SIEMENS

SIMATIC HMI

WinCC Unified Programming Custom Web Controls

System Manual

Online documentation

05/2020

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

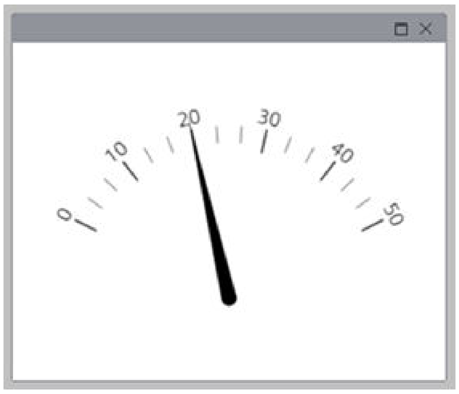
℗ 06/2020 Subject to change

Table of contents

Custom web controls 1

Custom web controls represent an independent web page with interface to Unified Runtime. Custom web controls offer you the option of adding your own elements to the visualization elements provided by WinCC. Custom web controls thus extend usability and functionality to achieve an optimal visualization result.

Custom web controls are run on the web client and hosted in Runtime Unified. A custom web control can be displayed as an independent Web page in any browser and on any mobile device.



Requirements for a web-based graphic interface

To use a custom web control within WinCC, the control must be provided in a custom web control container. The container is provided on the user side by the custom web control framework and contains components of the graphical user interface (GUI).

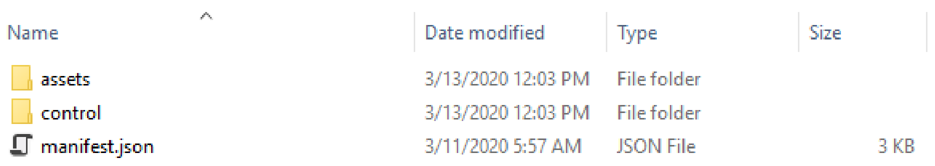
The following requirements apply to a web-based GUI component if it is to be provided in a custom web control container:

- The component must be HTML5-based and interpretable by current browsers.
- The component must be executable exclusively on the client side and without components on the server.
- The component must work without interaction with client-side components outside the container.
- The component must comply with the principle of a Single Page Application (SPA) and fit on a web page. All code (HTML, JavaScript and CSS) must be received when the page is called or dynamically added during user actions. The web page may not reload at any time.
- The component must not know in which environment it is deployed. The component must be executable independent of the environment.
- All communication must take place through communication between client and server.

General structure and folder structure

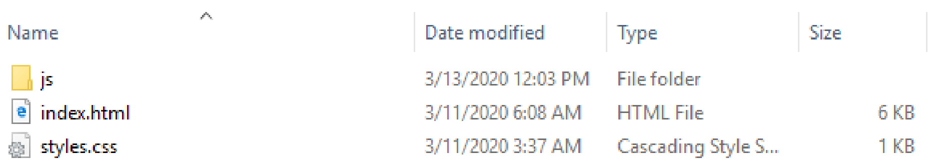
A ready-to-use custom web control must be available as a "*.zip" file containing all graphics and code files used. The structure is divided into two folders, "assets" and "control", and a "*.json" file (manifest.json). The "assets" folder contains a logo that is displayed in the TIA Portal. The "control" folder contains "*.html", "*.js" and "*.css" files, as well as used graphics and icons that the control needs for the display.

Folder structure of a custom web control:



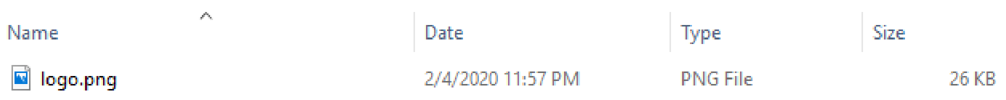
Name	Date modified	Type	Size
assets	3/13/2020 12:03 PM	File folder	
control	3/13/2020 12:03 PM	File folder	
manifest.json	3/11/2020 5:57 AM	JSON File	3 KB

Content of the "Control" folder:



