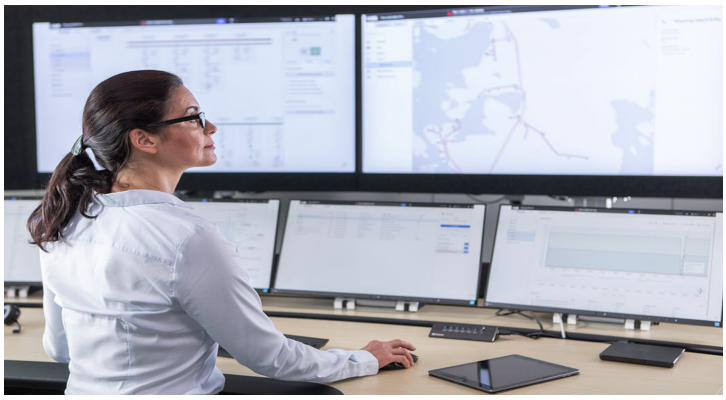
GRID AUTOMATION PRODUCTS

MicroSCADA X SYS600 10.2

View Writer's Guide



Document ID: 1MRK 511 506-UEN
Issued: March 2021
Revision: A
Product version: 10.2

© 2021 Hitachi Power Grids. All rights reserved.

Table of contents

© 2021 Hitachi Power Grids. All rights reserved.

© 2021 Hitachi Power Grids. All rights reserved.

Section 1 Introduction

This guide contains information on writing view files. It explains all major features in the view format with examples. One or more views produce everything you see in the Workplace X. Views are the files that define the user interface, the communication between the client and the server, and the UI logic. Engineers and system designers use views to create versatile and rich user interfaces and complex interactions.

Views can be used to build customization to existing UI applications or to implement new major UI features to SYS600. For example, it is possible to implement dashboard views or customize the behavior of the UI. Also, it is possible to implement new reusable functions. New views can be added to the existing user interface and certain aspects of the current user interface can be altered by writing a new view and extending the existing ones.

The view files are XML documents. To understand this document, the reader must be familiar with XML, HTML, CSS, and JavaScript. Many high-quality tutorials about these languages/ technologies are available Online. Examples in the tutorial are explained to understand them but may require some basic knowledge of these technologies.

All examples in this document are licensed under CC0 license (https://creativecommons.org/share-your-work/public-domain/cc0/), which allows anyone to use examples in any of their works in any way they wish without any obligations. Other parts of this document has the same license as all other documents included in the product.

1.1 Revision history

Revision	Version number	Date	History
A	10.2	31.03.2021	Updates to MicroSCADA X SYS600 10.2 Workplace X View Writer's Guide

© 2021 Hitachi Power Grids. All rights reserved.

Section 2 Getting started

This section describes a simple example, `Hello world` and how to get that up and running. For more information, see Section 2.1 — Section 2.4.

2.1 First view

An example of, `Hello world!` is shown in Example 1. This is a simple view of the static content. The helloworld.xml file can be stored in this path: c:\sc\apl\<myapplication>\views \dialogs.

Hello world example

Example 1

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="test-view" />
    </head>
    <body>
Hello world!
    </body>
</document>
```

Each view is an XML file.

Once the file is stored in the application library, it can be accessed without restarting SYS600. If the view file is only stored in the disk, it will not be automatically visible in the UI. There are two ways to access the view.

- It can be accessed by modifying the browser's URL.
- It can be added to the left-side menu structure.

2.2 Address

It is possible to navigate to an arbitrary view by changing the URL. Normally, the URL looks like this:

https://<computer name>/main/desktop-layout

In this example, SYS600 is installed to the same machine where the browser is executed. The path component of the URL, **main/desktop-layout** , has two parts:

- Template name
- View name

It is possible to use an empty template to use an arbitrary view. The empty template is called blank. To open the `Hello world!` -view, use the URL https://<computer name>/blank/Example1.

This empty template is good for testing, development, and various integrations to third-party systems, but for regular use, content must have been integrated to the rest of the UI. Figure 1 shows how Example 1 view looks when it is opened with a browser.

© 2021 Hitachi Power Grids. All rights reserved.

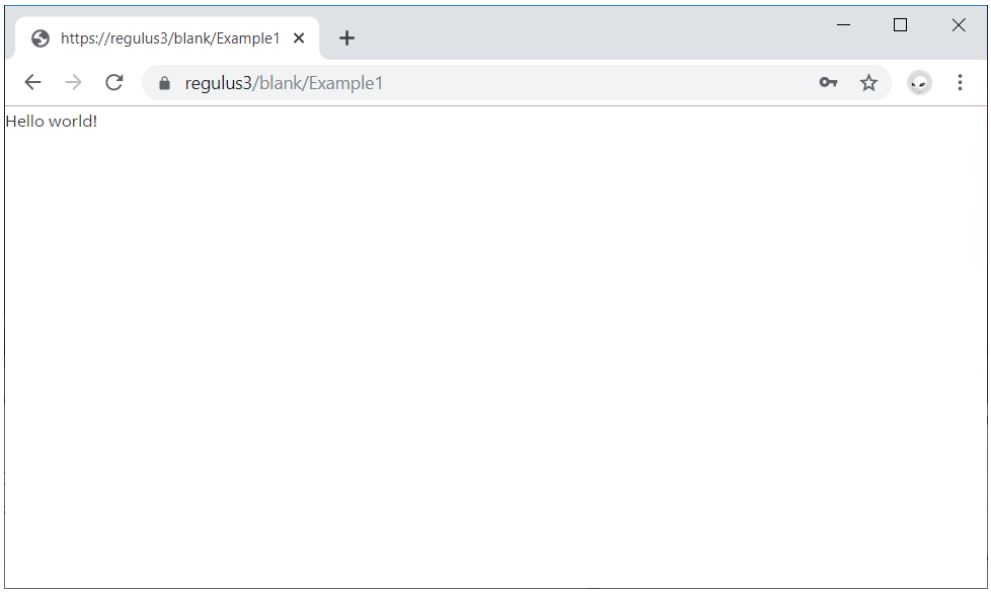


Figure 1: Hello world example

If the view contains errors, an error popup is shown. Example of an error popup is shown in Figure 2.

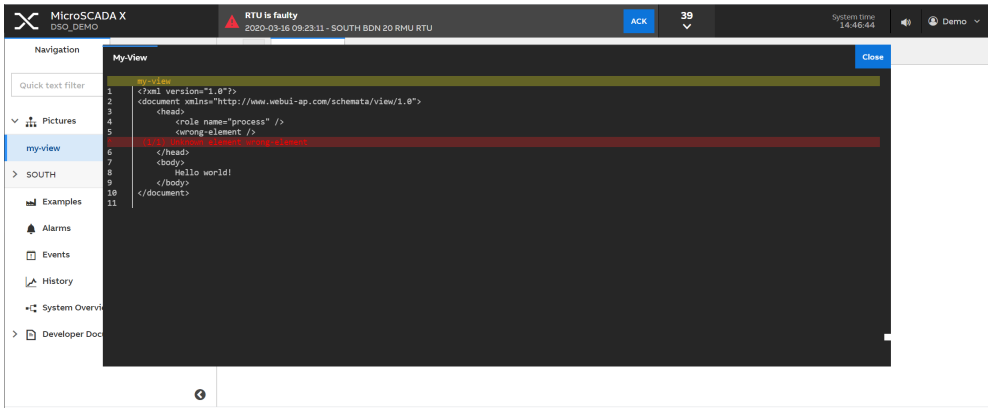


Figure 2: Error window

2.3 Menu structure

It is possible to add own views to the menu structure. A new main menu item can be created by adding a new driver view, for example, see Example 2. This can be stored in \sc\apl\<myapplication>\views\dialogs\test-menu-item.xml. Now, all views with the role element test-view in the body are visible under the new **Test Views** menu item. Example 3 shows an updated example with a role element. This can be stored in \sc\apl\<myapplication>\views\dialogs\testview.xml. In the following examples, all view files can be stored in the same directory \sc\apl\<myapplication>\views\dialogs. See Section 8 for a detailed explanation of the structure of the Workplace X content.

This section shows how to get items to the main menu. This is useful for trying things out with the Workplace X. All concepts used in this section are described more thoroughly later in this guide. Roles are described in Section 4 and the content of the top menu driver view is described in Section 11.

© 2021 Hitachi Power Grids. All rights reserved.

The driver view for a new top-level menu

Example 2

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="main-menu-item" />
        <variable type="external" name="menuItem" />
        <script type="text/javascript">
this.set("menuItem", {
"type": "queryViews",
"order": 2000,
"role": "test-view",
"icon": "webui_icon_24/ui_plugin",
"title": i18n._("Test Views")
});
        </script>
    </head>
</document>
```

An example view with a role

Example 3

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="test-view" />
    </head>
    <body>
Hello world!
    </body>
</document>
```

2.4 Programming model

Workplace X views combine declarative method and dataflow programming with scripting. In many places in view files, curly brace expressions can be used instead of a value. The value will be resolved during runtime based on the statement inside the curly braces. If the value of the statement changes, the change is automatically propagated.

Dynamics can be used in most parts of the views. Table 1 lists the use of dynamics.

© 2021 Hitachi Power Grids. All rights reserved.

Table 1: Use of dynamics

Dynamics	Examples
All HTML, SVG, and XLINK attribute values	<h:div class=”{variable}”>…</h:div>
All HTML and SVG text node values	{variable}
All styling attribute values	…
All naming attribute values	…
Most widget attributes	<phasor unit="kV" s:width="400px" s:height="400px" max="100"> <phasorvalue color="red" value="{phase1magnitude}" angle="{phase1angle}">>phasorvalue> <phasorvalue color="green" value="{phase2magnitude}" angle="{phase2angle}">>phasorvalue> <phasorvalue color="blue" value="{phase3magnitude}" angle="{phase3angle}">>phasorvalue> >phasor>

The declarative method is used for many purposes. For example, consider the complex spreadsheets that can be created without the need for programming. However, there are some limitations that must be considered.

The declarative method can be used for manipulating attributes and text content in the view. It cannot be used for creating components, but can be used for showing and hiding elements.

If the required functionality of the view is simple, select a declarative method. If the required functionality of the view is complex with a declarative method, select a procedural method.

The declarative method works for the following tasks:

- Setting values of variables, attributes, and text nodes
- Invoking scripts implemented with SCIL and JavaScript
- Navigating

Different methods can also be used together using declarative method to invoke JavaScript methods and JavaScript for setting values of data flow variables. Example 4 shows a simple view with two toggling buttons. The view is completely declarative. The same logic can be implemented with JavaScript as shown in Example 5.

View with toggling buttons

Example 4

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <title>First View</title>
        <variable type="internal" name="myvariable" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <d:button disabled="{not myvariable}">Click Me!
            <d:action type="set" name="myvariable" value="false" />
        </d:button>
        <d:button disabled="{myvariable}">Click Me Too!
            <d:action type="set" name="myvariable" value="true" />
        </d:button>
    </d:body>
</document>
```

View with toggling buttons implemented with JavaScript

© 2021 Hitachi Power Grids. All rights reserved.

