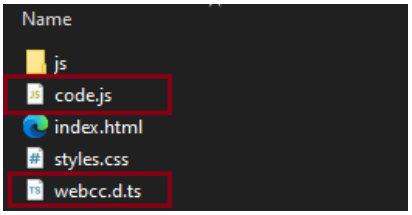
Now you need to specify the path of the template in the "manifest.json". It is sufficient to specify the path to the template immediately after starting the file. The parameter "$schema" is used for this purpose. In this case: insert the following line after the first opening curly bracket:

```
"$schema": "./CWC_manifest_Schema.json"
```

Save the changes. You can now use the auto-complete.

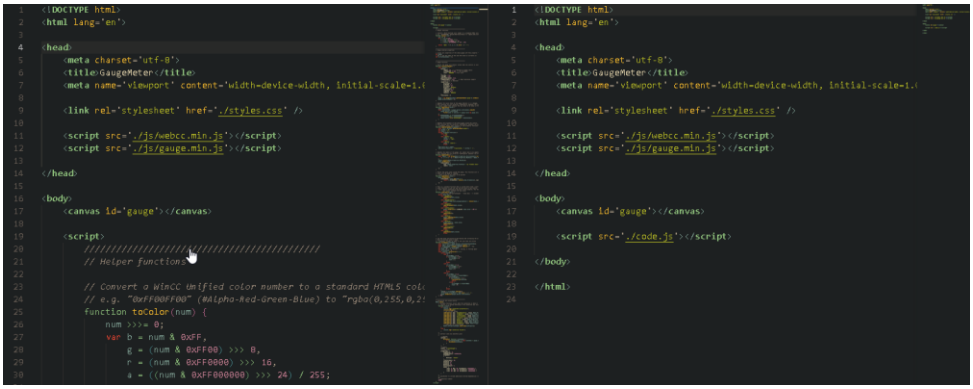
Scripting support with webcc.d.ts

For auto-completion in the script part, you need the webcc.d.ts file (see steps 4 and 5 of Section 4.2.1). It contains the relevant information for the WebCC object and must be in the same folder as the index.html file. This part of the script is included in the code.js file because Visual Studio Code's scripting support only works in a .js file, not in the index.html.



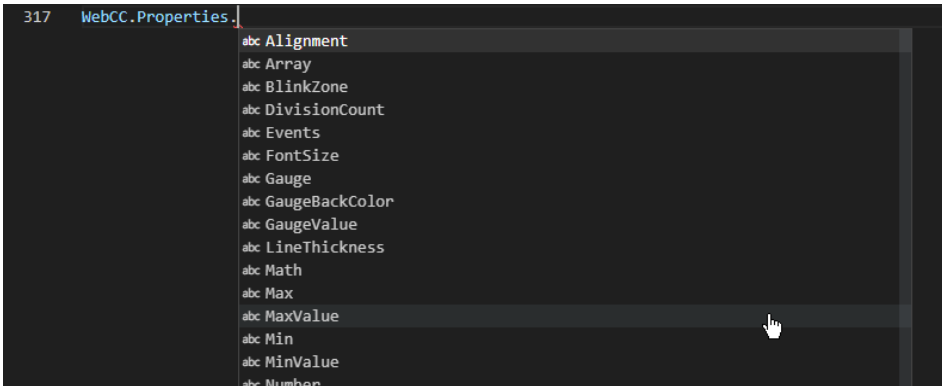
The CWC must be referenced in order to be able to locate scripts in the "code.js". You can do this with the following code snippet:

```
<script src='./code.js'></script>
```



Result

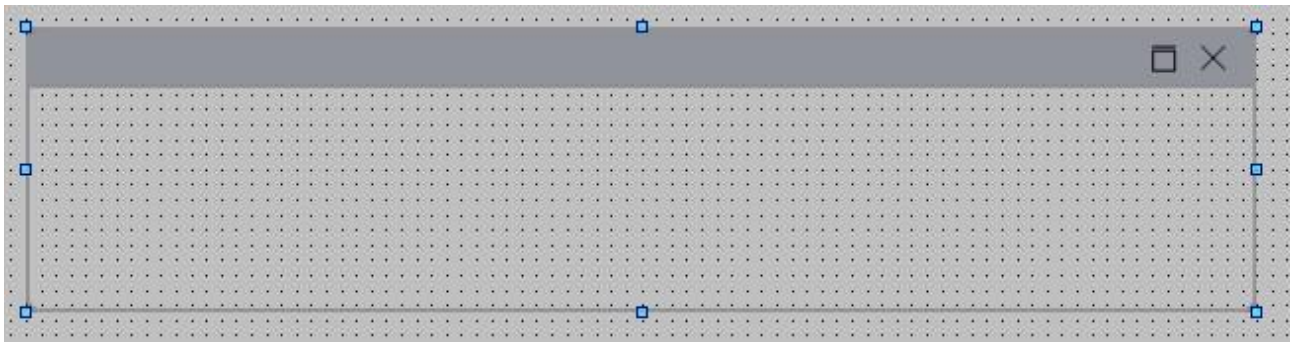
As a result, you get auto-complete and a clear separation of the scripts.



4.3. Operation

4.3.1. Overview

The custom web control (CWC) "Toolbox" is invisible because it has no user interface. When used in the project, the border and header bar can be hidden from the window settings. It is also possible to set the CWC to invisible in the settings. This custom web control is a collection of useful functions that extend the functionality of WinCC Unified.



The control has the following features:

Properties

- Screens: Specifies the screens, with their position and size, to be opened by openScreenOnMonitorX
- TabTitle: This property allows you to customize and override the title of the browser tab. You can define a static value or even dynamically design the tab title to display different values in different states of the machine.

Methods

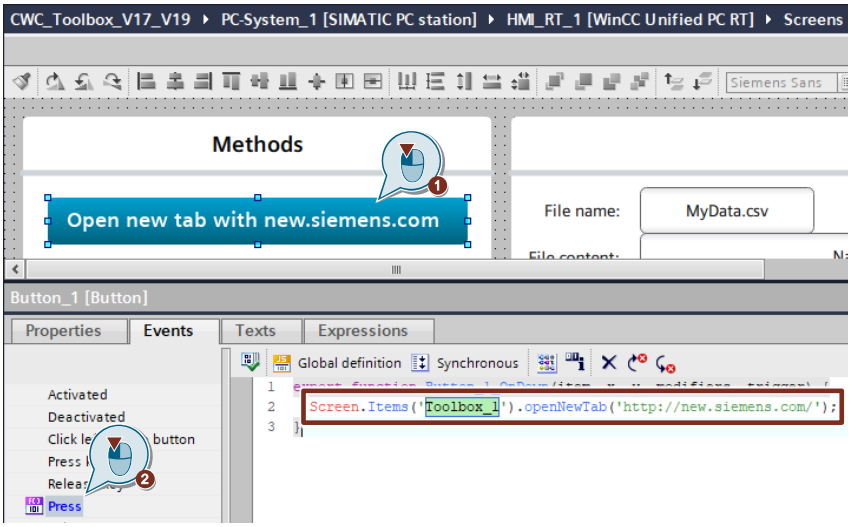
- Download: Downloads the content to the user in the specified file name
- switchUrl: Changes the current URL to change the Unified website to another website
- takeScreenshot: Takes a screenshot of the website and downloads it as a PNG to the client device
- openNewWindow: opens a new window with the given URL and the specified width and height
- closeWindow: Closes the oldest window opened by openNewWindow
- openScreenOnMonitorX: opens new windows with the screens specified in the interface values
- checkFullscreen: checks the URL and the specified interface values for the transition to fullscreen mode, only works together with openScreenOnMonitorX
- playSound: plays a sound file in the client browser, e.g., as an alarm horn
- openNewTab: opens the specified URL in a new tab

