Integrating user-defined controls into WinCC Unified (Custom Web Controls)

Custom Web Controls for WinCC Unified Systems

https://support.industry.siemens.com/cs/ww/en/view/109779176

Siemens Industry Online Support

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 **Fehler! Linkreferenz ungültig.**.
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 at: **Fehler! Linkreferenz ungültig.**.

Table of contents

1 Introduction

1.1 Overview

WinCC Unified offers the ability to integrate Custom Web Controls. Thus, your visualization is no longer limited to WinCC Unified's default controls.

You have the ability to create your customer-specific controls with web technology or to use existing controls and deploy them on other WinCC Unified runtime stations.

This purpose of this application example is to show you an example of how you can integrate a "Custom Web Control", created with Visual Studio Code, into WinCC Unified.

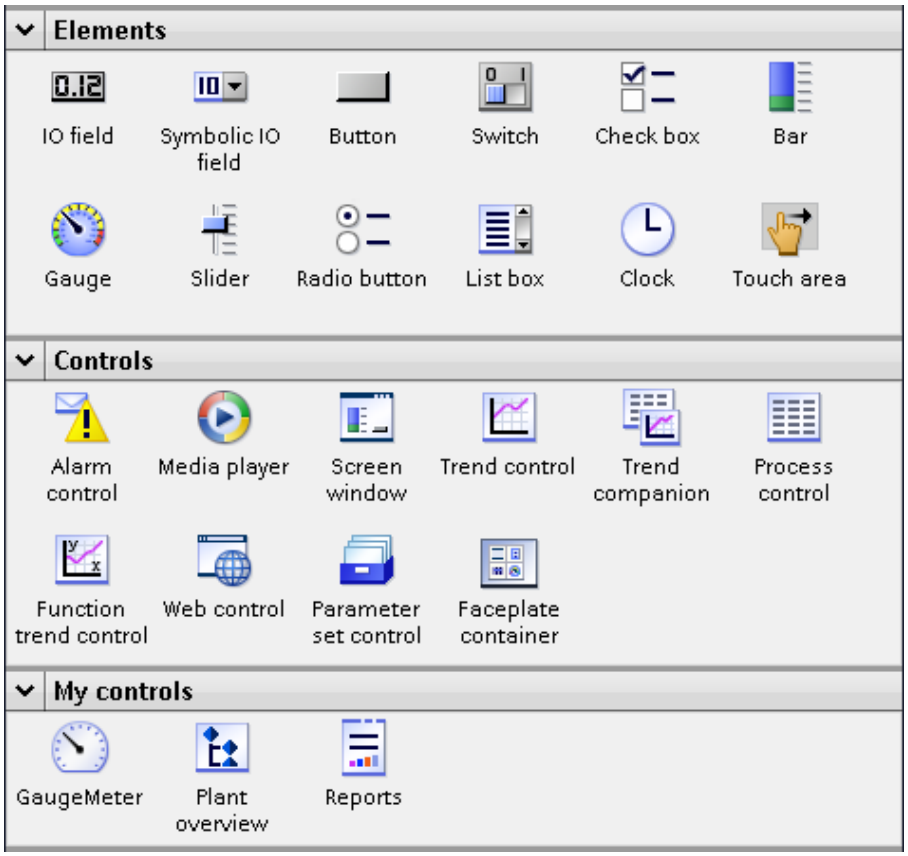
Note Custom Web Controls and the following configuration steps work with the WinCC Unified PC runtime and the Unified Comfort Panels.

Please observe the performance limits of the Panels. For example, Custom Web Controls with 3D graphics are not recommended for Unified Comfort Panels.

WinCC Unified controls and elements

Figure 1-1 shows the elements, default controls and custom controls of WinCC Unified. The section "My controls" contains the Custom Web Controls.

Figure 1-1

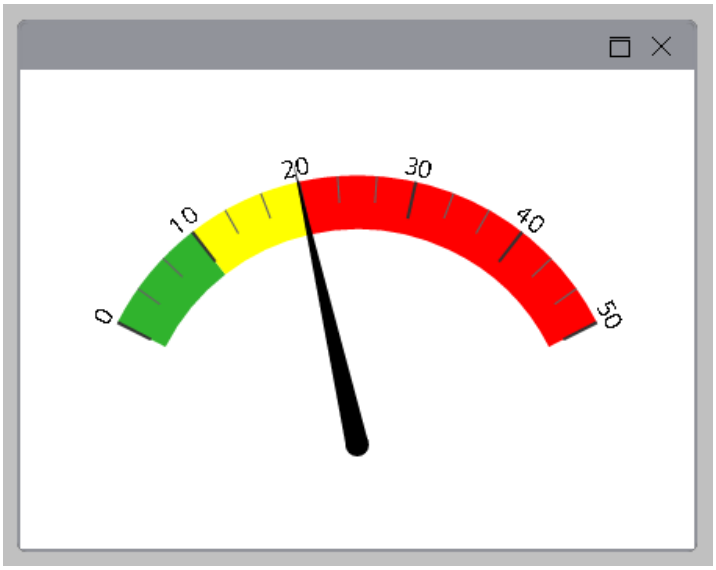


Custom Web Control in this example

Figure 1-2 shows the example Custom Web Control which will be used to demonstrate each of the configuration steps to integrate it into WinCC Unified.