Example 5

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <script type="client">
var rootNode = self.getRootNode();
var buttons = rootNode.querySelectorAll("button");
buttons[0].addEventListener("click", function() {
buttons[0].disabled = true;
buttons[1].disabled = false;
});
buttons[1].addEventListener("click", function() {
buttons[0].disabled = false;
buttons[1].disabled = true;
});
        </script>
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <d:button>Click Me!</d:button>
        <d:button disabled="disabled">Click Me Too!</d:button>
    </d:body>
</document>
```

2.5 Reference documentation

To access Workplace X reference documentation, go to c:\Program Files\ABB\MicroSCADA Pro\WebUI folder and execute enable_documentation.bat as administrator. Once MicroSCADA X is restarted, reference documentation will appear to the navigation tree. Documentation can be disabled with disable_documentation.bat.

Developer Documentation contains information about user interface widgets, JavaScript API, and Service API.

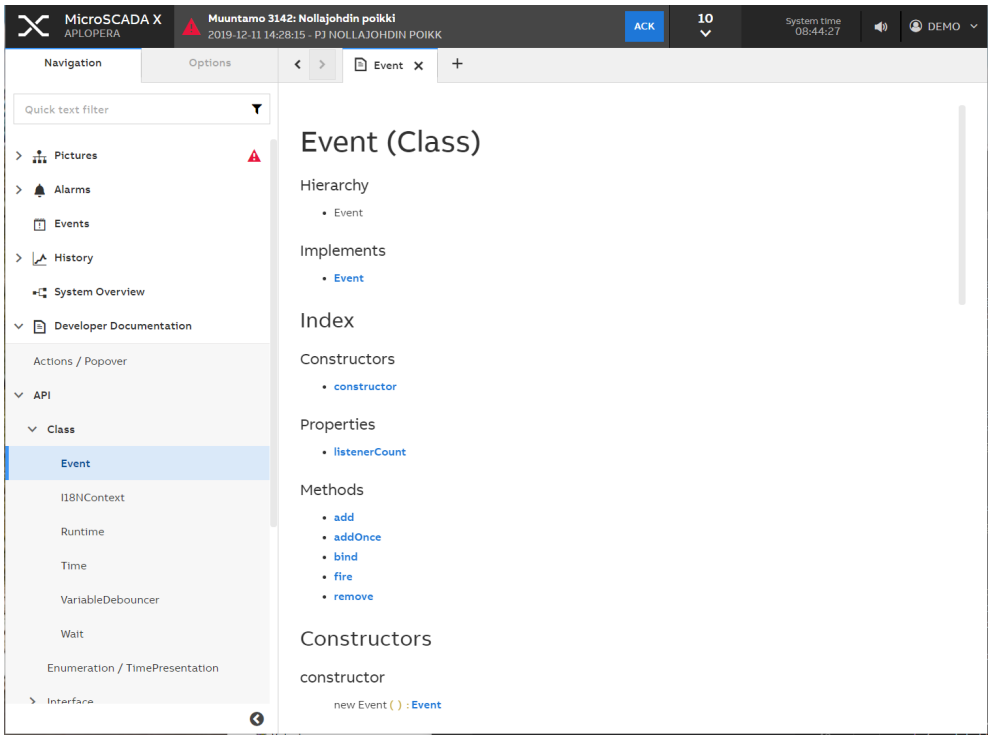


Figure 3: User interface opened with reference documentation enabled

© 2021 Hitachi Power Grids. All rights reserved.

2.6 Change log

Developer documentation contains a change log which lists all the changes and the new features that might affect the views created for earlier versions. Change log has the following three sections:

- New features: New feature are the new additions. They should not break any existing functionality.
- Bug fixes: Bug fixes are changes to the existing functionality in way that should not break any functionality. Though the earlier behavior had been incorrect, it is possible but does not ensure that a bug fix has modified the behavior of an existing view.
- API changes: API changes are the actual changes to the existing behavior, and they may affect the existing views.

© 2021 Hitachi Power Grids. All rights reserved.

Section 3 Design principles and guidelines

This section includes the most important design goals and guidelines that should be considered while extending Workplace X user interface. It is important to maintain the consistency in the user experience by following the guidelines, and by taking care of the product and company brand.

However, note that this is not a complete and detailed visual style guide. With any function, view or user interface element, consider user needs and the consistency with the rest of the product. Try to copy the existing look and feel whenever it is possible.

3.1 Design goals

Workplace X is designed to fulfill the following design goals:

Clarity and glanceability

- Make key elements stand out: all are not equal
- Identifiable elements, clear status, good contrast, and good font size
- Easy and simple navigation between key information

Meaningful beauty

- Modern and high quality visual design
- Optimized visualizations

Agile adaptivity

- Make it easy to select the content that is important
- Modern search and browse experience
- Optimize user interface for different context and use cases

3.2 Layout and main elements

While adding a new feature to the user interface, consider the **Main element** to which the new feature belongs to.

© 2021 Hitachi Power Grids. All rights reserved.

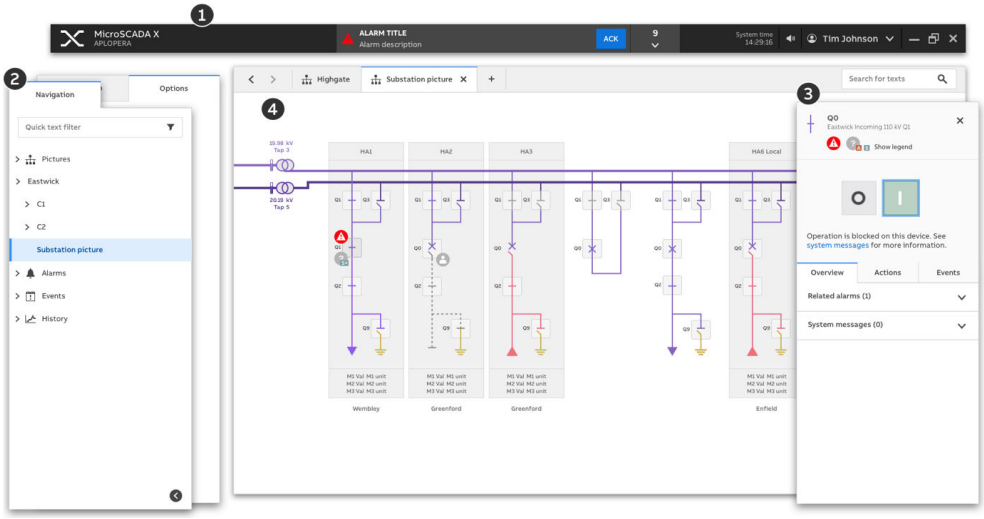


Figure 4: Main elements in Workplace X user interface

1. Global header

Global header is always visible and it includes the *product ID* (product name and logo), *system time*, *sound on/off*, and *user software settings*. In addition, there is an opening list of *global alarms* with acknowledge function and access to all alarms in the center.

It is possible to customize the global header using extension mechanisms.

2. Left pane

- **Navigation** tab for main navigation. All pages must be available through this navigation tree.
- **Options** tab for content-specific controls. Use **Options** tab to control the main content. For example, a user can add notes or save some zoom areas. Some control-specific functions can also be in the main content area, such as table column settings and filters.

Left pane is collapsible.

3. Content area

- **Content area** tabs - tab name shows the page (navigation tree item) selected by the user on the main navigation. On each tab, a user can navigate to different pages with the main navigation. Click/tap on the '+' tab to open a new tab. Use the **Search tool** located on the tab bar to search text in the main content.
- Main content provides overviews into the processes. In addition, it also provides content-specific controls, such as table filter, have a natural location here, instead of **Options** tab.

4. Right pane

1. Right pane is used to control the process objects.
2. Right pane is used to see more details of the items selected in the content area.

Attention should be given to the correct structure of the pane.

© 2021 Hitachi Power Grids. All rights reserved.

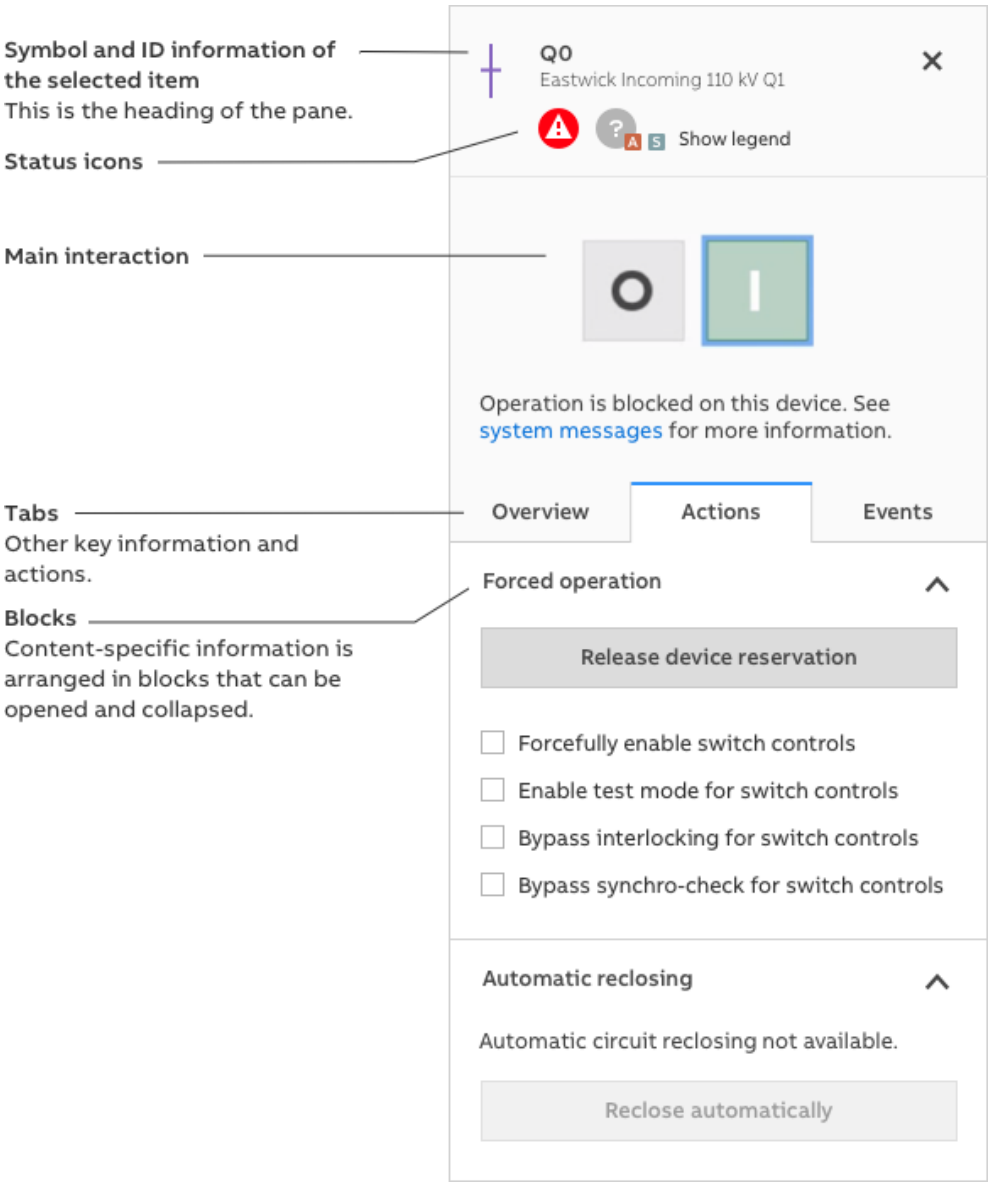


Figure 5: Right pane layout

3.3 UI widgets

For any information or item on the user interface, use the existing and built-in UI widgets. For more details, refer to *Reference documentation*, which is available in the MicroSCADA installation package.

Avoid creating new UI widgets. To create a new UI widget or item, apply the color palette and the applicable text styles. Follow the existing and common interaction patterns if the item is interactive. For example, ensure that **Normal**, **Hover**, **Active**, and **Disabled** states are included. Apply similar state styles used in the existing UI widgets.

© 2021 Hitachi Power Grids. All rights reserved.

Figure 6: States for primary button

In buttons, use the clear and short labels that refer to the action that is committed. Always use verb in the label, for example, **Save** or **Delete all**.

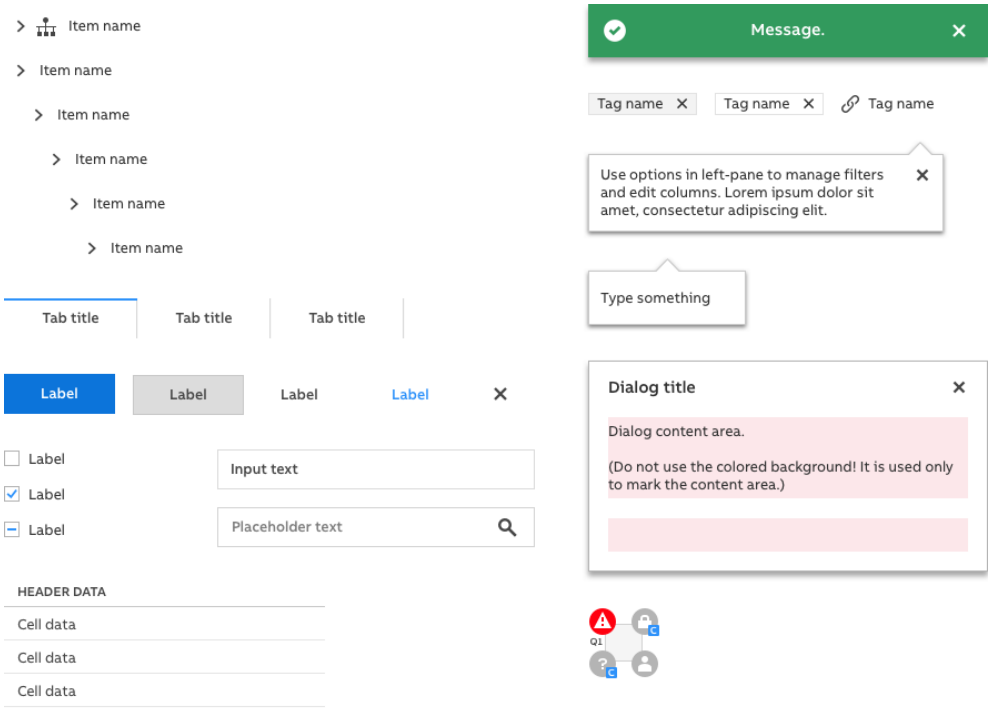


Figure 7: UI widget examples

3.4 Workflow and navigation

Support fluency in user's workflow. Consider the order in which the information and functions need to be. For example, different controls, such as buttons, can be grouped and placed according to their purpose. Visual clues should be given to the users on how to proceed and the attention that should be given to.

For example, the blue primary button should be used only for the main function in the view, while the other buttons should be used for normal or discreet.

© 2021 Hitachi Power Grids. All rights reserved.

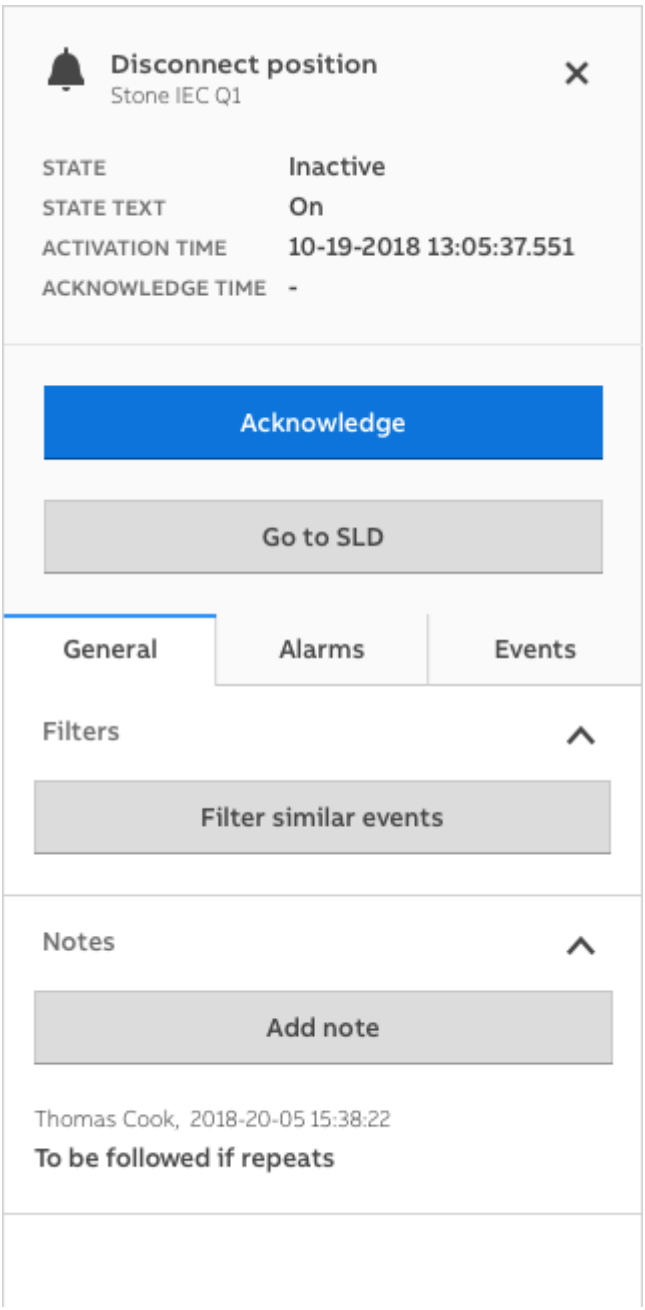


Figure 8: Primary button example
Example of using the primary button for the main function. In the alarm details pane, it is used to acknowledge the alarm.

Help the user to reduce time to navigate and filter key information of the system. If possible, provide the required information directly in the context. In some cases, using links or tooltips can be useful. Add a legend to the context when needed.

© 2021 Hitachi Power Grids. All rights reserved.

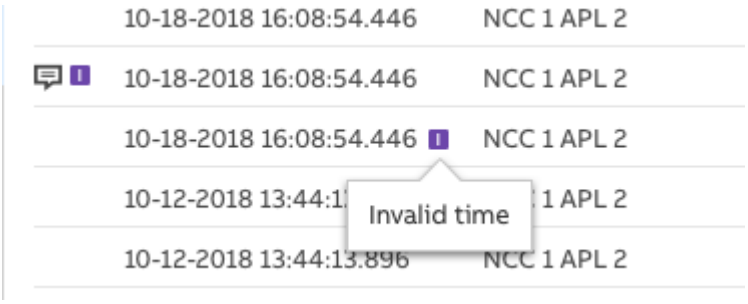


Figure 9: Add-on legend in a tooltip

3.5 Visual hierarchy

While designing a view or any bigger entity on the user interface, consider the visual hierarchy, that is, what needs to stand out and what requires less attention. Avoid bold colors and flashing elements. Group the items that belong to the same category and consider good visual view by creating clusters and summary views.

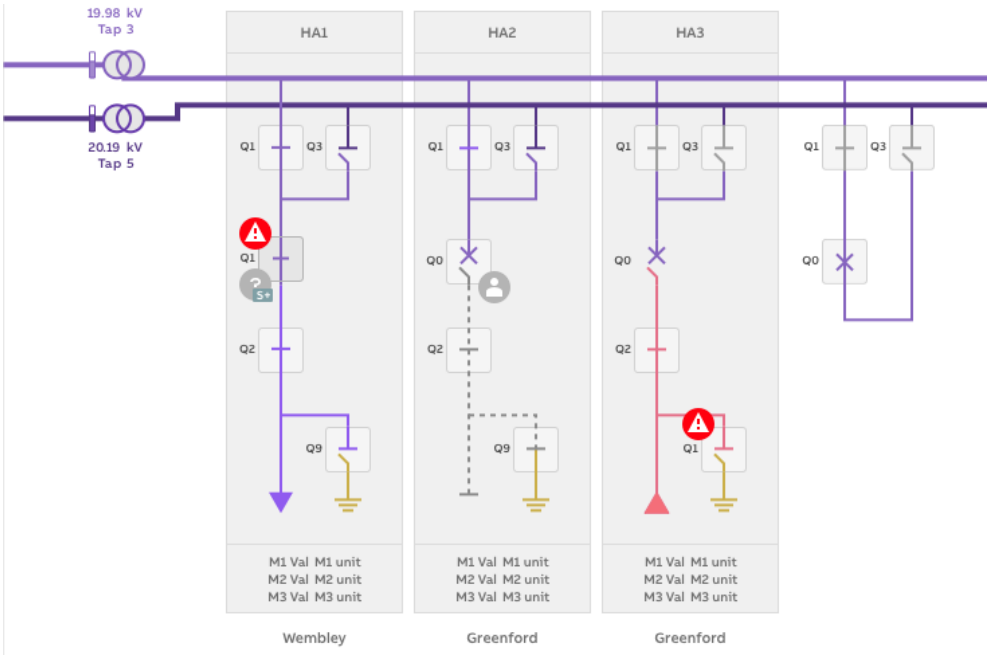


Figure 10: Example of highlighting the most important
In the SLD view, the alarms need to be the most distinguishable items.

3.6 Color palette