Events

- SizeChanged: triggers whenever the screen width or height changes
- URL: triggers with the creation of the CWC
- Touched: triggers
- UnifiedInContainer: triggers with the creation of the CWC
- IsMonitorMode: triggers when the CWC is created

4.3.2. Application of the Methods

The methods of the control can be triggered in the TIA Portal via scripting. A button can be used for this. The methods can then be called via any event. In this case, the "openNewTab" method is triggered via the button's "Press" event. To trigger the methods, the CWC must be properly referenced as shown below. So it should be included on the respective screen.



The following describes all the methods of the "Toolbox" custom web control.

Download Method

The download method creates a new text file with the name "Protocol.txt" (can also be .csv) and the content "2020-01-01 Happy New Year...". If the second parameter is missing, this file is searched for on the internet and downloaded directly into the client's browser so that it can be opened on the device (e.g., tablet, smartphone, PC).

```
Screen.Items('Toolbox').Download('Protocol.txt', '2020-01-01  Happy New Year \n 2020-01-02  Machine is up and running \n');
```

openNewTab Method

The openNewTab method opens a new tab in the client's browser with the specified URL, e.g., http://siemens.com/.

```
Screen.Items('Toolbox').openNewTab('http://siemens.com/');
```

openNewWindow Method

This method opens a new browser window with the specified URL, e.g., http://siemens.com/ and the specified width and height in pixels.

```
Screen.Items('Toolbox').openNewWindow('http://siemens.com/', 1920, 1080);
```

closeWindow Method

The closeWindow method closes the oldest window opened by openNewWindow().

```
Screen.Items('Toolbox').closeWindow();
```


takeScreenshot Method

The takeScreenshot method takes a screenshot of the current user view in the browser as a PNG. This PNG is downloaded directly to the client's browser so that it can be opened on the device (e.g., tablet, smartphone, PC).

```
Screen.Items('Toolbox').takeScreenshot();
```

openScreenOnMonitorX Method

The openScreenOnMonitorX method checks your monitors and the window placement API. It will then open as many new windows as there are screens specified in the interface. The new windows will all open with the screen of the CWC, but the URLs are different, to access the new screens, call this function in the "Url" event.

```
Screen.Items('Toolbox').openScreenOnMonitorX(url, actScreen);
```

checkFullscreen Method

The checkFullscreen method helps to get into fullscreen mode, requires user action (click anywhere on the screen), and only works with changed URL of the openScreenOnMonitorX method.

```
Screen.Items('Toolbox').checkFullscreen();
```

playSound Method

The playSound method plays a standard sound file in your client browser. This can be used as an alarm horn, for example. Together with AlarmSubscriptions, the sound can also be triggered by an attached alarm. It is possible to provide several sound files in the CWC. To do this, the respective sound file must be made available in the CWC folder under **"***./Control/dist/media***"**.

```
Screen.Items('Toolbox').playSound("./dist/media/Alarm01.wav");
```

switchUrl Method

This method swaps the current URL with the specified URL and presses Enter.

```
Screen.Items('Toolbox').switchUrl('http://siemens.com/');
```


4.3.3. Configuration of the Interface

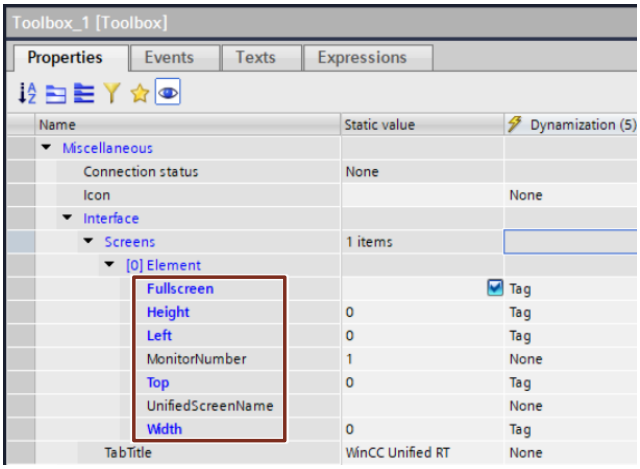
The behavior of the custom web control can also be modified by the following properties from the manifest file. The function "openScreenOnMonitorX" checks the entries from the interface and can be configured via them. These characteristics are explained in more detail below. They can also be configured from the TIA Portal via the properties of the cwc.

```
"ScreenInfo": {
    "type": "object",
    "properties": {
        "UnifiedScreenName": {
            "type": "string",
            "default": ""
        },
        "MonitorNumber": {
            "type": "integer",
            "default": 0
        },
        "Fullscreen": {
            "type": "boolean",
            "default": false
        },
        "Height": {
            "type": "integer",
            "default": 0
        },
        "Width": {
            "type": "integer",
            "default": 0
        },
        "Left": {
            "type": "integer",
            "default": 0
        },
        "Top": {
            "type": "integer",
            "default": 0
```

The screens contain the information for new windows to open at a defined position and size, as well as a screen to switch to.

- Fullscreen: if enabled, it ignores the height, left, top, and width and attempts to switch to full-screen mode.
- Height: specifies the height of the new browser window in pixels.
- Left: specifies the x position within the monitor screen where the new browser window should be placed.
- MonitorNumber: specifies the monitor on which to place the new browser window. The monitor at the top left is number 1 and the bottom right is the last number.
- Top specifies the Y position within the monitor screen where the new browser window should be placed.
- UnifiedScreenName: specifies the Unified screen to open in the new browser window.
- Width: specifies the width of the new browser window in pixels.