The example control is from an open-source third-party library "Gauge.js" and will be expanded with code for its use in WinCC Unified.

Figure 1-2



1.2 Principle of operation

The complete Custom Web Control has the following functions:

- Display a WinCC Unified tag value on a gauge
- Define upper and lower limits
- Define various stylistic parameters for better appearance
- Define value zones with different colors
- Method for blinking the zone where the value currently is
- Event that occurs at the transition from one zone to another

Required knowledge

The application example requires the following basic knowledge:

- Configuring with WinCC Unified
  Basics are taught in the SITRAIN course "WinCC Unified & Unified Comfort Panels". See article ID 109773211.
- Microsoft Visual Studio Code
- Website programming with HTML5 and JavaScript

1.3 Components used

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

Table 1-1

Components	Quantity	Item number	Note
WinCC Unified V16	1	6AV2102-0AA06-0AA7	Or later version
Microsoft Visual Studio Code 1.44	1	Refer to Internet	Or later version
Gauge.js 1.3.7	1	See Internet \5\	Open source component - **not contained in the example code.** Download the file and copy it into the appropriate folder (see chapter 2, Engineering).

This application example consists of the following components:

Table 1-2

Components	File name	Note
Example project/files	109779176_PrepareCustomWebControls_CODE.zip	V 1.0
Documentation	109779176_PrepareCustomWebControls_DOC.docx	V 1.0

2 Engineering

2.1 Configuration and implementation of the Custom Web Control

Using an example, this chapter will show how a Custom Web Control is created with Visual Studio Code.

Note

You may use other text-based development environments.

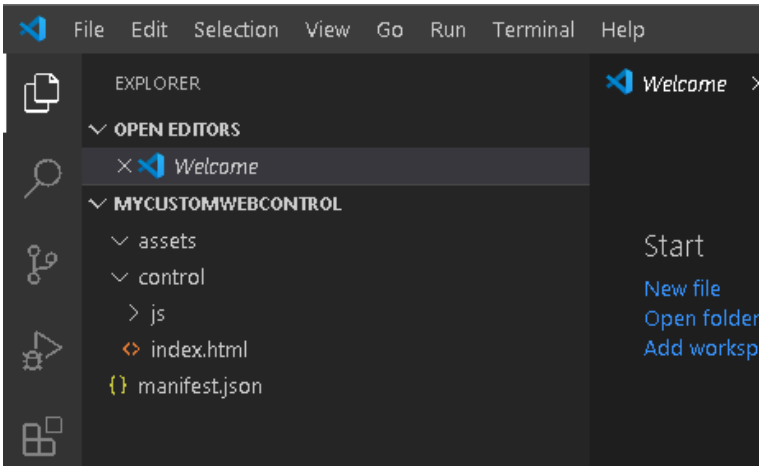
If you don't have much experience of the subject and are not familiar with the software tool "Visual Studio Code", chapter 3, Useful information will provide you with a brief introduction.

Creating a Custom Web Control

1. **Creating folder structure**
   (Also refer to the manual under: "General configuration and folder structure".)
   a. Open Windows Explorer and create the following elements in a working folder of your choosing:
   b. File named "manifest.json"
   c. Folder named "assets":
   Put an icon here in any image format. It will be shown in TIA Portal for this control.
   d. Folder named "control":
   Here, create a file, "index.html", "styles.css" and a folder "js" in which you place the file "webcc.min.js" from the attached files in this application example.
   Place the downloaded file "gauge.js" into the folder.
   (Source: https://bernii.github.io/gauge.js/ \5\)

2. **Open Visual Studio Code**
   a. Open Visual Studio Code (or a different editor).
   b. Use "File" → "Open Folder" to open the folder where the folder structure you just created is located.

Figure 2-1



3. Create the **Manifest.json** file
   (Also refer to the manual under: "Contract-based interaction and the Manifest file")
   a. Open the file Manifest.json in Visual Studio Code (or a different editor). This file necessitates a certain structure, which is explained in more detail in the manual. For this application example, it will be sufficient to copy an existing file and make a couple modifications.
   b. Copy the contents of the attached manifest.json file into your manifest.json.