To create new components or items, use the color palette. Consider peaceful and neutral overall outlook where the essential information stands out.

Avoid relying on colors, for example, in charts to convey the meaning. Try to provide the textual explanation whenever it is possible. Use contrast in the elements (apply Web Content Accessibility Guidelines (W3C) for contrast). Consider challenging work environments and users with impaired vision.

© 2021 Hitachi Power Grids. All rights reserved.

The most typical colors and usage are shown in Figure 11.

Figure 11: Common colors

The ranges of blue and grey shades in Figure 12 are the foundation of all components. Use these colors for backgrounds, details, and controls. The light blue shades are used as accent colors, for highlighting information, for selected items (Blue 1 and Blue 4 in selected state), and in data visualizations.

Highlight colors

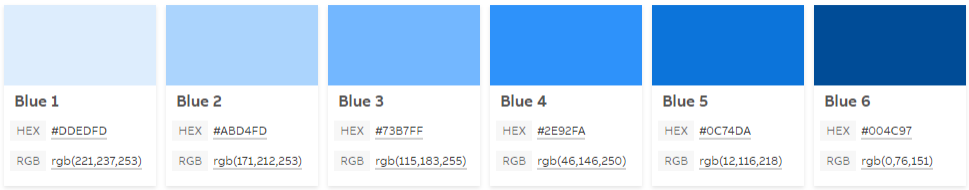


Figure 12: Highlight colors

Grey scale colors

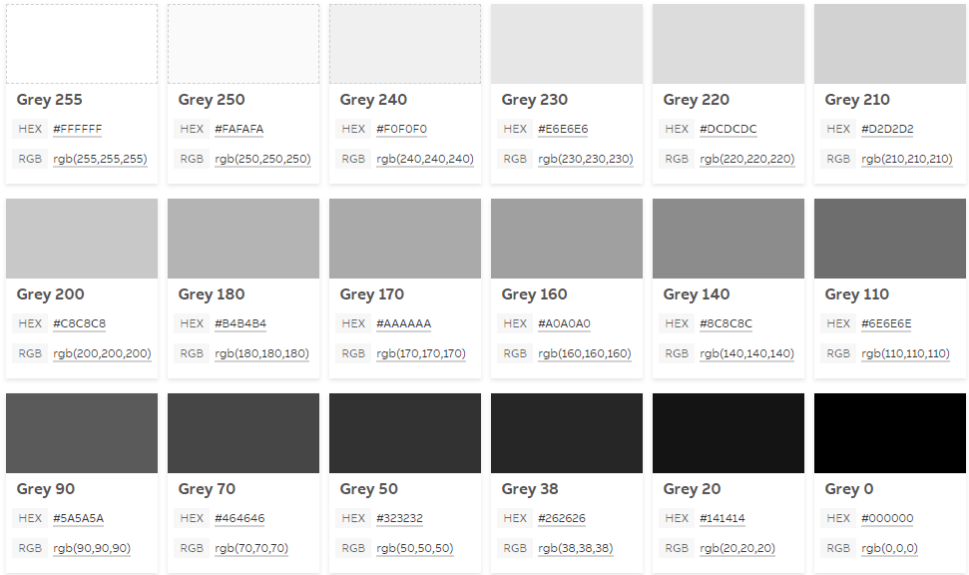


Figure 13: Grey scale colors

Additional colors

Use additional colors sparingly, and only in small areas. Use these colors in charts and in other graphical items to highlight essential information.

© 2021 Hitachi Power Grids. All rights reserved.

Figure 14: Additional colors

Alarm colors

Use alarm colors only for alarming information, such as system alarms and in some notifications.

Alarm colors should only be used in small areas.

© 2021 Hitachi Power Grids. All rights reserved.

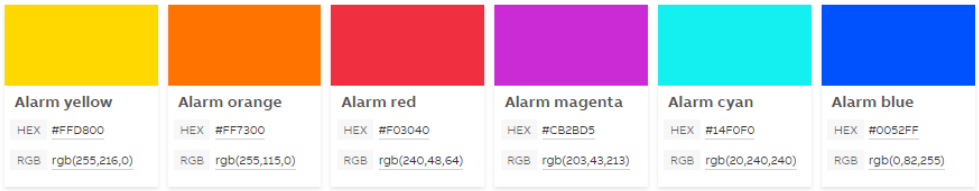


Figure 15: Alarm colors

3.7 Typography and text styles

ABBVoice is the font used for text on the user interface. Do not use other fonts. Apply the following text styles for their intended purpose.

Style	Specification	Used for
Heading-1, H1	ABBVoice, Regular Font-size 30px, Line-height 30px, Body-color (Grey-70)	As a heading in Main content, if it is needed.
Heading-3, H3	ABBVoice, Medium Font-size 15px, Line-height 19px, Body-color (Grey-70)	Dialog heading
Heading-4, H4	ABBVoice, Medium Font-size 14px, Line-height 18px, Body-color (Grey-70)	Right pane heading
Body text	ABBVoice, Regular Font-size 13px, Line-height 16px, Body-color (Grey-70)	Body text
Link text	ABBVoice, Medium Font-size 13px, Line-height 16px, Blue-5	Links, also with body text
Small text	ABBVoice, Light Font-size 11px, Line-height 14px, Body-color (Grey-70)	Subheading, e.g. with Right pane heading

Figure 16: Typography and text styles

3.8 Icons

Consider using icons if it makes the user interface more intuitive, recognizable, and efficient. Icon next to a label can gather attention to key information or function. However, avoid relying on untypical icons to convey meaning, since it can be difficult to understand.

Use icons from the Icon library included in the MicroSCADA installation. All icons have versions in three different sizes: 16, 20, and 26 pixels.

© 2021 Hitachi Power Grids. All rights reserved.

Figure 17: Icon examples

Alarm emblem

For alarm emblem, there are two different icons in use. The triangle-shaped icon is used in most places, for example, in global header. The round icon is only used as part of the device symbol in SLD picture.

Do not use this alarm icon anywhere else.

	Purpose	Colors
	Used only as part of the device symbol, in SLD picture, to indicate alarm.	Alarm-red (#FF000F) and Grey-255 (#FFFFFF)
	Used in other places, to indicate alarm.	Alarm-red (#FF000F) and Grey-255 (#FFFFFF)

Figure 18: Alarm emblem

3.9 Design checklist

A user needs to consider all the below points while designing new views to Workplace X.

- Whether all the information and functions are located in the correct **Main elements** according to their purposes?
- Whether all items are using the existing UI widgets on the user interface?
- If it is not possible to utilize the current components in the design, are all the colors from the color palette, and all the texts are according to the current text styles?
- Whether all items have enough contrast to the background? Apply Web Content Accessibility Guidelines (W3C) for contrast.
- Whether all interactive items have all the required interaction states?
- Is the essential standing out on each view?
- Is user workflow supported in the way in which different items have been arranged? Is the key action most visible element?
- Are all icons understandable and adding value, and not exist for decoration?

3.10 Programming style

There are certain programming rules that are followed by the system.

JavaScript does not distinguish between private methods, public methods, and fields. All names that start with an underscore are considered private and should not be accessed outside an object. Its content may not be consistent and may change without notice between versions.

© 2021 Hitachi Power Grids. All rights reserved.

The file name for views and modules over all the libraries are in the same namespace. When a new feature is developed, it is advisable to place all modules and views related to a single feature under a single folder. The namespace for variables within a view is local, and therefore there is no need to prefix these.

For variable names in views, write in camel case (for example, **variableName**) and for extension names, write in kebab case (for example, **extension-name**). For file names, use kebab case. All names can have any UTF-8 characters. For compatibility, it is advisable to use only characters and numbers included in ASCII character set. Otherwise, names are hard to type with keyboards from different regions.

In general, the main views are designed in such a way that it doesn't assume to be opened from one certain location. This allows for better integration of system content. For example, it should be possible to embed list display or map display to other displays.

During SCIL scripts design, all required permission checks must be implemented in SCIL in conjunction with the authorize attribute. An expert user can invoke server scripts, even though controls for doing so are not exposed in the user interface. Since the traffic is based in HTTP(S) protocol, it is easier to access the system from remote locations. For usability purposes, all access checks are done with JavaScript. A malicious user can easily bypass any such check.

© 2021 Hitachi Power Grids. All rights reserved.

Section 4 Views

Views are XML documents. The root node of the view is document and the namespace for that node is http://www.webui-ap.com/schemata/view/1.0. The structure of a view file is similar to HTML, but not the same. Workplace X uses XML namespaces extensively. Namespace is the method for separating different content in an XML document. When HTML elements are added, these must be in the XHTML namespace; Otherwise, those elements are ignored. If an attribute does not have a defined namespace, it is interpreted to be a part of the parent element's namespace. For example, in the markup <h:div class="foo">…</h:div>, the attribute class does not belong to any namespace, as defined in the XML specification, but in Workplace X, it is interpreted to be in the same namespace as the div element.

A simple view is shown in Example 6. In the body section, the document namespaces are changed so that the document namespace will have the prefix *d* and the default namespace will be the XHTML namespace. This is useful since typically most of the content of the body is either from XHTML or SVG namespace.

Simple view

Example 6

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="foo" default="'Hello World!'" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <p>{foo}</p>
    </d:body>
</document>
```