Name	Date modified	Type	Size
js	3/13/2020 12:03 PM	File folder	
index.html	3/11/2020 6:08 AM	HTML File	6 KB
styles.css	3/11/2020 3:37 AM	Cascading Style S...	1 KB

Content of the "Assets" folder:



Name	Date	Type	Size
logo.png	2/4/2020 11:57 PM	PNG File	26 KB

2 Contract-based interaction and the manifest file

To enable the Unified Runtime server to communicate with the provided control, the control must reveal attributes, methods and events to the Unified Runtime server. The sum of the information that the control releases with it is called the "contract". For the custom web control container, this information is contained in a "*.json" file (manifest.json). The manifest file contains two sectors, each of which reveals the elements to the container.

First sector

The first sector is an identity area. This includes, among other things, a name, a version number, a type, the path for the logo, which is displayed in the Unified screen editor of the TIA Portal, and the Start directory. This area also contains the metadata.

Identity type

Each custom control can be referenced via the types and therefore requires a pre-defined structure. Types must follow the 8-4-4-4-12 pattern of a 128-bit integer:

```
"identity":{
  "type": "guid://551BF148-2F0D-4293-99C2-C9C3A1A6A073",
}
```

Identity start directory

The start directory must be specified in order to set the starting point of the custom web control for the browser.

```
"identity":{
  "start": "./control/index.html",
}
```

Metadata

The metadata can contain the name of the author, a date of change, or keywords for the search.

```
"metadata":{
  "author": "Siemens",
  "keywords": [
    "Gauge",
    "GaugeMeter"
  ]
}
```

Second sector

The second sector contains methods, events and properties in the contract area. This sector is called the "contract". Methods, events and properties must be visible to the container on the client side.

Methods

Methods of a custom web control can be used in Unified Scripting to transfer information from the server to the client.

```
"methods": {
  "BlinkZone": {
    "parameters": {
      "zoneIndex": {
        "type": "number"
      }
    },
    "description": "Let the given zone blink."
  }
},
```

Events

Events contain a list of specific events that the control wants to reveal. Each event is defined as a string that represents the event name.

Events are triggered by the custom web control itself at any time. Events can be used in Unified Scripting to transfer information from the client to the server. They can be found in the engineering system under "Properties > Events".

```
"events": {
  "ZoneChanged": {
    "arguments": {
      "zoneIndex": {
        "type": "number"
      }
    },
    "description": "Whenever the zone is changed, this event is raised and 
gives you the new zone index."
  }
},
```

Properties

Properties are written and read within a JavaScript object. Data types and associated values are defined in this range.

```
"properties": {
  "GaugeValue": {
    "type": "number",
    "default": 20
  },
  "MinValue": {
    "type": "number",
    "default": 0
  },
  "MaxValue": {
    "type": "number",
    "default": 50
  },
}
```

Interaction between control and container via the API 3

A single API object is used to enable communication between the custom web control and the Unified Runtime server.

The following requirements apply to this API object:

1. All functionalities that the control needs for independent executability must be available on the client side.
2. The API object must be created and extended using the specific functionalities that the control provides through the manifest file.

As a requirement, a JavaScript file (webcc.min.js) must be integrated, which performs a handshake between the control and the container.

Integrating the WebCC object

Integrating the WebCC object makes the corresponding namespaces available to the custom web control.

```
<!doctype html>
<head>
  <script>…</script>
  <!-- Web Custom Control Facade -->
  <script type= text/JavaScript src='webcc.min.js'>
</head>
```

The API object

The API object is a JavaScript object. The API object represents the interface through which the methods, events and properties of the control are called or received from the framework.

For the initialization of the custom web control, the properties, methods and events of the manifest file must be declared again. You can link the properties, methods and events to your code.

```
<body>
  <script>
    var init = function() { … };
    var _function1 = function() { … };

    var controlAPI: {
      // Functions the control exposes
      function1: _function1,
      // Events the control exposes
      events: [ 'onSomethingChanged' ],
      // Properties the control wants to read and write
      properties: {
      Prop1: 0
      }
    };
  </script>
</body>
```

Note

Declaration

The name must match the name of the manifest file. Only alphanumeric characters of the ASCII character set are permitted for the naming convention.