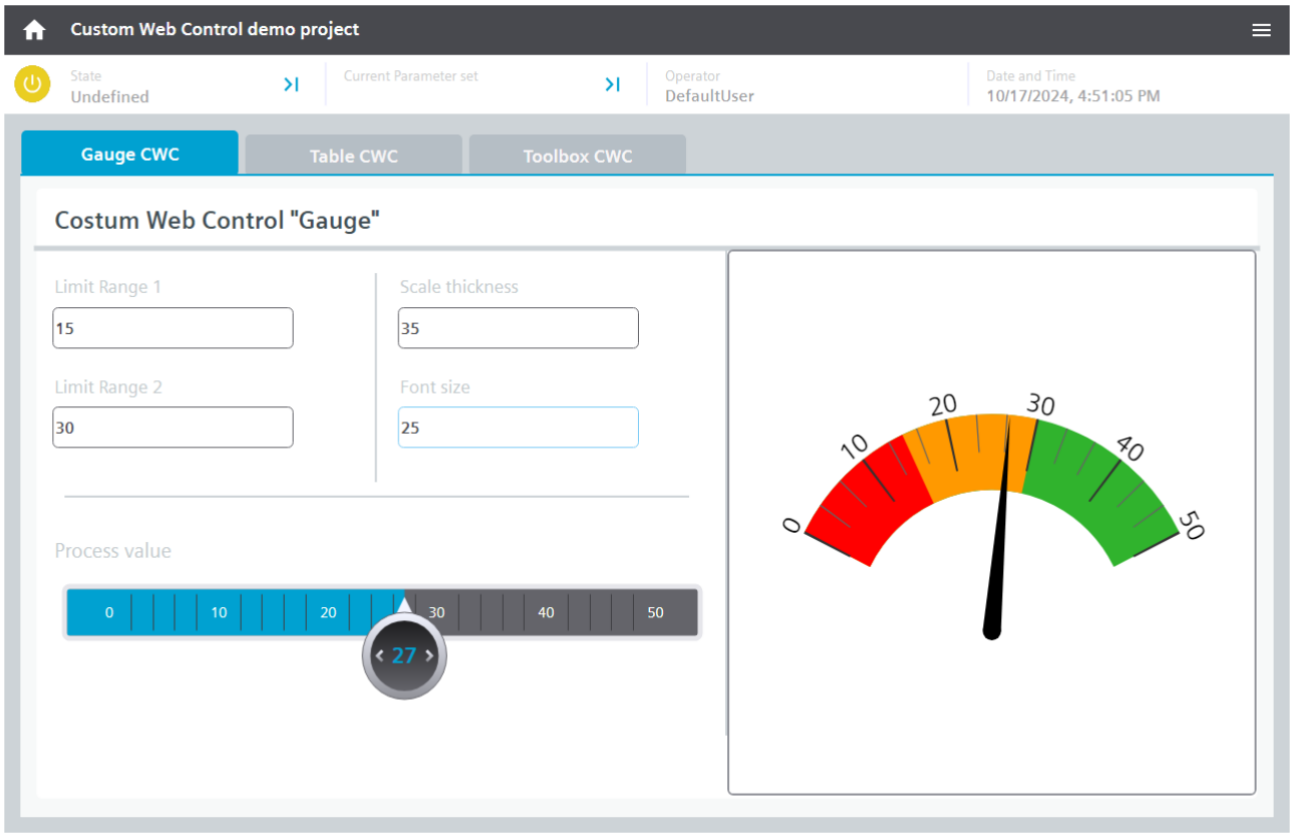
The following figure shows the properties in the TIA Portal, which can be set statically or dynamized.



5. Sample Project Operation

5.1. "Gauge" Custom Web Control

The sample project consists of three tabs. In the first tab, you will find the CWC "Gauge", in which four properties (via I/O fields) as well as the process value (via sliders) can be dynamized as an example.



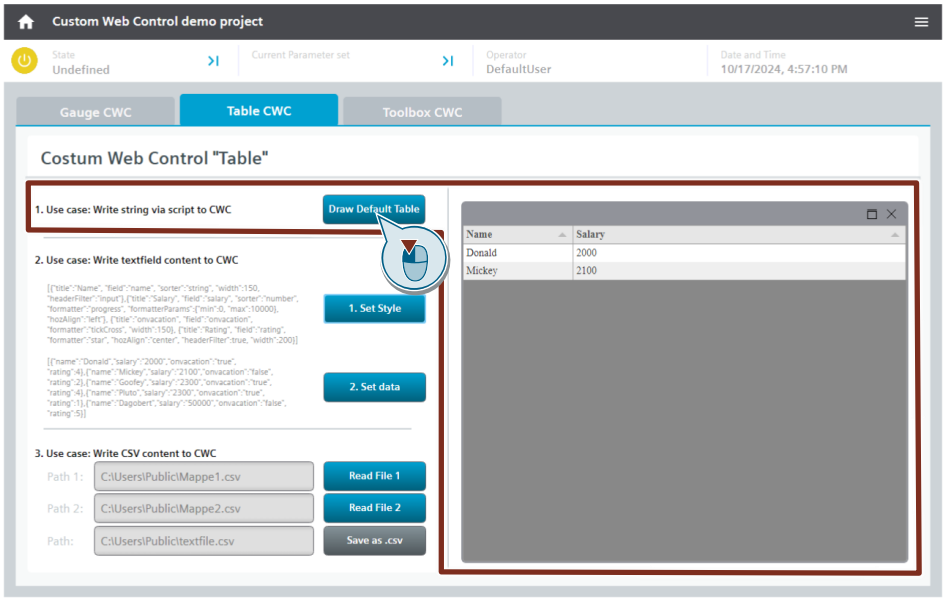
5.2. "Table" Custom Web Control

In the second tab, you will find the "Table" custom web control. You will be presented with three ways in which you can supply the custom web control with the required data (see Section 3.2.4 Necessary Data).

- Script: Table contents are stored in a script
- Screen object: Table contents are stored in a screen object (e.g., text field).
- CSV file: Table contents are included in a CSV file or need to be exported.

5.2.1. Submission of Table Contents via a Script

In the first example, the values are simply displayed in two columns. The two strings ("ColumnTableString" and "TableDataString") are passed to the CWC via the "Draw default table" button.



ColumnStyleString

For this example, the "ColumnStyleString" is composed as follows:

```
'[{"title":"Name", "field":"name", "sorter":"string", "width":150},
{"title":"Salary", "field":"salary", "sorter":"number","hozAlign":"left"}]'
```

TableDataString

For this example, the TableStyleString is composed as follows:

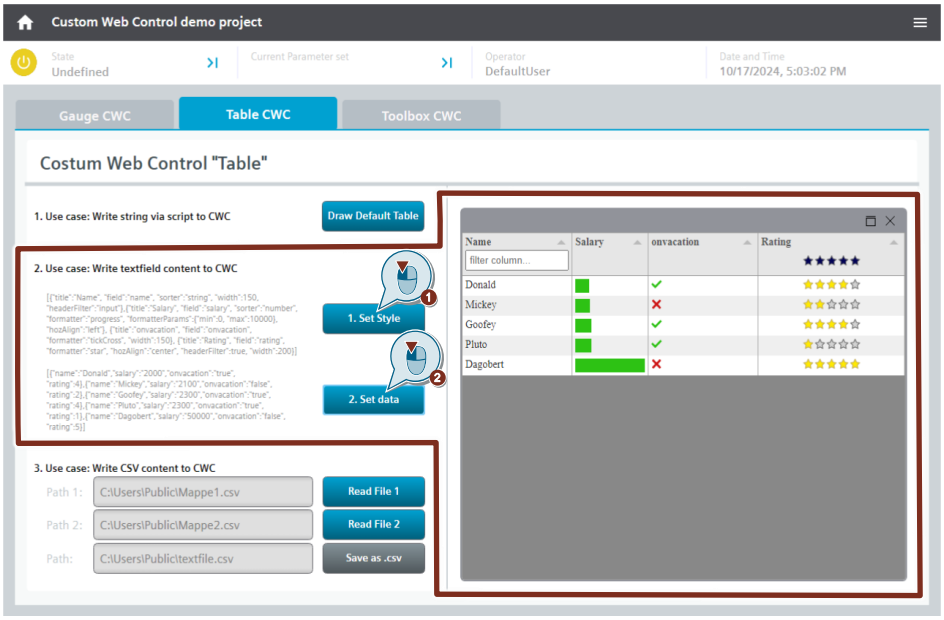
```
'[{"name":"Donald","salary":"2000"},
{"name":"Mickey","salary":"2100"}]'
```

NOTE Line breaks have been added to the two strings in the documentation to improve readability.