View has two sections:

- Head section
- Body section

The head section contains various declarations related to the view. Table 2 lists all elements that are valid for the head section.

Table 2: Elements in the head section

Element	Reference
Variables	Section 4.9
Scripts	Section 4.2 and Section 4.3
Style	Section 4.4
metadata	Section 4.11
Role	Section 4.10
Mapping	Section 4.5
Value list	Section 4.6
Title	Section 4.1
Use	Section 6.1
Service	Section 10

Table 3 lists the attributes of the document element.

© 2021 Hitachi Power Grids. All rights reserved.

Table 3: Attributes of the document element

Attribute	Description	Remarks
nature	Describes whether the views' body contains the HTML or SVG content. Possible URL of namespaces is either http://www.w3.org/1999/xhtml or http://www.w3.org/2000/svg.	Optional The default value is the HTML namespace.
zoomable	Describes whether the view can be zoomed or not. The default value is false. For more information about zooming, see Section 5.2.	Optional This is applicable for SVG content.
zoomScaleFactor	Defines the zoomable area of the view. The default value is 1. With the default value accessible, SVG coordinates are roughly from -20000 to +20000 coordinate points. The value range is multiplied with the zoomScaleFactor. For more information about zooming, see Section 5.2.	Optional
i18n:translate	Defines whether the view needs translation. See Section 9.	Optional

The body section contains content from either the XHTML or SVG namespace. The body section can contain certain elements from the document namespace. Even though the content of the body is similar to HTML or SVG, it is still XML. The difference is, for example, that in XML all tags must be closed and all attributes require values. For more details on the content of the body, see Section 5.

4.1 Title

Each view can have a title, see Example 7. If a view does not have a title, then the view name is used as a title. The title is accessed with JavaScript, see Example 8. This title is displayed, for example, in the tab's header.

View with a title

Example 7

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <title>My Title</title>
    </head>
</document>
```

Accessing a view title with JavaScript

Example 8

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <title>My Title</title>
        <script type="client">
alert(runtime.getMainDialog().getType().getTitle());
        </script>
    </head>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

4.2 Client scripts

It is possible to add the JavaScript code in views. The code is executed when the DOM tree of the view is built, but the view is not yet placed in the document DOM tree. Table 4 lists the JavaScript variables that are available in the JavaScript code. Also, scripts can access all standard JavaScript objects as defined by the HTML specification. JavaScript types and methods are documented in a separate document.

Table 4: Variables available for JavaScript blocks in views

Name	Type	Description
view	ViewUserContext	Gives access to a set of methods that can be used for accessing view's content. Views can also store its own methods to this object.
runtime	Runtime	Runtime object is the main object of the system. For example, it can be used to open new views.
i18n	Translations	Translation object is used to translate strings. For comprehensive description on translation, see Section 9.

Example 9 shows a view that opens another view in an element. All scripts are executed in the ECMAScript 5 strict mode.

A view using JavaScript

Example 9

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <script type="client">
runtime.showDialog(view, "Example1", {}, view.getElementById("e"));
        </script>
    </head>
    <body xmlns:h="http://www.w3.org/1999/xhtml">
        <h:div id="e" />
    </body>
</document>
```

All available objects are documents. Documentation is available in JavaScript API section in the reference documentation.

4.3 Server scripts

It is possible to include SCIL scripts in a view. Example 10 shows a simple view with a SCIL script. When the button is clicked, a message is written to the SYS600 Notify window.

View containing SCIL script

Example 10

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <script type="server" name="my-script" close="true"
authorize="GENERAL > 3">
@res = console_output("Hello world again!")
        </script>
    </head>
    <body>
        <button>Click me!
            <action type="invoke" name="my-script" />
        </button>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    </body>
</document>
-------------------------------------------------
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
<head>
    <variable type="internal" name="SS" />
    <variable type="internal" name="SP" />
    <script type="server" name="my-script" authorize="GENERAL >= 0"
    result="{SS: SS, SP: SP}" init="true">
    #return LIST(SS=APL:BSS, SP=APL:BSP)
    </script>
        <script type="client">
        self.invoke("my-script", {} ,
        response => {
        alert(JSON.stringify(response));
        });
    </script>
</head>
<body>
</body>
</document>
```

Table 5 lists the attributes of the script element with type server.

Table 5: Attributes of the server script element

Attribute	Description	Remarks
name	Name of the script	Required
authorize	Required authorization group and level	Required
result	Result expression defining how to handle the result of the SCIL script. For result expressions, see Section 4.8.	Optional
init	If true, this script is executed before the view is opened.	Optional

View expressions can be used in SCIL scripts with curly bracket notation. Additional parameters can be given as named parameters in the invoke action. See Example 11.

The authorize attribute has an expression that states if the user has permissions to execute a command. The format of the attribute is GROUP OPERATOR LEVEL. GROUP is one of the SYS600 groups, such as ALARM_HANDLING or GENERAL. Table 6 lists the operators of the authorize attribute. Typically, only > and >= operators are needed. LEVEL is an integer. SYS600 uses levels from 0 to 5. For example, if the authorization attribute is ALARM_HANDLING >= 2, then the SCIL script is executed only if the user has engineering permissions for alarms.

Table 6: Operators of the authorize attribute

Operator	Description
<	Less than
<=	Less than or equals
=	Equals
>	Greater than
>=	Greater than or equals
!=	Not equal to
<>	Not equal to

SCIL script that uses a parameter

© 2021 Hitachi Power Grids. All rights reserved.

Example 11

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="message" default="'Hello'" />
        <script type="server" name="my-script" authorize="ENGINEERING >
3">
@res = console_output({message} + " " + {message2})
        </script>
    </head>
    <body>
        <button>Click me!
            <action type="invoke" name="my-script">
                <parameter name="message2" value="'Maailma!'" />
            </action>
        </button>
    </body>
</document>
```

When the SCIL script returns a result, SCIL datatypes are converted to JSON datatypes. The conversion is described in Table 7. When JavaScript parameters are passed to a SCIL script, the conversion happens vice-versa. These rules are described in Table 8. Since there are some differences between type systems, conversion has some discrepancies.

Table 7: Conversion from SCIL datatypes to JavaScript datatypes

SCIL	JavaScript	Remarks
Boolean	Boolean	
Integer	Number	
Real	Number	
String	String	
List	Object	
Vector	Array	When converting from SCIL to JSON, the arrays are shifted to 0-based arrays.
Bit string	Array of numbers	
Byte string	Array of numbers	
Time	String in RFC 3339 format	

Table 8: Conversion from JavaScript databytes to SCIL datatypes

JavaScript	SCIL	Remarks
Boolean	Boolean	
Number	Number or real	If the value is a whole number, it is converted to a number, otherwise, to a real. If the number is too large or small, the largest or smallest numbers are used. Floating point numbers are converted from double precision to single precision.
String	String	The data, if converted to a non-Unicode format. If the characters cannot be converted, the characters are replaced with a dummy character.
Object	List	If the keys in the object are not valid identifiers in a SCIL list, non-valid characters are replaced with a dummy character.
Array	Vector	

© 2021 Hitachi Power Grids. All rights reserved.

4.4 Styling

Styling is done using CSS. There are three ways for adding styles:

- Using a global CSS or Less file
- Using a style-element in the head element
- Using a styling schema

The styling schema is used to add styles for individual elements. In Example 12, OV is colored red using the styling schema. This example assumes that the measurement signal ESTHA1_MEC:P20 is existing.