Initialization of the WebCC object

The WebCC object must have been successfully initialized before the control can be created. To check this, the tag "result" of the parameter "function(result)" is queried. It is used as an indicator and must supply "true" to continue. As an additional parameter, extensions can be called during initialization.

```
WebCC.start( function( result ) {
      if ( result ) {
        controlInit.Local.init()
      } else {
        …
      }
    },
controlInit.ControlApi,
  ['HMI'] };
```

4 Revision of a graphical user interface

Introduction

User interfaces can be used as custom web control by using the framework. This document is intended to describe the process by means of an example and uses a provided user interface. With this user interface, a slider controls the movement of a pointer.

The user interface can be found at the following address: SIOS entry

Conversion of the color coding

The TIA Portal and the manifest file use different color coding. The manifest file is not able to work with hexadecimal values, but only accepts decimal values. For this reason, the author of the manifest file must convert the hexadecimal values as defined in the TIA Portal into decimal values.

To use a web page as a custom web control, the encoding must be converted.

```
function toColor(num) {
  num >>>= 0;
  var b = num & 0xFF,
      g = (num & 0xFF00) >>> 8,
      r = (num & 0xFF0000) >>> 16,
      a = ((num & 0xFF000000) >>> 24) / 255;
  return 'rgba(' + [r, g, b, a].join(',') + ')';
}
```

Defining the default values of properties

Default properties are defined in the TIA Portal and for Runtime projects. These default properties define, among other things, font size, line thickness and value ranges.

Default properties are obtained in Runtime during initialization of the setup. In specific exceptional cases it cannot be guaranteed that the properties will be initialized. To counteract inconsistent system behavior, it is recommended that you define the default properties directly in the control.

```
var defaultProperties = {
  GaugeValue: 20,
  GaugeBackColor: 4294967295,
  Alignment:
  {
    Vertical: 'Center'
  },
  LineThickness: 20,
  FontSize: 16,
  MinValue: 0,
  MaxValue: 50,
  DivisionCount: 5,
  Zones: [
    { Min: 0, Max: 30, StrokeColor: 4281381677 },
    { Min: 30, Max: 40, StrokeColor: 4294958336 },
    { Min: 40, Max: 50, StrokeColor: 4293934654 }
  ]
}
```

Establishing the connection

To be able to function as a custom web control within WinCC, a connection must be initialized via the "WebCC" object. The "WebCC" object must have been successfully initialized before the control can be created.

```
WebCC.start(
    // callback
    function (result) {
      if (result) {
      console.log('connected successfully');
      initializeGauge();
      setProperty({ key: 'GaugeBackColor', value:
WebCC.Properties.GaugeBackColor });
      setProperty({ key: 'Alignment', value: WebCC.Properties.Alignment });
      setProperty({ key: 'LineThickness', value:
WebCC.Properties.LineThickness });
      setProperty({ key: 'DivisionCount', value:
WebCC.Properties.DivisionCount });
      setProperty({ key: 'FontSize', value: WebCC.Properties.FontSize });
      setProperty({ key: 'Zones', value: WebCC.Properties.Zones });
      setProperty({ key: 'MaxValue', value: WebCC.Properties.MaxValue });
      setProperty({ key: 'MinValue', value: WebCC.Properties.MinValue });
      setProperty({ key: 'GaugeValue', value:
WebCC.Properties.GaugeValue });
      // Subscribe for value changes
      WebCC.onPropertyChanged.subscribe(setProperty);
    }
    else {
      console.log('connection failed');
    }
  },
  // contract
  {
    methods: {
    },
    //Events
    events: {
    },
    //Properties
    //////////
    properties: defaultProperties
  },
  //placeholder for later features
  [],
  // timeout
  10000
);
```