5.2.2. Transmission of Table Contents via a Screen Object

In the second example, the table contents are displayed graphically formatted. In addition, a filter has been added to the "Name" and "Rating" columns in order to filter the entries. The two strings ("ColumnTableString" and "TableDataString") were also stored in a screen object (in this example as a text field).

The format ("ColumnStyleString") is first transferred to the CWC via the "1 Set Style" button. The table content ("TableDataString") is then transferred using the "2. Set data" button.



ColumnStyleString

For this example, the "ColumnStyleString" is composed as follows:

```
[{"title":"Name", "field":"name", "sorter":"string", "width":150, "headerFilter":"input"},
{"title":"Salary", "field":"salary", "sorter":"number", "formatter":"progress",
"formatterParams":{"min":0, "max":10000}, "hozAlign":"left"},
{"title":"onvacation", "field":"onvacation", "formatter":"tickCross", "width":150},
{"title":"Rating", "field":"rating", "formatter":"star", "hozAlign":"center",
"headerFilter":true, "width":200}]
```

TableDataString

For this example, the TableStyleString is composed as follows:

```
[{"name":"Donald","salary":"2000","onvacation":"true", "rating":4},
{"name":"Mickey","salary":"2100","onvacation":"false", "rating":2},
{"name":"Goofey","salary":"2300","onvacation":"true", "rating":4},
{"name":"Pluto","salary":"2300","onvacation":"true", "rating":1},
{"name":"Dagobert","salary":"50000","onvacation":"false", "rating":5}]
```

NOTE Line breaks have been added to the two strings in the documentation to improve readability.

5.2.3. Transfer of Table Contents via a CSV File

In the third use case, the values are displayed both raw ("Read File 1") and formatted ("Read file 2").

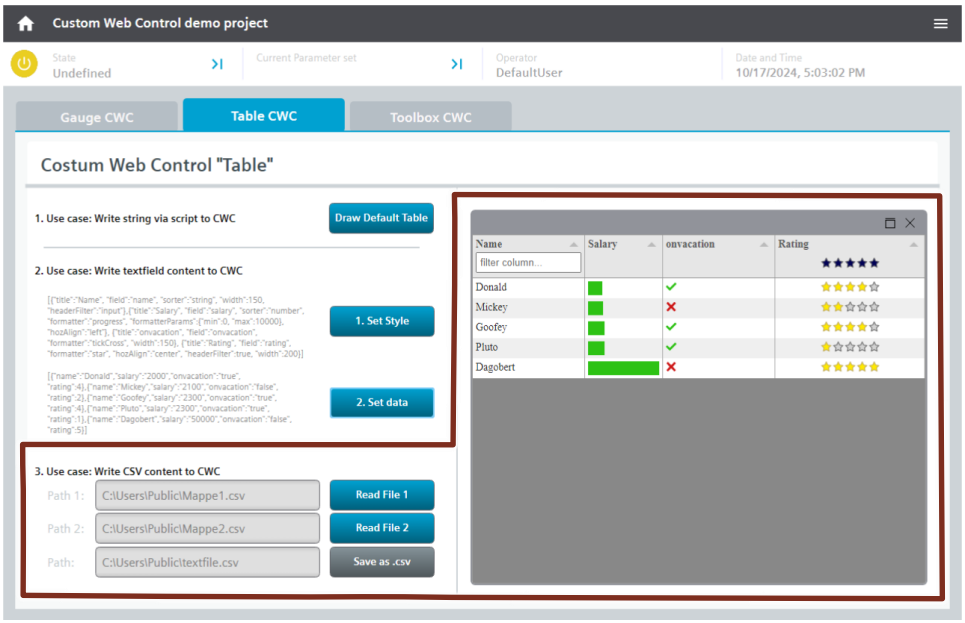
NOTE

Note : In the project directory of the sample project ("109779176_CustomWebControl_V30_PROJ\UserFiles") there are already two preconfigured CSV files. If you want to use them in the sample project, you must place both files (Folder 1 and Folder 2) in the following location on the Operator Panel you are using.

- Unified PC Runtime: "C:\Users\Public"
- Unified Comfort Panel: "\home\industrial\"

The string "ColumnTableString" is stored in the script of the respective button. It defines the raw or formatted display form.

The TableDataString table data is read from a CSV file and converted to JSON format by a script before being passed to the CWC.



ColumnStyleString

The "ColumnStyleString" is composed as follows for the unformatted example:

```
'[{"title":"Name", "field":"name", "sorter":"string", "width":150},
{"title":"Salary", "field":"salary", "sorter":"number", "hozAlign":"left"},

{"title":"Intern", "field":"intern", "sorter":"boolean", "hozAlign":"left"}]'
```

The "ColumnStyleString" is composed as follows for the formatted example:

```
'[{"title":"Name", "field":"name", "sorter":"string", "width":150, "headerFilter":"input"},
{"title":"Salary", "field":"salary", "sorter":"number", "formatter":"progress",
"formatterParams":{"min":0, "max":10000}, "hozAlign":"left"},
{"title":"OnVaccation", "field":"onvaccation", "formatter":"tickCross", "width":150},
{"title":"Rating", "field":"rating", "formatter":"star", "hozAlign":"center",
"headerFilter":true, "width":200}]'
```


TableDataString

For this example, the "TableStyleString" is as follows:

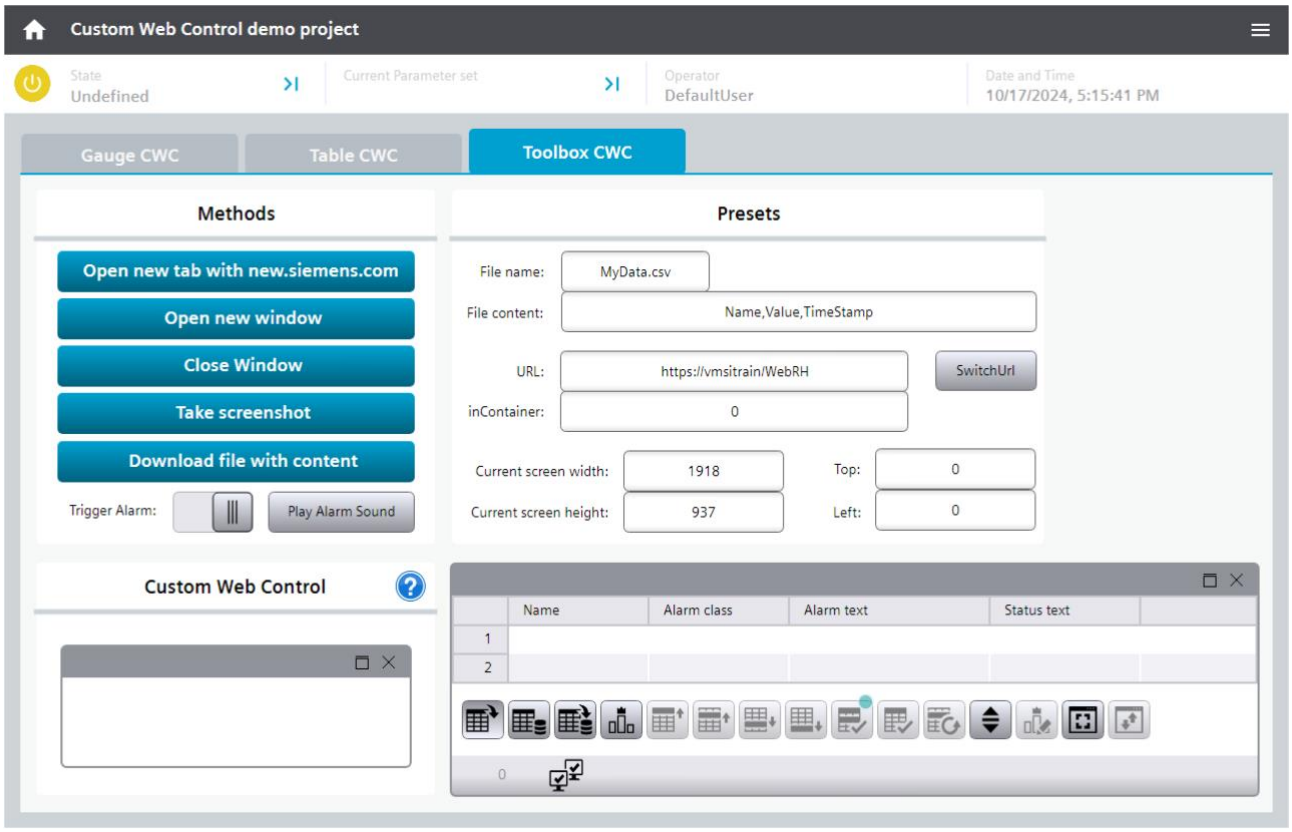
```
HMIRuntime.FileSystem.ReadFile("C:\\Users\\Public\\Mappe1.csv", "utf8").then(
  function(text) {
//read file and convert to a string in JSON format
  var lines = text.split("\n");
    var result = [];
    var headers;
    headers = lines[0].split(";");

    for (var i = 1; i < lines.length; i++) {
        var obj = {};
        if(lines[i] == undefined || lines[i].trim() == "") {
            continue;
        }
        var words = lines[i].split(";");
        for(var j = 0; j < words.length; j++) {
            obj[headers[j].trim()] = words[j];
        }
        result.push(obj);
    }

//write to Interface
Screen.Items('MyCWC').Properties.TableDataString = JSON.stringify(result);
```


5.3. "Toolbox" Custom Web Control

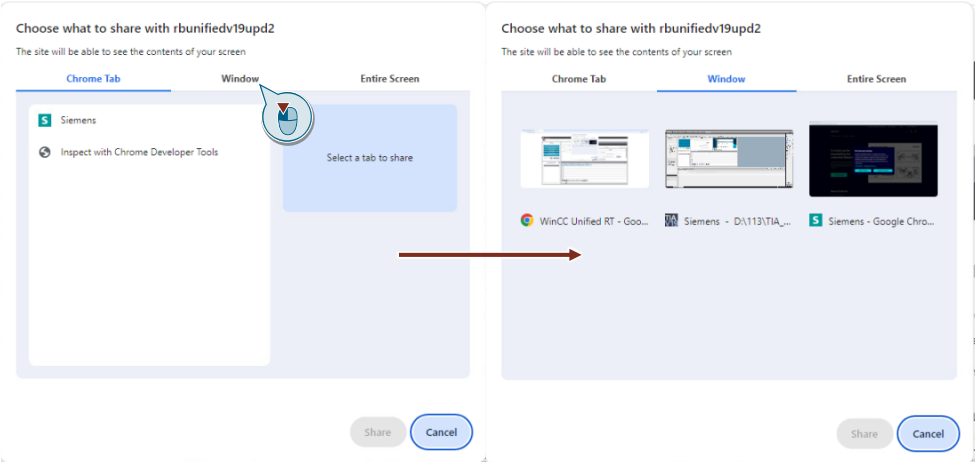
In the third tab, you will find the "Toolbox" custom web control. In the "Methods" area, the various methods of the control can be triggered. To the right of it are IO fields, which interact with the CWC in a variety of ways. The web control can be used to illustrate the "UnifiedInContainer" event. Below that is the CWC and an alarm control.



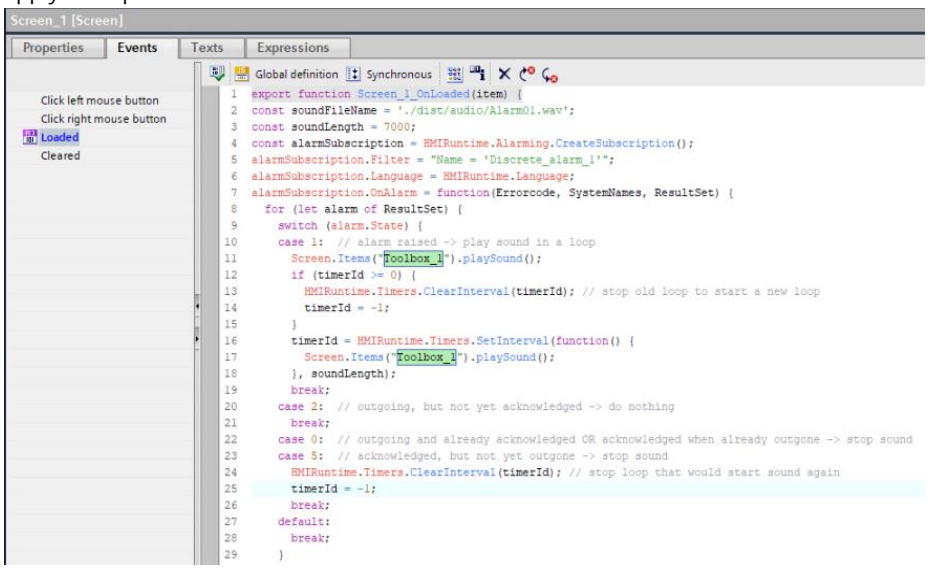
5.3.1. Methods

The following methods can be triggered via the buttons in the top left.