Defining styles with a styling attribute

Example 12

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <details n:LN="ESTHA1_MEC">
            <summary>Interesting value</summary>
            <div>
                <span s:color="red">{`20.OV`}</span> {`20.ST`} {`20.OX`}</
div>
        </details>
    </d:body>
</document>
```

In this example, the span elements' styles color property is set to red. The same can also be achieved with the following HTML attribute: . The benefit of using the styling namespace instead is that the styling can be made dynamic. It is possible to achieve conditional styling.

Styles can also be defined in the head section using the style element. The style element does not have any attributes.

The style element can be used to apply arbitrary CSS styles. View is placed in the DOM tree under a node with a class unique to the view type. The styles defined in the view are prefixed with that unique class name, so the styles defined in the view affect only the view that has defined it and its child views.

Even though a view has the body element in the document namespace, body element is not generated to the DOM tree. The root of the view is either HTML element or <g> SVG element. However, the root element can be styled with an element name *body* in the CSS. This is automatically converted to mean the root element generated for the view. This is shown in Example 14.

Styling with a style element

Example 13

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <style>
div {
color: red;
}
        </style>
    </head>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    <body xmlns:h="http://www.w3.org/1999/xhtml">
        <h:div>Foo</h:div>
    </body>
</document>
```

Defining a style of the root element of the view

Example 14

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <style>
body {
color: red;
}
        </style>
    </head>
    <body id="mybody"
        xmlns:h="http://www.w3.org/1999/xhtml">
Foo
    </body>
</document>
```

The global CSS and Less files are stored in the library's styles folder. These styles are loaded automatically and affect all views.

4.5 Mapping

Mapping functions are the type of user defined functions that can be used in expressions. For example, mapping functions are used to change the color of a text based on a value. In Example 15, the color of the value is changed based on the alarm state of the signal. The normal value is green, warning value is yellow, and alarming is red.

AZ value should be mapped to the corresponding color. This can be done with a mapping element. This example assumes that the measurement signal ESTHA1_MEC:P20 is existing.

Using mapping function for conditional styling

Example 15

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <mapping name="az-color">
      <step value="'red'" when="1" />
      <step value="'orange'" when="2" />
      <step value="'blue'" when="3" />
      <step value="'black'" when="4" />
      <otherwise value="'green'" />
    </mapping>
  </head>
  <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
    xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
    xmlns="http://www.w3.org/1999/xhtml">
    <details n:LN="ESTHA1_MEC">
      <summary>Interesting value</summary>
      <div>
        <span s:fontWeight="bold" s:color="{`az-
color`(`20.AZ`)}">{`20.OV`}</span>
        {`20.ST`} {`20.OX`} {`20.AZ`}
      </div>
    </details>
  </d:body>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

In Example 15, the AZ value is mapped to a color using a mapping called az-color. The value of each mapping is CSS color.

Mapping can have the *step* elements and one *otherwise* element as a child element. The value of the mapping function is the value of the first condition that has an expression in the *when* attribute that evaluates to *true*. If none of the expressions evaluates to *true*, the value is the value in the *otherwise* element. Mapping function can also have parameters. These are listed in the *parameters* attribute.

Mapping function can have parameters. Parameters can be used in both *value* and *when* attributes. Value of the parameter is provided when mapping function is invoked. Example 16 contains a view with one parameter, *value*. This parameter is used in both *value* and *when* attributes. Example contains a view with a mapping function that compares data point's value to high and low limits. The example uses a datapoint from one certain application and does not work with other applications if data point reference is not changed.

Mapping function with parameters

Example 16

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <mapping name="factorial" parameters="value">
      <step value="value * factorial(value - 1)" when="value > 1" />
      <otherwise value="1" />
    </mapping>
  </head>
  <body>Factorial of 5: {factorial(5)}</body>
</document>
```

Example Y contains a view with mapping function where datapoint value is compared to high and low limits.

Example 17

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <mapping name="status" parameters="dp, high, low">
      <step text="Unknown" when="resolveDataPointQualityValidity(dp) !=
0" />
      <step text="High" when="dp.value > high" />
      <step text="Low" when="dp.value < low" />
      <otherwise text="Normal" />
    </mapping>
  </head>
  <body>
    Status: {status(`urn:x-cda:Domain:1e2d9690-d7d5-4ca6-8acd-
b3738f3e7cb6.current_L1`, 0, 10)}
  </body>
</document>
```

4.6 Value list

The value list is a simpler form of a mapping function. Value lists can also be used for providing list of options to a select widget.

Example 18 shows how a value list can be used as a function. The example is a simple function for finding the English suffix for an ordinal number from 1 to 29.

Simple function for finding suffix for English ordinals

Example 18

© 2021 Hitachi Power Grids. All rights reserved.

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <value-list name="pos" min="1" max="29" default-text="th">
            <option text="st" />
            <option text="nd" />
            <option text="rd" />
            <option text="st" value="21" />
            <option text="nd" />
            <option text="rd" />
        </value-list>
    </head>
    <body xmlns:h="http://www.w3.org/1999/xhtml">
        <h:div>1{pos(1)} 23{pos(23)} 17{pos(17)} 29{pos(29)}</h:div>
    </body>
</document>
```

Example 19 shows how a value list can be used with a select widget. This example implements a simple month selector.

The select element is in the document namespace, not in the XHTML namespace.

Simple month selector

Example 19

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="var" default="9"/>
        <value-list name="weekdays" min="1" max="17" default-text="no-
day">
            <option text="Mon" />
            <option text="Tue" />
            <option text="Wed" />
            <option text="Thu" />
            <option text="Fri" />
            <option text="Sat" />
            <option text="Sun" />
        </value-list>
    </head>
    <body>
        <select id="s" variable="var" values="weekdays"></select>
    </body>
</document>
```

When a value list is executed, the option is selected based on the arguments value. An argument is assumed to be a whole number. If the option does not have an explicitly defined value, then the option's value is one greater than the previous value, or the minimum if the option is the first option. Table 9 lists the attributes of value list elements.

© 2021 Hitachi Power Grids. All rights reserved.

Table 9: Attributes of the value list elements

Attribute	Description	Remarks
name	Name of the list. Names of value lists and mapping functions are in the same namespace and must be unique.	Required
min	Minimum value of the argument. It must be a whole number.	Required, if the option does not have an explicit value.
max	Maximum value of the argument	Optional
default-expression	Expression must be used if the option is not found for a given argument. Mutually exclusive with the default-text. Only applicable when value list is used as a function.	Optional
default-text	Text value must be used if the option is not found for a given argument. Mutually exclusive with the default-expression. Only applicable when value list is used as a function.	Optional

Table 10 lists the attributes of the option element.

Table 10: Attributes of the option element

Attribute	Description	Remarks
value	When used as a function, this is a numeric value of the input argument. When used as a selection list, this is either numeric or string of the output value.	Optional
text	When used as a function, this is the result of the function. When used with a select widget, this is the label. Mutually exclusive with an expression.	Optional
expression	When used as a function, this is the result of the function. When used with a select widget, this is the label. Mutually exclusive with a text.	Optional
selected	Applicable only when used with a select widget. If attribute's value is true, then this option is selected in the select widget.	Optional

4.7 Expressions

In previous sections expressions are used to bind data to the attribute and text node values. This section describes the expression language.

There are two types of expressions.

- text expressions
- expression

The text expression is a text that may contain expression in curly brackets.

An expression in the curly brackets is evaluated and replaced with the string representation of the result.

Expressions operate all JavaScript data types. Table 11 lists the valid expressions. The term name and extended name are used in Table 11. A name can start with any English letter or underscore and can contain any English letter, underscore, or digit. An extended name can

© 2021 Hitachi Power Grids. All rights reserved.

contain any characters except backtick (`). Backtick is typed as accent grave character, Unicode character 0x60.

Table 11: Structural elements for the expression language

Type	Description	Examples
True	Boolean constant true	
False	Boolean constant false	
Null	Constant null	
Number	Any number, format is same as in JavaScript.	42 1.24 -23 1e-10
String literal	String surrounded with apostrophe '	'Lorem ipsum'
Array literal	'[' (<expression> (',' <expression>)*)? ']'	[1, "a", null, null] []
Object literal	'{' (<name> ':' <expression> (',' <name> ':' <expression>)*)? '}'	{ a: 1, b: [] } {}
Unary operator	'-' | 'not' <expression>	-a not true
Binary operator	<expression> <binary operator> <expression> Binary operators are listed in Table 12.	1 + 1
Parenthesis	'(' <expression> ')'	(1 + 1) * 2
Variable reference	<name> or '`' <extended-name> '`'	foo `foo-bar`
Function call	<name> '(' (<expression> (, <expression>)*)? ')'	foo(1, 2) bar()
Field reference	<expression> (('.' <name> | '`' <extended-name> '`') | '[' <expression> ']')	foo.bar a[3]

Language has a single reserved word: undefined. It cannot be used in any expression.

Table 12 lists the available operators. Expressions are evaluated from left to right. Precedency of operators is shown in the table. The smaller number donates higher precedency. For example, 1 + 2**3 equals to 1 + (2**3). Some operators are associative. For example, since plus (+) operator is associative, it is possible to write 3 – 2 – 1, which is calculated as ((3 – 2) – 1).

Table 12: Operators in the expression language

Operator	Precedence	Associativity	Description	Example
or	8	left to right	Translates to JavaScript's || operator	a or b
and	7	left to right	Translates to JavaScript's && operator	a and b
!=	6	none	Inequality. Works on all datatypes. For object and arrays, operator compares an identity. For other datatypes, it compares values.	
=	6	none	Equality. See !=.	
<=	5	none	Translates to JavaScript's <= operator	1 <= 2 (true) "b" <= "a" (false) "a" <= "b" (true) "a" <= 1 (false) 1 <= "a" (false)
>=	5	none	Translates to JavaScript's >= operator	
<	5	none	Translates to JavaScript's < operator	

Table continues on next page

© 2021 Hitachi Power Grids. All rights reserved.

Operator	Precedence	Associativity	Description	Example
>	5	none	Translates to JavaScript's > operator	
in	4	none	Checks if the right-hand argument is an array-like and it contains the left-hand argument. If the right-hand side is not an array, operator evaluates to false.	a in ["a", "b", "c"] (true) 1 in 2 (false)
+	3	left to right	Translates to JavaScript's + operator	
-	3	left to right	Translates to JavaScript's - operator	
&	3	none	Bit-wise AND operator. Translates to JavaScript's & operator.	7 & 4 (4) 9 & 4 (0)
|	3	none	Bit-wise OR operator. Translates to JavaScript's | operator	7 | 4 (4) 9 | 4 (13)
<<	3	none	Left-shift operator. Translates to JavaScript's << operator.	127 << 3 (1016) 4 << 4 (64)
>>	3	none	Right-shift operator. Translates to JavaScript's >> operator.	127 >> 3 (15) 4 >> 4 (0)
*	2	left to right	Translates to JavaScript's * operator	
/	2	left to right	Translates to JavaScript's / operator	
**	1	right to left	Exponentiation	2 ** 4 (16)

There are built-in functions and user-defined functions. Table 13 lists the built-in functions.

Table 13: Built-in functions

Function	Arity	Description	Example
empty	1	Returns true if the argument is null, an empty collection, or an empty string	empty('') => true empty(3) => false
count	1	Returns number of elements in a collection	count([1, 2]) => 2 count(false) => 0
reverse	1	Reverses an array. If the argument is not an array, null is returned.	reverse([1, 2]) => [2, 1]
swap12	1	swaps 1 and 2	swap(1) => 2 swap('foo') => 'foo'
swap01	1	swaps 0 and 1	swap(1) => 0 swap('bar') => 'bar'
format	Variadic	Formats a string using sprintf-style syntax. Modifiers %s, %u, %d, %x, %X, and %f are supported.	format("%n apples", 3) => '3 apples'
scale	5	Does linear scaling of the first argument. The second and third arguments define the input range, and the fourth and fifth arguments define the output range.	scale(1, 1, 3, 10, 30) => 10
if	3	If the first argument is true, returns the second one. If not, then returns the third one.	if(true, 1, 2) => 1 if(false, 3) => undefined
min	Variadic	Returns the smallest argument	min(1, 2, 3) => 1
max	Variadic	Returns the largest argument	max(1, 2, 3) => 3
gradientMapping	6	Maps a value from the given range to a gradient color. The first argument is the input value. The second and third arguments are starting color and ending color. The fourth and fifth values are the input range. The sixth argument is the number of steps in the gradient.	gradientMapping(3, 'red', 'green', 1, 5, 10)

Table continues on next page

© 2021 Hitachi Power Grids. All rights reserved.

Function	Arity	Description	Example
resolveSubscriptionConnectionStatus	1	Returns status about a datapoint. 0 if the datapoint is valid, 1 if the subscription is not succeeded, 2 if the value is missing, and 3 if the datapoint is not mapped.	
resolveSubscriptionValue	2	Returns the first argument if the argument is a datapoint and the subscription has succeeded. If not, returns the second value.	
isConnectionLost	1	Returns false if the parameter is a datapoint and has a valid value, for example, the subscription has succeeded.	
resolveDataPointQuality	2	Extracts information from the datapoint's quality field. The first argument is the field to be extracted and second argument is the quality field of a datapoint. Possible values for the first argument are overflow, outOfRange, badReference, oscillatory, failure, oldData, inconsistent, inaccurate, source, test, blocked, derived, and reserved.	resolveDataPointQuality('blocked', swPos.quality)
resolveDataPointQualityValidity	1	Extracts the validity information from the datapoint's quality field. Possible return values are 0 (Good), 1 (Invalid), 2 (Undefined validity), and 3 (Questionable).	resolveDataPointQualityValidity(swPos.quality)
resolveDataPointSubtitutedBy		Extracts the substitution information from the datapoint's quality field. Possible return values are 0 (Undefined), 1 (Process bus IED), 2 (Station bus IED), and 3 (Station IED).	resolveDataPointSubstitutedBy(swPos.quality)
unmappedValue	0	Returns a symbol that denotes an unmapped signal. It can be used in conjunction with resolveSubscriptionConnectionStatus.	
date	2	Formats a date. The first argument is the date format and second is the date to be formatted. The date can be either a JavaScript date object or an object representing 64-bit integer as nanoseconds since 1 January 1970, UTC.	
shortTime	1	Formats the given timestamp as short representation of time, but not date. The format contains hours, minutes, and seconds.	
longtime	1	Formats the given timestamp as long representation of time, but not date. The format contains hours, minutes, seconds, and milliseconds.	
shortDate	1	Formats the given timestamp as short representation of date, but not time. This format can be used, for example, in tables.	
longDate	1	Formats the given timestamp as long representation of date, but not time. This format can be used, for example, in textual context.	
shortDateTime	1	Concatenation of shortDate and shortTime	
longDateTime	1	Concatenation of shortDate and longTime	
number	1 to 3	Formats a number according to the current locale. The first argument is the number to be formatted, second is the number of fraction digits, and third is the digits to which the string is padded to.	number(1.2323, 3) => 1.23 number(1.232, 1, 2) =>1.2_ Underscore represent a figure space, that is, a space that has the same length as one digit when tabularly aligned numbers are used.

© 2021 Hitachi Power Grids. All rights reserved.

It is possible to define new built-in functions to the language. The JavaScript function in Example 20 is used to add a factorial function. After executing this JavaScript function, a new function can be used in an expression.

The built-in functions are global; a new function is available for all expressions in all views. It is advisable to declare such function in the JavaScript code in any views, but in a separate global JavaScript files (see Section 8).

Creating new built-in function using JavaScript

Example 20

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <script type="client">

runtime.getBuiltinFunctions().add("factorial", function(a) {
    let res = 1;
    while (a > 1) {
        res *= a;
        a--;
    }
    return res;
});

        </script>
    </head>
    <body xmlns:h="http://www.w3.org/1999/xhtml">

{factorial(5)}
    </body>
</document>
```