Figure 2-2

   c. Give a custom name for your Custom Web Control under the "name" attribute in lines 5 and 7.
   d. Adjust the name of your logo in line 8 (all common image formats will work, such as JPG, PNG, BMP, etc.).
   e. In line 9, assign a new, custom GUID. You can create one with an online generator, such as: https://www.guidgenerator.com/.
   f. Where necessary, modify the metadata starting on line 12.
   g. Where necessary, change the interface of your Custom Web Control starting on line 19. Some possible data types include: "boolean", "number", "string", "array", or one of your own defined data types, starting on line 79.
   (For usage, see the included Manifest.json file.)
   h. Tip: In order to be sure that you created a valid JSON file, refer Visual Studio Code to it or copy the file's contents to an online validation tool (such as https://jsonlint.com: Insert content and click "Validate JSON" in the lower left.)

4. **Index.html file**
   (Also refer to the manual under "Interaction between control and container via the API".)
   a. Open the file Index.html in Visual Studio Code (or a different editor). This file is the portal to your website. Here you will also establish a connection to the WinCC Unified runtime server in order to exchange data.
   b. The connection data for WinCC Unified are in the attached file "webcc.min.js". Place it in the "js" folder and reference the file in index.html as follows:
   `<script src='./js/webcc.min.js'></script>`
   This reference is best made in your <Head> tag (also see line 11 of the attached example).
   c. The connection is established with the function `WebCC.start()` (in line 226). For this, it is important that the connection is established right when the site is visited. You can best achieve this if the function (like in the example) is directly in the <Script> tag and not in a more deeply nested function, which may only be retrieved at a later point in time.
5. **Data exchange between Custom Web Control and WinCC**
   (Also refer to the manual under "Using the control via WinCC")
   a. From anywhere in your application, you can now access the data that are defined in the Manifest.json file.
   b. Access is facilitates with the API object "`WebCC`". You already used it to establish the connection. You now have additional options.
   c. If you wish to read or write properties (in the Manifest.json file under "properties" starting on line 41), you can for example access the property "GaugeValue" with write access in the following manner:
   `WebCC.Properties.GaugeValue = 5` to set the value to 5. The value of the linked WinCC tag will also be set to 5.

d. If you are interested in changing the connected WinCC tags (possibly a PLC tag), use the function "`WebCC.onPropertyChanged.subscribe()`". You should call this function directly after the connection is successfully established in order to receive all changes from the beginning onward (see line 230 and 244). For handoff parameter, use a function that you defined (callback function). In the attached example, this is the function "setProperty" in line 143.
For any data change at all, the function "setProperty" is called and passes a "data" object which contains a "key" and a "value". The "key" is the name of the property which has changed, while "value" is the new value. Therefore, it is recommended to program switch-case branch point in order to be able to process the new value appropriately.

Figure 2-3

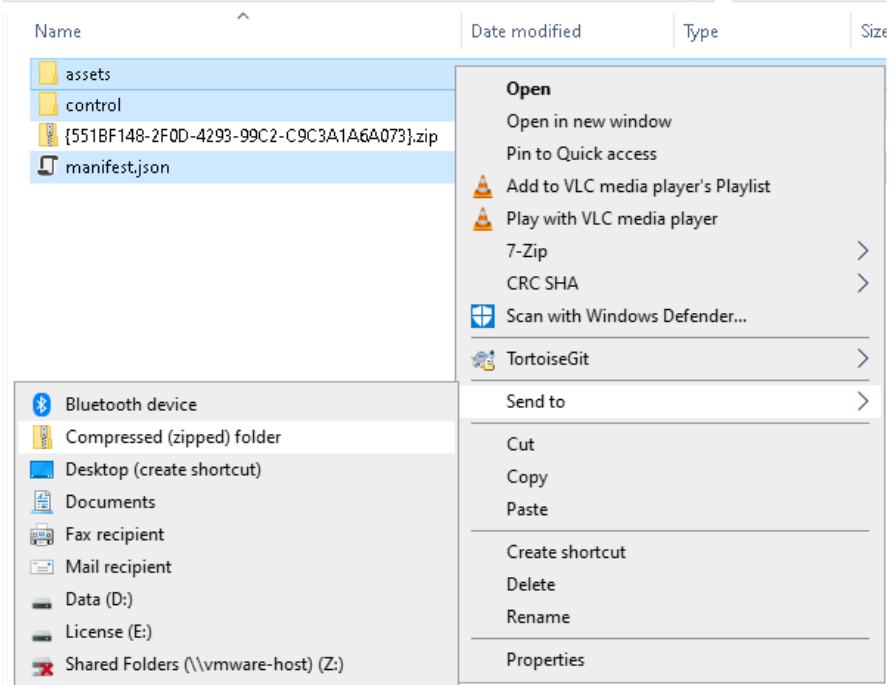
```
139         // This is a callback function that is called every ti
140         // other functions so you can see the new value in the
141         // - data: object containing a key and a value propert
142         //         the "value" contains the new value.
143         function setProperty(data) {
144             // console.log('onPropertyChanged ' + data.key);
145             switch (data.key) {
146                 case 'GaugeValue':
147                     updateValue(data.value);
148                     break;
149                 case 'GaugeBackColor':
150                     document.body.style.backgroundColor = toCo
```

e. If you declared methods in the Manifest.json file (see line 21), WinCC can call these methods and you can respond to them in the Custom Web Control. WinCC can call these methods at any time, meaning that you must define what should happen in each case before the connection is established. You will define a function to the exact name that you named in the Manifest.json file, and the same parameters. You can find an example in the attached index.html file starting on line 254.