- The "Open new tab" button opens a new browser tab. In our sample project, the web page "http://new.siemens.com" is linked in the code of the button. The link can be adapted in Engineering.
- "Open new Window" also opens a new browser tab, but in this case as a new window. Again, the web page "http://new.siemens.com" is linked in the code of the button.
- "Close Window" closes the new browser window that has just been opened.
- The "Take Screenshot" button opens a browser dialog. In this view, you can choose from which view a screenshot should be taken. This feature is browser-dependent and may vary.



- Download file with content creates a file with content from Runtime. The file name and content can be dynamically adjusted further to the right in the "Presets" area.
- "Trigger Alarm" and "Play Alarm Sound" are available for the demonstration of the alarm horn feature. When the alarm is triggered, a sound will be played along with it. The code for linking the sound file to the alarm can be found in the Loaded event of the screen. An alarm subscription is required for this. The type of alarm and the respective sound can be set individually. For example, ".Filter" can be used to define the alarm class that should apply to a specific sound file.



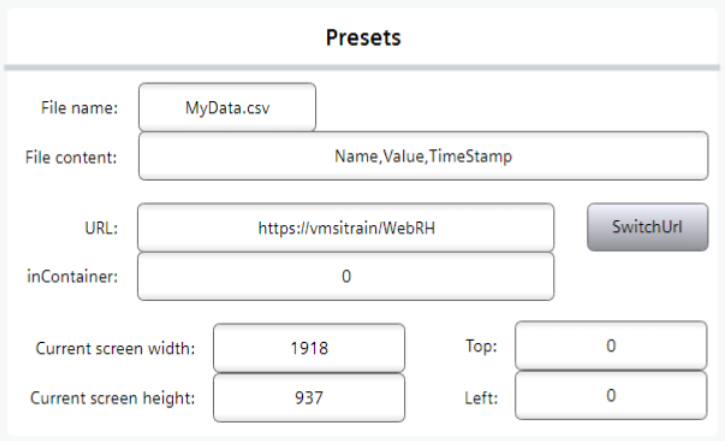
In this example, the filter in line 5 for the "Name" attribute was set to "Discrete_alarm_1". This means that the sound file is only played for this alarm.

NOTE New sounds can be stored in the folder of the custom web control under "./dist/audio".

- The OpenScreenOnMonitorX method was configured in the url event of the CWC. The event is triggered when Runtime is started. Together with the CheckFullscreen method and the default settings from the interface of the CWC, you can specify here on which screen of the device certain windows are to be opened.

5.3.2. Default Settings

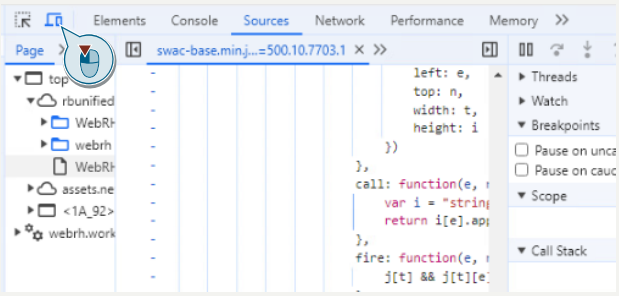
The IO fields in the "Presets" area interact with the CWC in various ways. File-name and File-content are required for the "Download file with content" method. Below this, the current URL of the browser window can be read out via the "url" event. The "inContainer" field uses a transfer parameter of the CWC to indicate whether the control is located in a container such as the web control. At the bottom are the displays for the current screen size and the position of a touch on Runtime, via Touchmove.



The "SwitchUrl" button can be used to overwrite the current url. In the example, "Siemens.com" is stored in the button.

NOTE

For the touch event, the browser on the PC must be switched to a mobile device. To do this, press the F12 button once and then press the button on the permission side to switch devices.



The two lower IO fields then record the position of the clicks in Runtime.

The IO field "inContainer" displays a 1 as soon as the CWC is in a container. This can be checked using a web control. To do this, simply insert the Runtime URL into the Web Control window. Runtime should then be displayed nested in the Control.

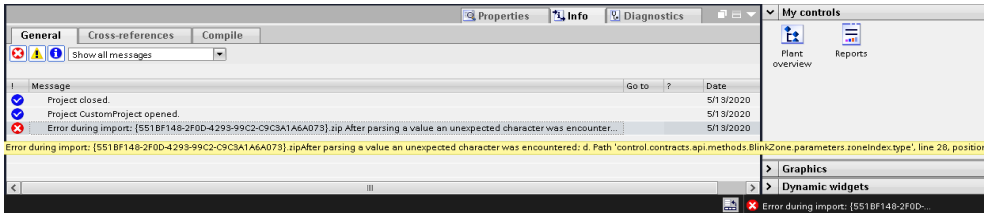
6. Common Error Patterns

This subsection describes common error patterns that can occur when creating and integrating custom web controls, as well as their solutions.

6.1. Custom Web Control Does not Appear in the TIA Portal Toolbox

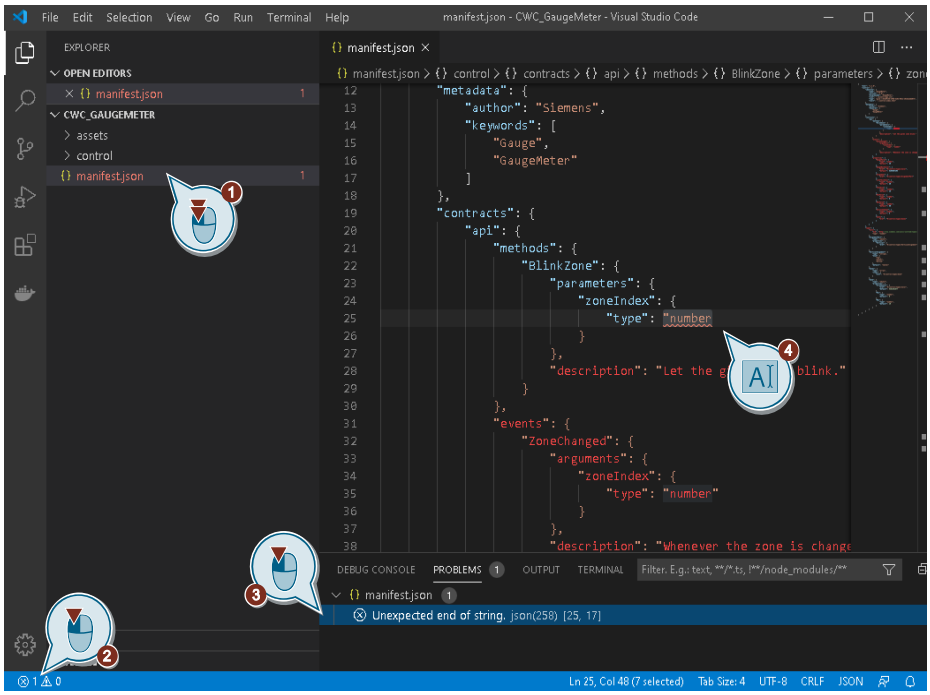
Error description

If you open a screen in the TIA Portal and your custom web control does not appear in the toolbox on the right, then your Manifest.json file contains an error. In addition, an error message appears in the info box, as shown in this figure:



Solution

To solve the problem, open your Manifest.json file with Visual Studio Code and fix the errors as shown in the figure below:

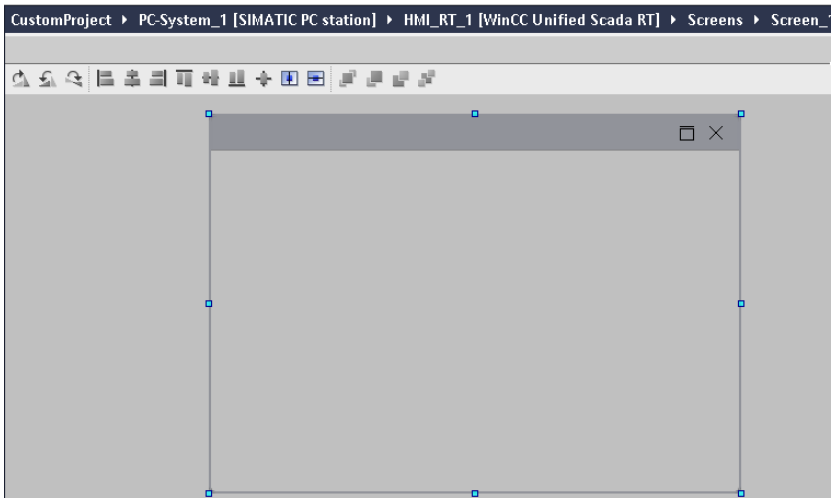


1. Double-click to open the Manifest.json file. Visual Studio Code will then automatically check them and display the file name in red and the number of errors on the permission page.
2. You will then see the number of errors at the bottom left. Click on it to open a window on the permission page with a detailed description of all errors.
3. Click on an error. Visual Studio Code jumps directly to the position in the document where the error is located.
4. Correct the error with the help of the error description.

6.2. TIA Portal Does not Display the Custom Web Control Correctly in the Screen

Error description

You can successfully insert your custom web control into the screen and link it to the properties. This also works without any problems in Runtime, but the screen editor in the TIA Portal displays an empty or faulty control, such as in this figure:



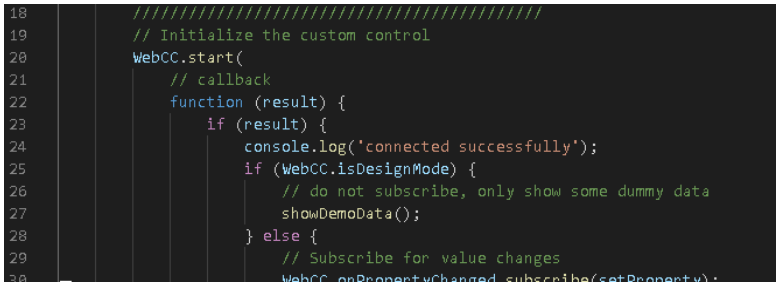
Explanation: TIA Portal tries to display the web content, but for security reasons it disables certain functions, with the result that your custom web control is displayed incorrectly or not at all.

Solution

As a solution, you can program a preview in your custom web control that only the TIA Portal screen editor displays. In Runtime, however, it will work as usual.

To do this, after a successful connection, you need to define in JavaScript how you want the code to proceed. With an IF query of the "isDesignMode" property on the "WebCC" API object, you can find out whether your custom web control is currently in the TIA Portal or in Runtime.

Your code should be similar to the following figure, with a new function "showDemoData()" defined here, in which you program your code to display a preview for the TIA Portal:



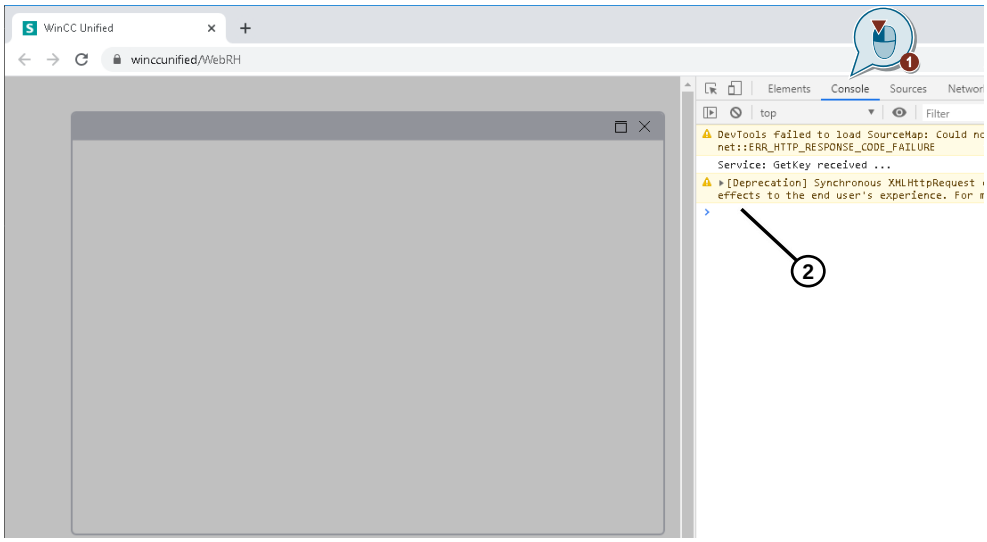
```
18        ////////////////////////////////////////////
19        // Initialize the custom control
20        WebCC.start(
21            // callback
22            function (result) {
23                if (result) {
24                    console.log('connected successfully');
25                    if (WebCC.isDesignMode) {
26                        // do not subscribe, only show some dummy data
27                        showDemoData();
28                    } else {
29                        // Subscribe for value changes
30                        WebCC.onPropertyChanged.subscribe(setProperty);
```


6.3. Custom Web Control Remains Empty in Runtime

NOTE The error description explained in this Section does not apply to the "Toolbox" CWC. With the CWC "Toolbox", it is normal that nothing is displayed, see Section 4.3.1.

Error description

You have successfully loaded your custom web control from the TIA Portal in Runtime. When you open the screen, you will see its edges but no content.



Solution

First check whether a connection has been established between your custom web control and WinCC Unified. To do this, open the developer console with F12 and then click "Console" as shown in Figure 5.5. If you do not see the success message "successfully connected" as shown in the figure, no connection has been established.

In this case, check that the "WebCC.Start()" function is called directly when the custom web control is started.

If a connection has been established but nothing still appears on the screen, then there is an error in the code you have written. Debug your code and fix the error.

6.4. Custom Web Control does not Include WinCC Data

Error description

The browser displays your custom web control correctly. However, if you change the linked tags, you will not see any change in value in the custom web control.

Solution

First, check that the tags are written correctly throughout your code and manifest.json file. Pay particular attention to matching upper- and lower-case letters.