4.8 Result expressions

Result expressions define how the results of various operations are handled. Result expressions are like expressions, but each immediate value is replaced with a variable name.

Table 14: Example result expressions

Expression	Result	Remarks
foo	2	2 is stored to the variable foo.
[foo, bar]	[“a”, “b”, “c”]	In this example, a is stored to the variable foo, b is stored to the variable bar, and c is discarded.
foo	[“a”, “b”, “c”]	An array containing a, b, and c is stored to the variable foo.
[foo, bar]	2	An error is raised, as the result cannot be stored with a given result expression.
{a: foo, b: bar}	{a: “Hello world”}	The string *Hello world* is stored to the variable foo. The variable b is set to undefined.
{a: `foo-bar`}	{a: 1}	1 is stored to the variable foo-bar.
[{a: foo}, {a: bar}]	[{a: 1}, {}]	1 is stored to variable foo. Variable bar is set to undefined.

Result expressions are used, for example, in server scripts and service invocations. Example 21 shows how a result expression can be used with a SCIL script.

Example of using a result expression with a SCIL script

Example 21

© 2021 Hitachi Power Grids. All rights reserved.

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="SS" />
        <variable type="internal" name="SP" />
        <script type="server" name="my-script" authorize="GENERAL >= 0"
result="{SS: SS, SP: SP}" init="true">
#return LIST(SS=APL:BSS, SP=APL:BSP)
        </script>
        <script type="client">
    self.invoke("my-script", {} ,
    response => {
    alert(JSON.stringify(response));
    });
        </script>

    </head>
    <body>
    </body>
</document>
```

4.9 Variables

Views can have variables like the variables used in most of the programming languages. There are five types of variables: internal, external, session, window, and server.

Internal variables are internal to the view; views use them to store an internal state. Internal variables are not hidden from other views; so, by using JavaScript, internal variables of another view can be accessed.

External variables are arguments to the view. When a view is opened, external variables must be given values, unless a variable has a default value. The variable values are given based on the way how a view is opened.

- If a view is opened directly with an URL, variable values are provided as URL query parameters.
- If a view is opened using the view element, variables are defined with parameter elements.
- If a view is opened using a JavaScript function, variables are given as arguments to that JavaScript call.

Session variables are like internal variables, but its values can be persisted for the duration of a page session. The data does not disappear when the page is refreshed but disappears when the page is closed.

Window variables are variables that are shared among all views opened in the same window.

Server variables are variables that are subscribed from the server. If the data that needs to be subscribed is used in expressions, it is automatically subscribed from the server when the view is opened. Also, if the data is used in JavaScript, it is automatically subscribed. However, if the data needs to be subscribed, it is available only for asynchronous calls. If a server variable is defined, then the scripts in the views are executed after the value is received, so the value can be used directly in all scripts.

Variables are defined using the variable tag. Each variable has a name and can have a default value. Example 22 shows different variable types. Table 15 lists the attributes of the variable tag.

© 2021 Hitachi Power Grids. All rights reserved.

Table 15: Attributes of the variable tag

Attribute	Description	Remarks
name	Name of the variable. It can be any string, but it might be easier to use if the naming convention of simple expression names are used. See Section 4.7.	Required
type	Possible values are internal, external, session, window, and server.	Required
datatype	If datatype is defined, then value of the variable is converted to the given datatype. Possible values are string, number, and boolean.	Optional
defaultValue	Default value of the variable. If the default value is given to an external variable, the parameter becomes optional.	Optional

Different variable types

Example 22

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <variable type="internal" name="foo" default="1" />
    <variable type="external" name="bar" default="''" />
    <variable type="server" name="urn:x-
cda:Domain:AA1.C1.Q01.QB1.switch_position" />
    <script type="client">
      alert(self.get("urn:x-
cda:Domain:AA1.C1.Q01.QB1.switch_position").toString());
    </script>
  </head>
  <body>
    ...
  </body>
</document>
```

Variables can have actions. The actions are executed when a value of the variable is changed. For more details about actions, see Section 5.4. A view with an action in a variable is shown in Example 23. In the example, when the value of variable **a** is changed, variable **b** is assigned the value that is **a+1**.

Action within a variable

Example 23

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="a">
            <action type="set" name="b" value="a + 1" />
        </variable>
        <variable type="internal" name="b" />
    </head>
    <body>
        <slider variable="a" min="0" max="10" />
 A: {a} B: {b}
    </body>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

4.10 Role

Role attributes define the role of a view in the system. The role is not related to the access control. It is possible to query views that have a certain role. View can have many roles. Role is defined as in Example 24.

A view with a role

Example 24

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="process" path="Rivers" priority="10" />
        <role name="my-role" />
    </head>
    <body>
...
    </body>
</document>
```

The role has one mandatory attribute: name. The attributes path and priority are optional. It is interpreted based on the usage of a particular role. Table 16 lists the roles that are used in SYS600.

Path can contain any characters. The forward slash character (/) is interpreted as a path separator. For example, when a menu structure is built based on the path, elements separated by a forward slash denotes different menu levels. For example, if the path is **Transformers/ Eastwick/Primary** and the role is used for constructing a menu structure, the view navigates to the **Transformers** menu and under that to the **Eastwick** menu.

Table 16: Known roles

Role	Description
process	View included in the **Pictures** menu in the menu tree. If path is defined, then that is used for placing the view in the menu structure. If path is not defined, then the folder structure is used. Priority defines the order of views to be placed in the tree.
supervision	View included in the **System Overview** menu. If path is defined, then that is used for placing the view in the menu structure. If path is not defined, then the folder structure is used. Priority defines the order of views to be placed in the tree.
main-menu-item	Denotes an item in the main menu. For more information on how to add items to the menu, see Section 11.1.
sld	Single line diagrams used by MonitorPro+. These SLDs cannot be used with WebMonitor or web browser.

It is possible to use custom roles. Example 25 shows how query views can be invoked using JavaScript. The example displays a message box with a JSON structure containing all views with the process role. If the used application does not have any process pictures, the result will be empty.

Invoking query views with JavaScript

Example 25

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <service interface="WebServer" method="queryViews" />
        <script type="client">
self.invokeService("WebServer", "queryViews", {role: "process"},
```


© 2021 Hitachi Power Grids. All rights reserved.

```
response => {
alert(JSON.stringify(response));
});
        </script>
    </head>
    <body>

    </body>
</document>
```

4.11 Metadata

It is possible to attach metadata to views. This data is then accessible using JavaScript and can be used for implementing various application logics. Example 26 shows a trivial view with a metadatum.

View with a single metadatum

Example 26

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <meta name="message" content="Lorem ipsum" />
    </head>
    <body>
This my Example24 view
    </body>
</document>
```

Metadata can be used by other views. This is done with JavaScript. If a view in Example 26 is stored as view1.xml, then the view in Example 27 shows a dialog box with a message Lorem ipsum.

View using a metadatum from another view

Example 27

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <meta name="description" content="Description of view
Example25" />
        <script type="client">
alert(view.getType().metadata.get("description"));
runtime.showDialog(self, "Example24", {}, self.getRootNode(), {
onAfterShow: view => {
alert(view.getType().metadata.get("message"));
}
});
        </script>
    </head>
    <body>
This my Example25 view
    </body>
</document>
```

Some system views use metadata. For more information on metadata used by these views, see Section 11.4.

© 2021 Hitachi Power Grids. All rights reserved.

4.12 Managing large views

Views can be large. A complex view can have thousands of lines of code. Typically, largest sections are JavaScript blocks and CSS blocks. Writing long blocks of JavaScript within an XML document might not be convenient because, typically, editors do not understand the context and the features, such as syntax checking, and auto-completion are not available.

It is possible to split a view to separate files so that scripts are in a separate file and styles are in a separate file. The server then combines the view to a single view file.

Normally, a view is a single file, for example, \sc\apl\<myapplication>\views\dialogs \myview.xml folder. A view can also be a directory instead of a file. The directory shall contain an XML file with the same name as the directory and a JSON file named as view.json. Then, the directory can contain arbitrary number of JavaScript, TypeScript, CSS, or Less files. TypeScript files are automatically compiled to JavaScript and Less files are automatically compiled to CSS. The view.json file shall contain empty JSON object {}. Currently, there are no fields in the object.

4.13 Embedding views

It is possible to include one view in another. To make this useful, parameters to the views must be defined and set. Consider the view in Example 28. The view shows information only from one bay. This view can be reused but the reference to the bay is hard-coded.

View for visualizing the measurement

Example 28

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <style>
summary {
color: #0076b7;
padding: 5px;
padding-bottom: 15px;
font-size: large;
}
details {
margin: 4px;
border: 1px solid #d2d2d2;
padding: 5px;
}
details>div{
padding: 5px;
}
span.measurement {
display: inline-block;
width: 4em;
font-weight: bold;
text-align: end;
margin-right: 0.5em;
}
        </style>
        <mapping name="az-color">
            <step value="'red'" when="1" />
            <step value="'red'" when="2" />
            <step value="'#fc0'" when="3" />
            <step value="'#fc0'" when="4" />
            <otherwise value="'green'" />
        </mapping>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <details n:LN="ESTHA1_MEC">
            <summary>{`20.IL`.STA} {`20.IL`.BAY}</summary>
            <div>
                <span class="measurement" s:color="{`az-
color`(`20.AZ`)}">{number(`10.OV`, 3)}</span> {`10.ST`} {`10.OX`}</div>
            <div>
                <span class="measurement" s:color="{`az-
color`(`20.AZ`)}">{number(`20.OV`, 3)}</span> {`20.ST`} {`20.OX`}</div>
            <div>
                <span class="measurement" s:color="{`az-
color`(`21.AZ`)}">{number(`21.OV`, 3)}</span> {`21.ST`} {`21.OX`}</div>
        </details>
    </d:body>
</document>
```

The name of the bay can be used as a parameter. Parameters are defined in the view's head section using the variable tag. The parameter definition must be added for the LN and the n:LN attribute must bind to the parameter. This is shown in Example 29.

Modification to the view with LN as parameter

Example 29

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <style>
summary {
color: #0076b7;
padding: 5px;
padding-bottom: 15px;
font-size: large;
}
details {
margin: 4px;
border: 1px solid #d2d2d2;
padding: 5px;
}
details>div{
padding: 5px;
}
span.measurement {
display: inline-block;
width: 4em;
font-weight: bold;
text-align: end;
margin-right: 0.5em;
}
        </style>
        <variable type="external" name="LN" />
        <mapping name="az-color">
            <step value="'red'" when="1" />
            <step value="'red'" when="2" />
            <step value="'#fc0'" when="3" />
            <step value="'#fc0'" when="4" />
            <otherwise value="'green'" />
        </mapping>
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
```


© 2021 Hitachi Power Grids. All rights reserved.

```
        <details n:LN="{LN}">
            <summary>{`20.IL`.STA} {`20.IL`.BAY}</summary>
            <div>
                <span class="measurement" s:color="{`az-
color`(`20.AZ`)}">{number(`10.OV`, 3)}</span> {`10.ST`}
{`10.OX`}</div>
            <div>
                <span class="measurement" s:color="{`az-
color`(`20.AZ`)}">{number(`20.OV`, 3)}</span> {`20.ST`}
{`20.OX`}</div>
            <div>
                <span class="measurement" s:color="{`az-
color`(`21.AZ`)}">{number(`21.OV`, 3)}</span> {`21.ST`}
{`21.OX`}</div>
        </details>
    </d:body>
</document>
```

If the view is opened in the browser without passing the LN parameter, an error about missing parameter is displayed. To avoid the error, the view must be opened with an URL that contains the LN parameter.

http://<computer name>/view/Example27?LN="ESTHA1_MEC"

The information about ESTHA1_MEC is displayed.