f. If you have specified an event in the Manifest.json file (see line 31), you can trigger this event at any point in your code with "`WebCC.Events.fire()`" so that WinCC will be notified. The first handoff parameter in such case is always the name of the event that you wish to trigger, followed by all handoff parameters in the proper order as you specified in the Manifest.json file. You can find an example in the attached index.html file in line 87.

6. **Deploy process for using the Custom Web Control in TIA Portal**
   (Also refer to the manual under "Creating the ZIP file")
   a. When you are finished programming, you still have to package the code in such a way that TIA Portal correctly recognizes your Custom Web Control.
   b. To do this, open the Windows Explorer and go to your project folder. Select the folders that you created originally, "assets", "control" and the file "manifest.json" and archive them with right-click → "Send to" → "Compressed (zipped) folder".
   c. Use your GUID file from the Manifest.json file in order to modify the name of your generated .zip file as follows (the X's stand for your GUID):
   `{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip`
   Please note the curly braces before and after the GUID.

   Figure 2-4



   d. Your Custom Web Control is now finished and you can use it in TIA Portal.

2.2 Installation and integration into the user project

Note The project from the download in this article still does not contain the file "gauge.js".

Download the file from the address https://bernii.github.io/gauge.js \5\ and place the file in the proper folder. See chapter 2.1.

Integration into TIA Portal

Before you can use the Custom Web Control in TIA Portal, you have two ways to install it (also refer to the official manual under "Installing a Custom Web Control"):

1. Only make available for a specific project
   Place your Custom Web Control in your project folder:
   `…Project_1\UserFiles\CustomControls\{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip`

2. Make available for all projects
   Place your Custom Web Control in the installation path of TIA Portal:
   `C:\Program Files\Siemens\Automation\Portal Vxx\Data\Hmi\CustomControls\{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}.zip`

   Note Please note that if you copy the project to another PC, the Custom Web Control will then not be copied alongside it, and you will have to repeat this step.

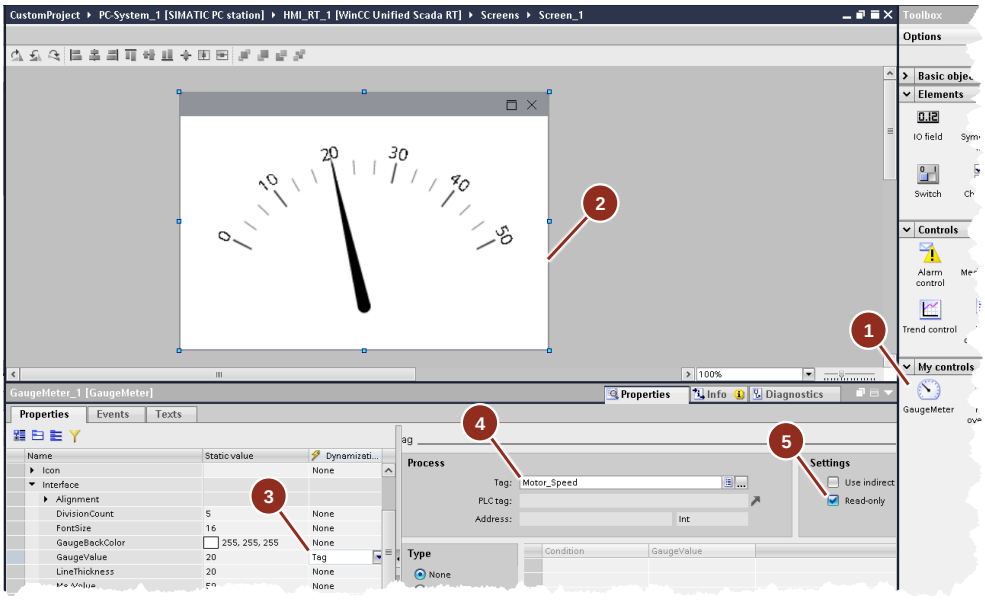
Note With the download to the runtime, TIA Portal will also transfer your Custom Web Control. There is no installation to the runtime server.

Integration into the user project

Now open a screen in your Unified Comfort Panel or PC station.