Next, check that you have linked the correct tag of the value you want to adjust. If required, output the tag next to the custom web control in an I/O field.

If the I/O field is updated and all tags in the TIA Portal are connected correctly, but the value is still not displayed in the custom web control, then the error is in the code.

Check whether you have called the "`WebCC.onPropertyChanged.subscribe()`" function after a successful connection and passed a callback function (see "Configuration and Implementation of the Custom Web Control" section, step 5).

Use the debugging function to set a breakpoint at the start of the callback function.

Now change the linked value and observe whether the breakpoint is reached.

If the browser does not stop at the breakpoint, check whether you have passed the property with the correct name when calling "`WebCC.Start()`".

Here you can see another valid example of a correct call of the function:

```
WebCC.Start(function(result) {
   if (result) {
      console.log('connected successfully');
      WebCC.onPropertyChanged.subscribe((data) => {
        console.log(data);
      });
    }
  }, {
    properties: {MyIntProperty: 0, MyStringProperty: 'test'}
  }, [], 10000 // timeout
);
```

The Custom Web Control has requested the properties "MyIntProperty" and "MyStringProperty" from WinCC Unified. They must also be present in the Manifest.json file. Pay attention to upper and lower case letters and correct spelling.

6.5. Custom Web Control Cannot Write WinCC Data

Error description

The browser is displaying your custom web control correctly.

Your custom web control is programmed so that it can write properties. The properties are linked to WinCC tags in the TIA Portal. The custom web control should therefore change the WinCC tags directly.

If your custom web control writes the property, you will not see any change to the WinCC tags.

Solution

First, check that the tags are written correctly throughout your code and manifest.json file. Pay particular attention to matching upper- and lower-case letters.

Next, check that you have linked the correct tag of the value you want to adjust. If required, output the tag next to the custom web control in an I/O field.

If the I/O field is not updated, the error is in the code. Check whether you reach the line of code in which you are writing the value.

Use the debugging function to set a breakpoint in the line in which you write the property. Observe whether the stopping point is reached.

If the browser stops at the breakpoint and the tag value in WinCC does not change, check again whether you have used the correct "Write" property.

Another example of correct spelling:

```
WebCC.Properties.MyIntProperty = 5;
```

The custom web control writes the property "MyIntProperty". This must also be present in the Manifest.json file. Pay attention to upper- and lower-case letters and correct spelling.

7. Helpful Information

7.1. Tips & Tricks

Version management

Version management offers several benefits, for example that your code is saved again. Another advantage is the ability to track changes in case you can't trace bugs and want to go back to a stable version.

Version management platforms include http://github.com/.

Update of the custom web control when it is created

The "Installation and Integration into the User's Project" section explains how to load the custom web control into Runtime. If you need to experiment a lot when creating it, you'll need to go through the steps more often. The following procedures can save you time.

There is another method for programming custom web controls:

1. Create the Manifest.json file as completely as possible.
2. Create an empty "index.html" and "code.js" file.
3. Create a custom web control from these three files and integrate it (see "Installation and integration in the User Project" section).
4. Launch your browser and view your blank custom web control.
5. Go to the code of the online instance of the custom web control. In Windows, it is stored here: "C:\Users\Public\Documents\Siemens\WebUX_ResourceCache_CustomWebControls".
6. Modify the code.
7. Press F5 to refresh the Unified Runtime web page in the browser.
8. The custom web control has now been updated without having to download it again.
9. Repeat steps 6 and 7, until you are finished programming.

ATTENTION **Your code may be lost in this process!**

When you have finished programming, save the code of the online instance of the custom web control separately.

When you restart Runtime, WinCC Unified overwrites your code with the original code.

If you need to change the Manifest.json file at a later time, save your code separately as you would have to start again from step 3. Re-downloading the custom web control will also result in an override of your online code.

7.2. Alternatives

Runtime ODK and OpenPipe

Custom web controls are used to exchange data between the operator and the data in WinCC Unified (such as PLC tags).

If you're using custom web control to consume data from other web services, you should ask yourself if you only need the data for a specific operator, or if the data is relevant for all operators.

If the data is intended for a specific period of time and WinCC Unified needs to show the data directly to this operator using a custom web control, the custom web control is the right choice.

If, on the other hand, you query external data with the custom web control and write tags back to WinCC with properties in order to make the data available to several or all users, you may have the problem that several users query data and write tags to WinCC. This is an unnecessary additional load for your other web service and also for the WinCC Unified Runtime Server.

However, there is a faster and better solution: Write a program that runs on the WinCC Unified Runtime Server and accesses the data. This program can then write the data to WinCC via the OpenPipe or Runtime ODK interface and tags.

If you still need a specific display format that WinCC Unified does not currently offer, create a custom web control that only uses these WinCC tags.

In this way, you leave the topics of authentication, authorization and future redundancies to the WinCC Unified Server and at the same time have less development effort in Custom Web Control.

SIMATIC WinCC Unified – Toolbox

If you are not familiar with the programming of custom web controls or want to keep the time and effort low, there are also conventional alternatives in the form of the "SIMATIC WinCC Unified – Toolbox".

You can find the corresponding post in SIOS under the article ID: 109770480 \7\

This article contains some useful tools that you can easily integrate into your HMI configuration via libraries, e.g., a stopwatch or a calculator. The toolbox for SIMATIC WinCC Unified offers several tools and is constantly being expanded.

8. Appendix

8.1. Service and support

SiePortal

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

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

You can access SiePortal via this address: sieportal.siemens.com

Technical Support

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

SITRAIN – Digital Industry Academy

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

Industry Online Support app

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

8.2. Links and Literature

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the article page of the application example https://support.industry.siemens.com/cs/ww/en/view/109779176
\3\	Microsoft Visual Studio Code 1.44 https://code.visualstudio.com/
\4\	Gauge.js 1.3.8 https://bernii.github.io/gauge.js/
\5\	Tab v6.2 http://tabulator.info/
\6\	CWC Toolbox https://code.siemens.com/simatic-systems-support/visualization/customwebcontrols-unified/cwc_toolbox
\7\	Application example "SIMATIC WinCC Unified – Toolbox" https://support.industry.siemens.com/cs/ww/en/view/109770480
\8\	SITRAIN System Course "WinCC Unified & Unified Comfort Panels" https://support.industry.siemens.com/cs/ww/en/view/109773211
\9\	SITRAIN advanced course: SIMATIC WinCC Unified for PC systems https://support.industry.siemens.com/cs/ww/en/view/109781323
\10\	SITRAIN: You can find out more about the training and courses as well as their locations and dates at: https://www.siemens.com/sitrain

8.3. Change Documentation

Version	Date	Change
V1.0	06/2020	First edition
V2.0	04/2021	"Table" custom web control added, and document restructured
V3.0	06/2023	Section 3.2.4 "Autocompletion" added. Updated to TIA V18 / WinCC Unified V18
V4.0	09/2024	Added "Toolbox" section. Updated to TIA V19 / WinCC Unified V19