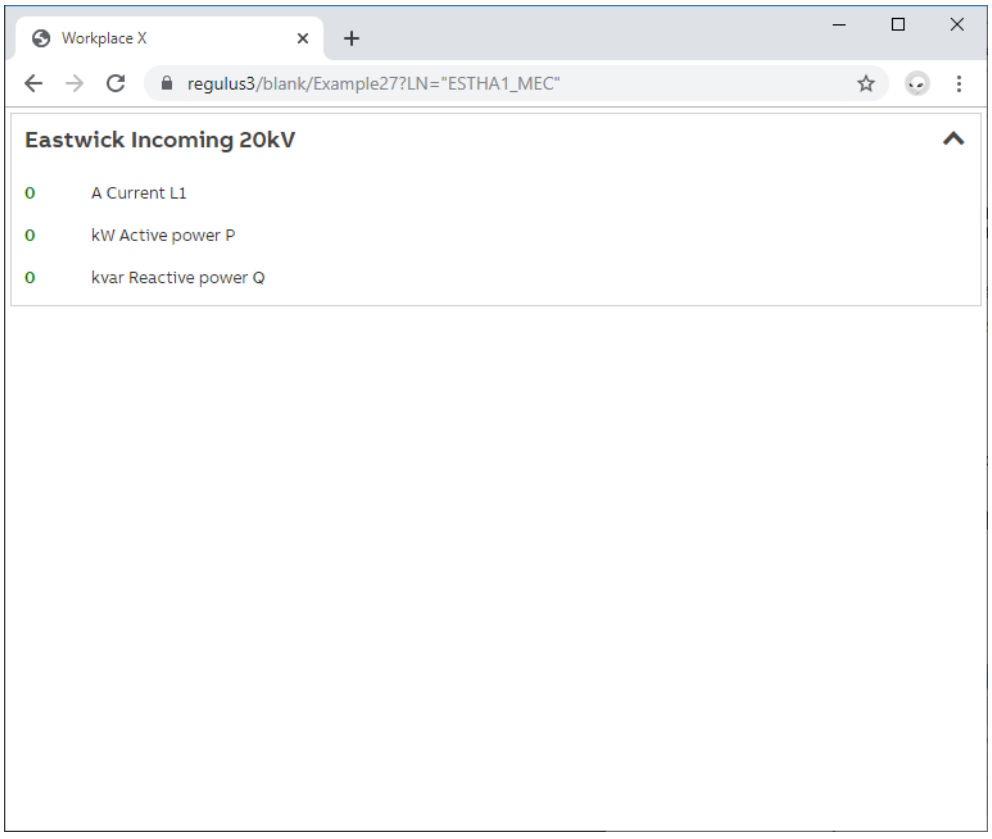


Figure 19: Information about ESTHA1_MEC

Next, another view is created that embeds information about different devices.

It is possible to use Example 30 using the view in the Example 29.

Example 30 using the view in the Example 29

Example 30

© 2021 Hitachi Power Grids. All rights reserved.

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <d:view name="Example27">
            <d:parameter name="LN" value="'ESTHA1_MEC'" />
        </d:view>
    </d:body>
</document>
```

Open http://<computer name>/view/Example28.

The information about ESTHA1_MEC is displayed again. When the d:view is multiplied and the parameter is changed accordingly, a display that shows the whole station can be created. Also, other views can be added. Example 31 is a view, which displays data from different measurements in the left side and an alarm list in the right side for Eastwick (see Figure 20).

View with different measurements and alarm values

Example 31

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml" s:display="block">
        <d:view name="Example27">
            <d:parameter name="LN" value="'ESTHA1_MEC'" />
        </d:view>
        <d:view name="Example27">
            <d:parameter name="LN" value="'ESTHA2_MEC'" />
        </d:view>
        <d:view name="Example27">
            <d:parameter name="LN" value="'ESTHA3_MEC'" />
        </d:view>
        <d:view name="Example27">
            <d:parameter name="LN" value="'ESTHA4_MEC'" />
        </d:view>
    </d:body>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

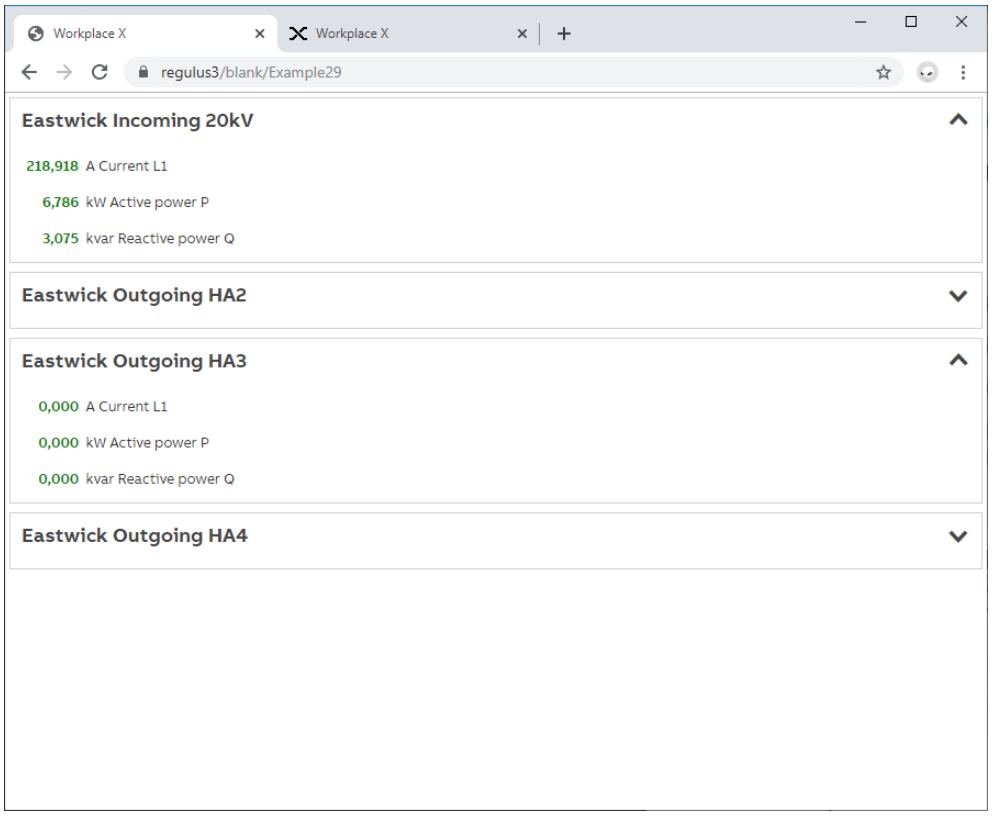


Figure 20: The browser window

4.14 Authorization

User authorization information can be used in views. Authorization is based on user rights. User rights are defined in separate XML files.

User rights are defined in the XML files in the WebUI libraries. Files are located in the folder called rights. Example of a right file is show in Example 32. The example defines two rights, **PROCESS** and **CHANGE_PASSWORD**. The convention is to write right names in capital letters and use underscore instead of whitespace. Both rights in the example have a single condition. A right can have multiple conditions and is granted if all the conditions are satisfied. **PROCESS** rights condition is based on a product license field and **CHANGE_PASSWORD** right's condition is based on SYS600 authorization group.

A simple right definition

Example 32

```
<?xml version="1.0"?>
<rights>
  <right name="PROCESS">
    <condition expression="SA_LIB" type="license" />
  </right>
  <right name="CHANGE_PASSWORD">
    <condition expression="PASSWORD_HANDLING > 0" type="authorize" />
  </right>
</rights>
```

User right can be used in view files. Example 33 shows a view that contains the right expression. The text *"Change password"* is only shown if the user opens the view which has **CHANGE_PASSWORD** right.

© 2021 Hitachi Power Grids. All rights reserved.

Using authorization information in a view

Example 33

```
<d:if condition="access(`right:CHANGE_PASSWORD`)">
   <span>Change password</span>
   ...
</d:if>
```

The above example subscribes data **right:CHANGE_PASSWORD** from the server. The function called access is then used to convert the data type of the subscribed data to a Boolean value.

List of built-in user rights is available in the developer documentation.

© 2021 Hitachi Power Grids. All rights reserved.

Section 5 Content

Views can contain both HTML and SVG content. The content can be combined as defined by the HTML and SVG specifications. The content is defined in the body element. The used body element is part of the document namespace, not the HTML namespace. Table 17 lists the attributes of the body element.

Table 17: Attributes of the body element

Attribute	Description	Remarks
oninit	JavaScript snippet that is executed when the view is opened	Optional
onresize	JavaScript snippet that is executed when the window is resized	Optional
x1, x2, y1, y2	Initial zoom viewport. See Section 5.2.	Optional. Either all must be present or all must be absent.

5.1 HTML content

Most views, such as control dialog boxes, lists, and so on, are build using HTML content. Views can contain any HTML elements. When the view is instantiated, the DOM tree of the view is built out of the HTML elements.

HTML elements can be combined with other features of the framework. Styles can be set using the styling namespace. For more information, see Section 4.4.

In a view file, it is possible to use any HTML element. In some cases, there is an element with the same name in both HTML and document namespaces. For example, *button* can be an HTML element or a built-in button widget. Button in the document namespace has some additional properties. Therefore, it is important to know which namespace does elements belong to.

There are several auxiliary attributes and elements in the document namespace that can be used to help writing an HTML content. Some of the attributes are as follows.

Binding

Binding can be used to bind an HTML input element to a variable. Example 34 shows a view that uses the binding attribute. When the value of the input field is changed, the change is automatically propagated to the variable.

Using the binding attribute

Example 34

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable name="a" type="internal" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <p>A: {a}</p>
        <input d:binding="a" />
    </d:body>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

Hidden

This attribute hides the element. It works similarly with both SVG and HTML elements. Since the attribute receives a boolean value as a parameter, it is easier to use it with expressions than the display or visibility styles. Example 35 shows how the hidden attribute can be used.

Using the hidden attribute

Example 35

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable name="a" type="internal" default="false" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <d:switch variable="a" />
        <p d:hidden="a">Hello</p>
    </d:body>
</document>
```

Class

Set the class of the element similar to the HTML class attribute. The benefit of this attribute is that it can also be used with widgets.

Id

Set the id of the element similar to the HTML id attribute. The benefit of this attribute is that it can also be used with widgets.

Object-name

Metadata that can be attached to a widget.

5.2 SVG content

SVG content can be used for single line diagrams and other graphical representations. SVG content works similar to HTML content. For example, styling attributes can be used. There are some auxiliary attributes in the document namespace that can be used with SVG content.

Tooltip

This adds a tooltip to an SVG element.

Hidden

This hides the element. It works similarly with both SVG and HTML elements. Since the attribute receives a boolean value as a parameter, it is easier to use with expressions than the display or visibility styles.

Class

Set the class of the element similar to the HTML class attribute. The benefit of this attribute is that it can also be used with widgets.

Id

Set the id of the element similar to the HTML id attribute. The benefit of this attribute is that it can also be used with widgets.

© 2021 Hitachi Power Grids. All rights reserved.

SVG content can be zoomed and panned. Table 18 lists the attributes that are used to control the zooming functionality. Zooming is available only if an SVG view is opened either as the main view or to a view widget inside an HTML view. If SVG is opened inside another SVG, zooming related attributes are ignored.