1. Click on your Custom Web Control and hold down the mouse button.
2. Keeping the mouse pressed, drag your control into the screen

Figure 2-5



In the Properties window in the bottom left under "Interface" you will find all properties that you defined in the Manifest.json file.

As with all other properties of screen objects, you can assign a static value or create a dynamic effect. See Figure 2-5.

3. Create a dynamization "Tag" for "GaugeValue".
4. Select a suitable PLC or HMI tag.
5. If your Custom Web Control has read-only access to the tag, then leave the box checked. However, if your implementation entails that the Custom Web Control will modify your tags, then uncheck the box. In this case, the "GaugeMeter" is merely displaying the tag.

2.3 Debugging

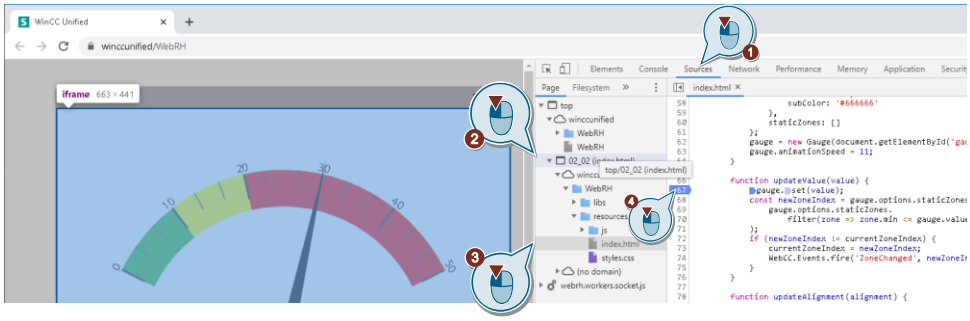
In this chapter, you will learn to debug Custom Web Controls in the runtime.

Debugging is not possible in TIA Portal. If you cannot transfer the Custom Web Control into the runtime because encountered problems earlier, look for a solution in the chapter **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**.

The Unified Runtime sends the Custom Web Control directly to the web client when used, so that you can also debug it in the web client with the browser's default developer console.

Open the Unified Runtime screen containing the Custom Web Control in Google Chrome and press F12 to open the developer console.

Figure 2-6



In Figure 2-6 you will see how debugging proceeds once the developer console has been opened with F12.

1. Select the "Sources" tab in the developer console.
2. Slowly move the mouse cursor to the left side of the developer console over the folders under the "top" node. Your Custom Web Control has no explicit name here, but the browser will highlight it in blue once you have selected it. (Here you will find a separate folder for each Custom Web Control instance.)
3. Open the respective folder and navigate to the file that you want to debug. (Here, all the code will be in the index.html file.)
4. In the right-hand window, scroll to the corresponding position and click the line number to insert a breakpoint. Once the code comes upon this position again, it will stop and you will receive detailed information about the program's run. In the image, there is a breakpoint at the beginning of the function "updateValue". The script will now always stop here if you update the linked value. Remove the breakpoint by clicking the line number again.

Note

For more information on working with the developer console and which information you can glean from it, consider familiarizing yourself with the official documentation from Google: https://developers.google.com/web/tools/chrome-devtools/javascript#sources-ui

You can also debug using any other browser. Use the documentation for the browser in question when doing so.

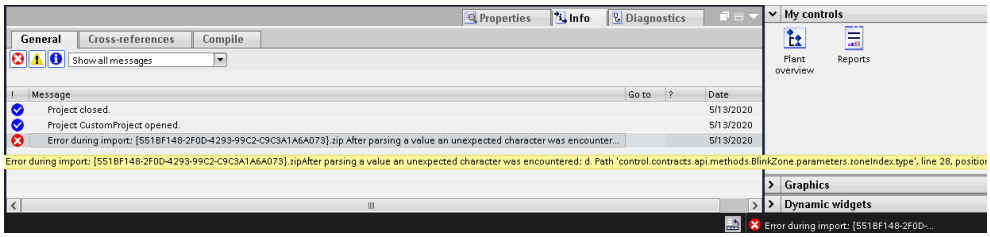
2.4 Frequent error constellations

This chapter describes common error constellation (and their solutions) which can occur while creating and integrating a Custom Web Control.