Operating the control via WinCC

To operate the custom web Control via WinCC, some functions need to be implemented. The functions show an example of the use of the API object. The sequences of the functions are specific to this example, so they are not explained in detail.

```
function updateAlignment(alignment) {
  const item = document.getElementById('gauge');
  let vertVal = '0';
  let topVal = '0';
  switch (alignment.Vertical) {
    case 'Top':
      break;
    case 'Center':
      topVal = '50%';
      vertVal = '-50%';
      break;
    case 'Bottom':
      topVal = 'inherit';
      break;
    }
    item.style.top = topVal;
    item.style.transform = 'translate(0,' + vertVal + ')';
  }

  function updateLabels() {
    const labels = new Array(WebCC.Properties.DivisionCount).fill(0).map(
      (x, i) => (i + 1) * (WebCC.Properties.MaxValue -
WebCC.Properties.MinValue) /       WebCC.Properties.DivisionCount +
WebCC.Properties.MinValue
    );
    labels.unshift(WebCC.Properties.MinValue);
    gauge.setOptions({
      staticLabels: {
        font: WebCC.Properties.FontSize + 'px "Siemens Sans"',
        labels: labels
      }
    });
  }

  function updateZones(zones) {
    gauge.setOptions({
      staticZones: zones.map(item => {
       return { strokeStyle: toColor(item.StrokeColor), min: item.Min, max:
item.Max };
      })
    });
  }

  function setProperty(data) {
    switch (data.key) {
    case 'GaugeValue':
      updateValue(data.value);
      break;
    case 'GaugeBackColor':
      document.body.style.backgroundColor = toColor(data.value);
      break;
    case 'Alignment':
      updateAlignment(data.value);
      break;
    case 'LineThickness':
        gauge.setOptions({ lineWidth: data.value / 100 });
        break;
      case 'FontSize':
        updateLabels();
        break;
      case 'MinValue':
        gauge.setMinValue(data.value);
        updateLabels();
        break;
      case 'MaxValue':
        gauge.maxValue = data.value;
        updateLabels();
        break;
      case 'DivisionCount':
        updateLabels();
        break;
      case 'Zones':
        updateZones(data.value);
        break;
    }
  }
```

Creating the ZIP file

To use the custom web control in WinCC, the hierarchy of folders and files must be compressed. For WinCC it is necessary that this data is available in ZIP format. The name of the ZIP file must match the GUID, for example, "{551BF148-2F0D-4293-8E10-C9C3A1A6A073}.zip".

Note

GUID generation

A GUID can be generated in Visual Studio via "Tools > Create GUID" or other GUID generators.

Restrictions 5

Some restrictions regarding data types must be observed when using custom web controls. Tags that use large data types can result in rounding errors when used in custom web controls. Data types that are affected by this restriction are, for example, the data types "DInt" or "Date".

Use these data types at your discretion.

6 Installing a custom web control

Use

In WinCC, you can use custom web controls that have been created externally. Custom web controls are freely-programmable and serve as a specific solution that goes beyond the functionalities of the toolbox provided. Like all other tools, custom web controls are used within screens and displayed in Runtime.

Requirement

- A WinCC project has been created
- A screen has been created

Procedure

To install custom web controls for a TIA Portal project, proceed as follows:

1. Open the directory of your project.
2. Open the "UserFiles" subfolder.
3. Create a folder with the name "CustomControls".
4. Store the created program as *.zip archive in the "CustomControls" folder.

To install custom web controls across projects for the TIA Portal, proceed as follows:

1. Open the TIA Portal installation directory.
2. Navigate to the subdirectory "Siemens/Automation/Portal Vxx/Data/Hmi/CustomControls".
3. Store the created program as *.zip archive in the "CustomControls" folder.

Note

Update

To use custom web control in the tool list, you need to close and restart the WinCC application.

---

7 Updating a custom web control

Introduction

If you want to update a custom control already installed with a new version, you must take a few steps to ensure that the update is performed without errors.

Procedure

To update a custom web control, proceed as follows:

1. Delete all uses of the custom control within the project that contains the custom control.
2. Remove the outdated version of the custom control from the project directory.
3. Copy the new version of the custom control as *.zip archive into the subfolder "UserFiles/CustomControls" in the directory of your project.
4. Restart the WinCC application.

Note

You can specify the most current version of your custom web control by editing the manifest file and reflect it within WinCC. To do this, change the attribute "displayname", e.g. "displayname": "MyControlV1.2".

Result

The custom web control has been updated.