2.4.1 Custom Web Control does not appear in the TIA Portal Toolbox

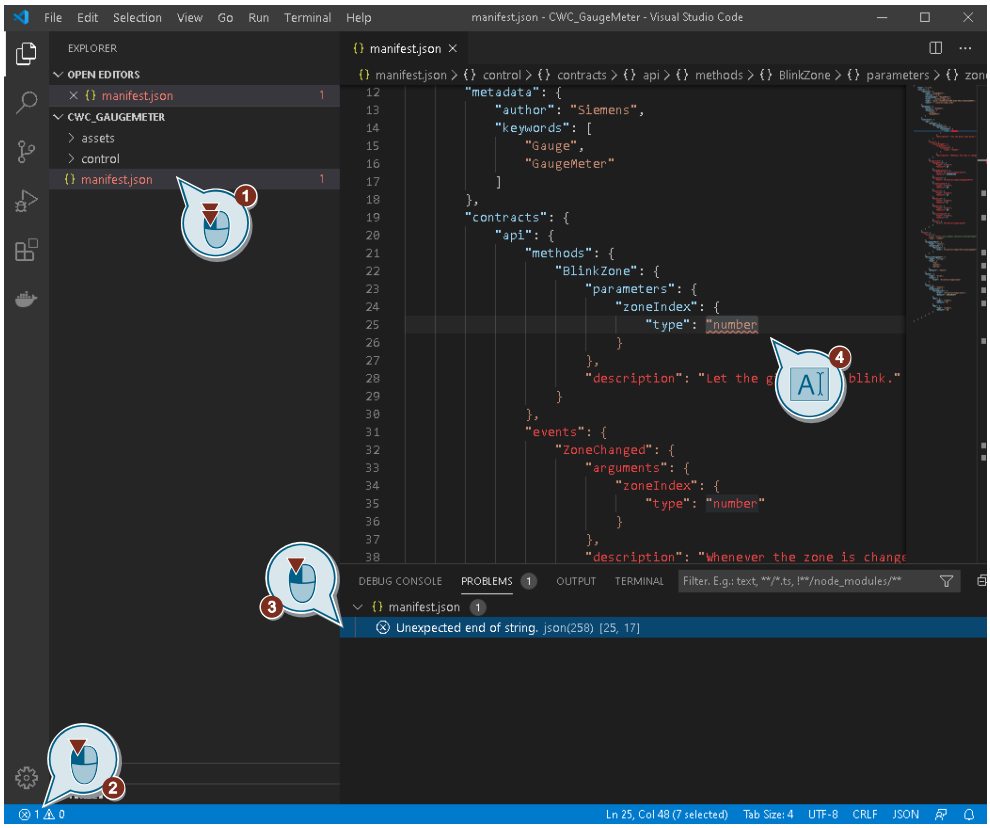
If you open a screen in TIA Portal and your Custom Web Control does not appear on the right in the Toolbox, then your Manifest.json file has an error. An error message will also appear in the info box, as can be seen in Figure 2-7.

Figure 2-7



To solve the problem, open your Manifest.json file with Visual Studio Code and fix the error as shown in Figure 2-8.

Figure 2-8



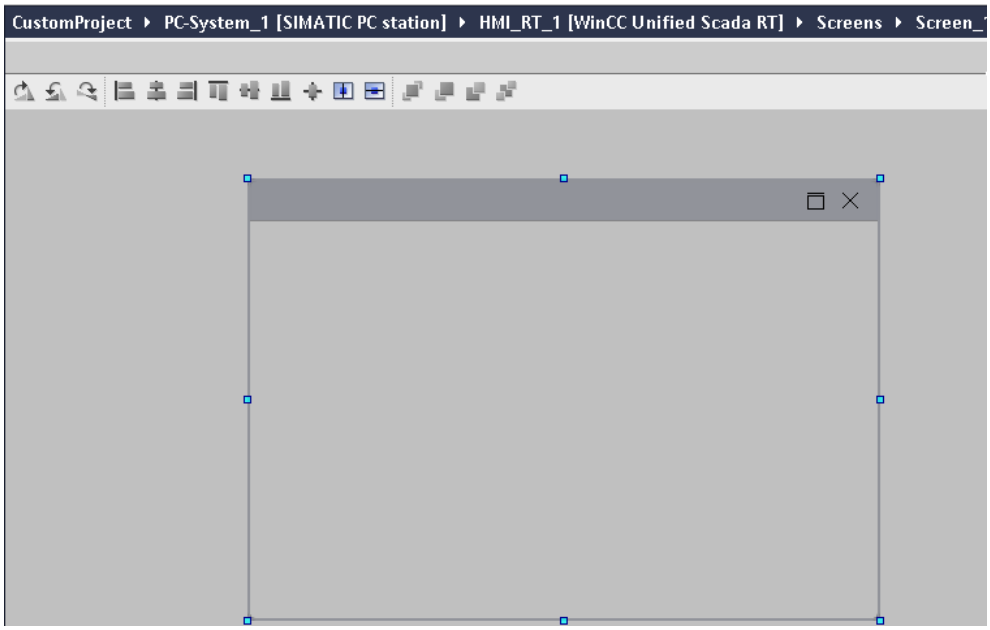
1. Double-click to open the Manifest.json file. Visual Studio Code will then automatically validate it and show the file name in red, as well as the number of errors to the right.
2. You will again see the number of errors to the bottom left. Click on it to open a window on the right with detailed descriptions of all errors.

3. Click on an error. Visual Studio Code will jump directly to the point in the document where it occurs.
4. Correct the error with the help of the error description.

2.4.2 TIA Portal does not display the Custom Web Control correctly in the screen

You can successfully insert your Custom Web Control into the screen and interconnect the properties. It also works fine in the runtime, but the TIA Portal screen editor shows an empty or erroneous control as seen for example in Figure 2-9.

Figure 2-9



Explanation: TIA Portal will try to try to display the web contents but for security reasons it will deactivate some functions, with the result that your Custom Web Control is displayed incorrectly or not at all.

As a solution, you can program a preview in your Custom Web Control that only displays the TIA Portal screen editor. In the runtime, however, it will function as expected.

In order to achieve this, you must define in JavaScript after a successful connection how the code is to proceed. Using an IF query of the property "isDesignMode" at the API object "WebCC", you can find out whether your Custom Web Control is currently in TIA Portal or in the runtime.
You code could look like the code in Figure 2-10; here you will have defined a new function "showDemoData()", in which you program your code to display a preview for TIA Portal.

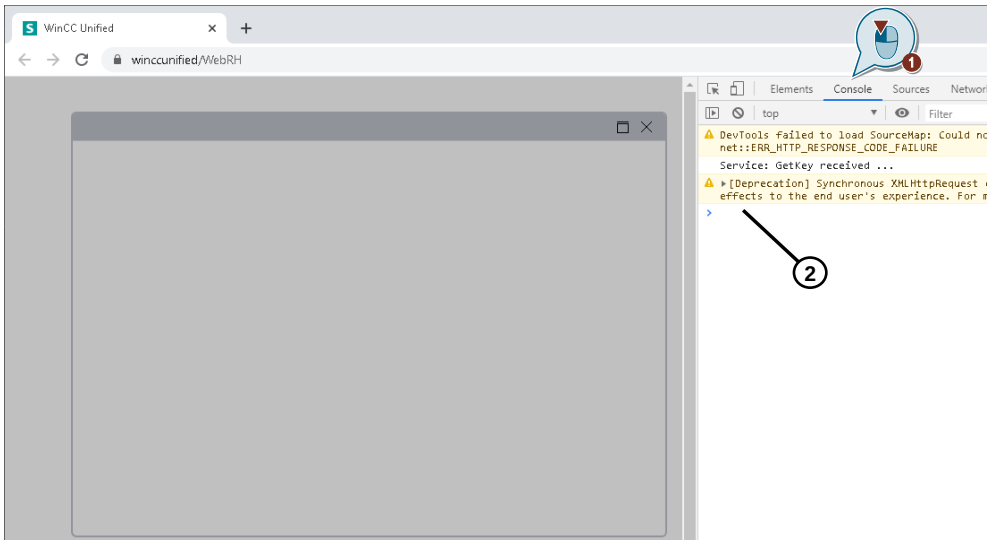
Figure 2-10

```
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);
```


2.4.3 Custom Web Control remains empty even in the runtime

You were able to successfully load your Custom Web Control into the runtime from TIA Portal. When you open the screen, you see the border but the contents are missing.

Figure 2-11



First check whether a connection was established between your Custom Web Control and WinCC Unified. To do this, open the developer console with F12 and then click on "Console" as seen in Figure 2-11. If you do not see a success message, "connected successfully", as in the Figure, then no connection was established.

In this case, please check whether you are calling the function "`WebCC.Start()`" right when the Custom Web Control starts.

If a connection is established but still no screen appears, the error is in the code that you wrote. Debug your code to remedy the error (see 2.3 Debugging).

2.4.4 Custom Web Control contains no WinCC data

The browser is displaying your Custom Web Control correctly. But when you modify the linked tags, you don't see the values change in the Custom Web Control.

First check that the tags are written correctly everywhere they are used in your code and the Manifest.json file. Pay particular attention to matching the capitalization.

Then check if you connected the right tag in TIA Portal. Where necessary, output it in an I/O field next to the Custom Web Control.

If the I/O field updates and all tags are correctly interconnected in TIA Portal, but the value still can't be seen in the Custom Web Control, the error is in the code.

Check whether you are calling the function "`WebCC.onPropertyChanged.subscribe()`" after successfully connecting and that you have passed a callback function (see 2.1 Configuration and implementation of the Custom Web Control, step 5).

With the help of debugging, (see 2.3 Debugging) set a breakpoint at the start of the callback function (in this example, in line 145).

Now change the linked value and pay attention to whether the breakpoint is reached.

If the browser does not stop at the breakpoint, check if you have correctly passed the property with the correct name when calling "`WebCC.Start()`" (in the example, see line 263).

Another valid example of a correct call of the function can be found here:

```
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 the spelling and capitalization.

2.4.5 Custom Web Control cannot write WinCC data

The browser is displaying your Custom Web Control correctly. You have also programmed your Custom Web Control in such a way that it writes to properties, which are linked with WinCC tags in TIA Portal, in order to modify WinCC tags directly from the Custom Web Control. Your Custom Web Control writes the property, but you don't see any change in the WinCC tags.

First check that the tags are written correctly everywhere they are used in your code and the Manifest.json file. Pay particular attention to matching the capitalization.

Then check if you connected the right tag in TIA Portal. Where necessary, output it in an I/O field next to the Custom Web Control.

If the I/O field does not update, the error is in the code.

Check whether you are reaching the code line where you write the value.

Use the debugging process (see 2.3 Debugging) to set a breakpoint at the line where you write the property.

Monitor whether the breakpoint is reached.

If the browser stops at the breakpoint but the tag value still does not change in WinCC, check one more time that the writing of a property is correctly used. Another valid example of correct writing:

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

The Custom Web Control writes the property "MyIntProperty". They must also be present in the Manifest.json file. Pay attention to the spelling and capitalization.

3 Useful information

3.1 Tips & tricks

Version management

Version management offers some advantages, one of which is that your code is saved again. Another is tracking of changes, in the event that errors can't be traced and you wish to go back to a stable version.

Platforms for version management include http://github.com/.

Updating the Custom Web Control while creating it

Chapter 2.2 explains how you can load the Custom Web Control into the runtime. While creating it, you will do a lot of testing. This means that you will go through these steps very frequently, which costs significant time.

There is an alternative method when programming a Custom Web Control:

1. Create the Manifest.json file as completely as you can.
2. Create an empty "index.html" file.
3. Create a Custom Web Control from just these two files and integrate it (see 2.2 Installation and integration into the user project).
4. Launch your browser and view your empty Custom Web Control.
5. Navigate to the code of the online instance of the Custom Web Control. In Windows, it is located here: „C:\Users\Public\Documents\Siemens\WebUX_ResourceCache_CustomWebControls“.
6. Modify the code.
7. Update the Unified Runtime website in the browser with F5.
8. The Custom Web Control has now been updated without needing to download it again.
9. Repeat steps 6 and 7 until you are finished programming.

CAUTION	**With this process, your code can be lost!**
	Once you are finished programming, save the code of the online instance of the Custom Web Control separately.
	When the runtime restarts, WinCC Unified will overwrite your code with the original code.
	If you have to modify the Manifest.json file after the fact, also save your code separately, as you would have to start over from step 3. Downloading the Custom Web Control again overwrites your online code in the same manner.

3.2 Alternative solutions

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 use the Custom Web Control to utilize data from other web services, ask yourself whether you only need the data in the display for a specific operator or if the data are relevant for all operators.

If the data are intended for a specific duration and WinCC Unified must show that operator the data with a Custom Web Control directly, then you made the right choice with the Custom Web Control.

On the other hand, if you query external data with the Custom Web Control and write back tags in WinCC with properties in order to provide the data to multiple users or all users, then you may have the problem of multiple users requesting data and writing tags in WinCC. This is an unnecessary load on your other web service as well as for the WinCC Unified runtime server.

There is a shorter way that yields higher performance: Write an application that runs on the WinCC Unified runtime server and fetches the data. This application can then write the data via the OpenPipe or the runtime ODK interface and write tags in WinCC.

If you still wish to have a specific display form that WinCC Unified does not currently offer, create a Custom Web Control that uses solely these WinCC tags.

In this way, you leave the issues of authentication, authorization and future redundancy to the WinCC Unified server and have less development work to do in the Custom Web Control.

4 Appendix

4.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

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

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

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks: **Fehler! Linkreferenz ungültig.**

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:
www.siemens.com/industry/supportrequest

SITRAIN – Training for Industry

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:
www.siemens.com/sitrain

Service offer

Our range of services includes the following:

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

You can find detailed information on our range of services in the service catalog web page:
Fehler! Linkreferenz ungültig.

Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for Apple iOS, Android and Windows Phone:
Fehler! Linkreferenz ungültig.

4.2 Links and literature

Table 4-1

No.	Subject
\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\	SITRAIN course "WinCC Unified & Unified Comfort Panels" https://support.industry.siemens.com/cs/ww/en/view/109773211
\4\	Microsoft Visual Studio Code 1.44 https://code.visualstudio.com/
\5\	Gauge.js 1.3.7 https://bernii.github.io/gauge.js/
\6\	SITRAIN: You can find out more about the training and courses as well as their locations and dates at: https://new.siemens.com/global/en/products/services/industry/sitrain.html

4.3 Change documentation

Table 4-2

Version	Date	Change
V1.0	06/2020	First edition