Table 18: Zooming related attributes

Element	Attribute	Remarks
document	zoomable	Defines if the view can be zoomed or not. Default is false. For more information about zooming, see Section 5.2.1.
document	zoomScaleFactor	Defines the zoomable area of the view. Default value is 1. With the default value accessible, SVG coordinates are roughly from -5000 to +5000 coordinate points. The value range is multiplied with the zoomScaleFactor. For more information about zooming, see Section 5.2.1.
document	zoomLevelFactor	Attribute controls how the zoom levels are interpreted in the decluttering algorithm. For more details, see Section 5.2.1 and Section 5.2.2.
document	background	Color that is assigned to the parent element of the view if the parent element is an HTML element.
body	x1	Left coordinate of the initial viewport
body	y1	Top coordinate of the initial viewport
body	x2	Right coordinate of the initial viewport
body	y2	Bottom coordinate of the initial viewport

5.2.1 Zooming

The *zoomScaleFactor* attribute (in the main XML document element) defines the zoomable SVG view area. The default value is 1. With the default value, accessible SVG coordinates are from -5000 to +5000 coordinate points. The value range is multiplied by the given *zoomScaleFactor* value. The *zoomScaleFactor* also affects the range of the zoom that can work. The smallest and largest scales for the zoom are defined as follows.

- The smallest scale: 0.5 / zoomScaleFactor
- The largest scale: 250 / zoomScaleFactor

For example, with the default factor 1, the view can be zoomed out to be 0.5 scale and zoomed in to be 250 times of the default scale.

View that uses zoom related attributes and elements

Example 36

```
<?xml version="1.0" encoding="utf-8"?>
<d:document nature="http://www.w3.org/2000/svg"
    xmlns:d="http://www.webui-ap.com/schemata/view/1.0" zoomable="true"
zoomScaleFactor="25" zoomLevelFactor="0.010">
    <d:head>
        <d:role name="process" />
    </d:head>
    <d:body xmlns:s="http://www.w3.org/2000/svg">
        <d:layer visibility="10">
            <s:circle cy="0" cx="0" r="250" stroke="#f00" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="9">
            <s:circle cy="0" cx="0" r="500" stroke="#0f0" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="8">
```


© 2021 Hitachi Power Grids. All rights reserved.

```
            <s:circle cy="0" cx="0" r="750" stroke="#00f" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="7">
            <s:circle cy="0" cx="0" r="1000" stroke="#ff0" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="6">
            <s:circle cy="0" cx="0" r="1250" stroke="#f0f" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="5">
            <s:circle cy="0" cx="0" r="1500" stroke="#ff0" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="4">
            <s:circle cy="0" cx="0" r="1750" stroke="#f88" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="3">
            <s:circle cy="0" cx="0" r="2000" stroke="#8f8" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="2">
            <s:circle cy="0" cx="0" r="2250" stroke="#88f" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="1">
            <s:circle cy="0" cx="0" r="2500" stroke="#888" stroke-
width="25" fill="none" />
        </d:layer>
        <d:layer visibility="0">
            <s:circle cy="0" cx="0" r="2750" stroke="red" stroke-
width="25" fill="none" />
        </d:layer>
    </d:body>
</d:document>
```

Example 36 shows a view that uses various zoom related attributes and elements. It has different colored circles that appears and disappears based on the zoom level. The innermost red circle is visible only if the user has zoomed in and the largest black circle is always visible.

5.2.2 Decluttering

It is possible to control which elements in the SVG are to be displayed or hidden according to the zoom level. This is done by placing elements inside the layer element by adjusting zoom using the zoomLevelFactor attribute.

The zoom layers are for making view objects visible or invisible based on the current zoom state. The layers are defined with the layer elements. Each layer element must have an attribute visibility for defining the zoom layer as an integer number starting from zero. The layers help in decluttering views by displaying fewer details on the lower layers (that is, small zoom scale, when a full view is visible) and more details when zooming into the higher layers.

When defining the layer elements in an XML view, following rules can be followed:

- Layer 0: child elements are always visible
- Layer 5: child elements are visible when zoomed in to the semi-detailed level
- Layer 10: child elements are visible only when zoomed in to the most detailed level

The zoomLevelFactor attribute (in the main document element) defines the factor for moving from a zoom layer to the next one. The default value for the zoomLevelFactor attribute is the same as the smallest zoom scale, that is, 0.5 / zoomScaleFactor. The current active zoom layer is defined by the formula: currentZoomScale / zoomLevelFactor. This means that the zoom

© 2021 Hitachi Power Grids. All rights reserved.

layer 0 is active when the view is zoomed to its minimum scale, and the layer 1 is activated when the current zoom scale is larger than the zoomLevelFactor (or the minimum scale, if the default value is used).

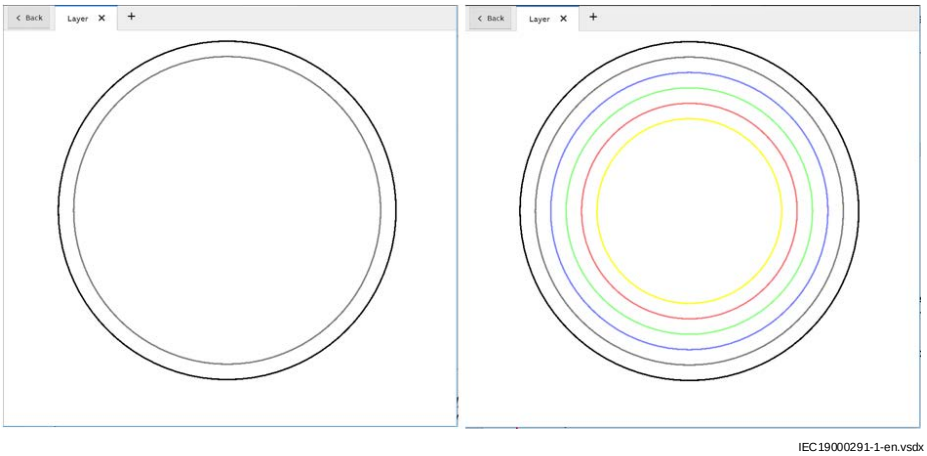


Figure 21: Effect of the zoomLevelFactor

Figure 21 shows a view from Figure . Left side has the zoomLevelFactor defined to 1 and right side to 0.02, with both the views in a default zoom level. This means that the user has just opened the view and has zoomed in or out.

The main use case for zoomLevelFactor is for enabling the same zooming user experience for sub-views as for a main top-level view. This can be done by setting the same zoomLevelFactor value for each view in the view hierarchy. This way only the zoom layer 0 is active on the top-level view (for example, station view), and sub-views (for example, bay views) inside it follow the same layers when opened separately as when zooming in from the top-level view.

5.3 Name binding

Views can reference data in the server. This is done with the curly bracket notation. Example 37 shows a trivial view that shows one data item.

Simple view showing a value from the server

Example 37

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <body>
{`urn:x-cda:Domain:AA2.C1.Q05.QZ1.QA1.switch_position`.value}
    </body>
</document>
```

This example is very static. It only shows the switch position of a certain device. It is possible to bind a name partially. This allows same view to subscribe different items. Example 38 shows how a partial name binding can be used. Name binding allows name to be resolved using expression. In this example, it is possible to give a part of the name as parameter to the view.

View using name binding

Example 38

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:n="http://www.webui-ap.com/schemata/naming/1.0">
    <head>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
        <variable name="ref" type="external"
default="'AA2.C1.Q05.QZ1.QA1'" />
    </head>
    <body n:cda-topic="Domain" n:cda-name="AA2.C1.Q05.QZ1.QA1">
{`cda:switch_position`.value}
    </body>
</document>
```

There are two namespaces that can be used: cda and sys. CDA stands for common data address. This is the naming schema shared by various products. SYS is a namespace specific to SYS600.

CDA name has three parts: topic, object name, and attribute. The structure of the CDA name is shown in Figure 22.

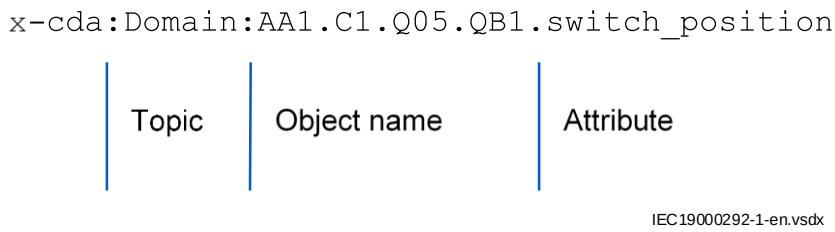


Figure 22: Structure of a CDA name

SYS namespace allows access to the SYS600's OPC namespace. Table 19 lists the attributes of the SYS namespace. Example 39 shows how the SYS namespace can be used. This example is a view that shows the OV attribute of a given signal identified with LN and IX.

To open Example 39, use URL https://%3Ccomputer%20name%3E/blank/Example35?LN=%22ESTH01_MEC%22&IX=%2210%22

Table 19: Attributes of the SYS namespace

Attributes	Remarks
LN	Logical name of an application object
IX	Index of an application object
type	Type of the object
apl	Application number

Using SYS namespace

Example 39

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:n="http://www.webui-ap.com/schemata/naming/1.0">
    <head>
        <variable name="LN" type="external" />
        <variable name="IX" type="external" />
    </head>
    <body n:LN ="{LN}" n:IX="{IX}">
{`OV`}
    </body>
</document>
```

It is also possible to subscribe items with an absolute OPC name. It is, however, advisable to use the form that does not include the application number. So, instead of subscribing to \\APL\1\P\ESTH01_Q0\10:OV, the format \P\ESTH01_Q0\10:OV should be used. Former

© 2021 Hitachi Power Grids. All rights reserved.

syntax should only be used when referring to items that are not in the current application. Example 40 shows a subscription with both formats.

Subscribing data using SCIL syntax

Example 40

```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
    <d:head>
        <d:title>SCIL Items</d:title>
    </d:head>
    <d:body y:padding="2em"
        xmlns:h="http://www.w3.org/1999/xhtml">
OX: {`\\APL\1\P\ESTH01_Q0\10:OX`}         <!-- use when referring to
other APLs -->
OV: {`\P\ESTH01_Q0\10:OV`}         <!-- preferable syntax -->
    </d:body>
</d:document>
```

5.4 Actions

Actions are used to attach behavior elements in the page. Table 20 lists the attributes that are common to all actions.

Table 20: Attributes common to all actions

Attribute	Description	Remarks
type	Defines the type of the action. Other attributes are depending on the type of the action. All built-in action types and its attributes are listed in this section. It is also possible to add more custom actions.	Required
event	Defines when the action is executed. The default value is of context specific. If the action is inside a DOM element or a component, the default value is click.	Optional
when	Defines a condition on whether the action should be executed	Optional

Back

If a view is opened using the view element, for example, in a tab, then the action returns in the history buffer of the element. Otherwise, the action returns in the window's back buffer. An action with type back does not accept any additional parameters.

Close

This action closes the current view. If a view is opened as a dialog box, the view can return a value. This value can be defined using the value attribute. Example 41 shows how the close action can be used as from a popup. The example contains two views: main and popup. When the button in the main is clicked, a popup displays requesting the user to enter a message. When the user types the message, a notification with the same message is displayed. Table 21 lists the additional attributes of the close action.

Using open-modal and close actions

Example 41

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    <head>
        <variable type="internal" name="msg" />
        <role name="process" />
    </head>
    <body>
        <button>Click me!
            <action event="click" type="open-modal-dialog"
name="Example38" result="msg">
                <action event="close" type="open-popover"
name="Example38" />
            </action>
        </button>
    </body>
</document>
```

Popover view

Example 42

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="value" default="'Hello!'" />
    </head>
    <body>
Enter message: <edit variable="value" />
        <button>Close
            <action event="click" type="close" value="value" />
        </button>
    </body>
</document>
```

Table 21: Attributes of the close action

Attribute	Description	Remarks
value	Return value of the view	Optional

Commit

This action publishes changes to a variable. Each view has a copy of the variable value. When a variable is committed, the change is propagated to the parent view. Table 22 lists the attributes of the commit action.

Table 22: Attributes of the commit action

Attribute	Description	Remarks
name	Name of the variable to be committed	Required

Dec

This action decrements a value of the variable. Example 43 shows how the inc and dec actions can be used to increment and decrement a value of the variable. Table 23 lists the additional attributes of the dec action.

View using inc and dec actions

Example 43

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="value" default="10" />
    </head>
    <body>
Value: {value}
```


© 2021 Hitachi Power Grids. All rights reserved.

```
        <button>+
            <action event="click" type="inc" name="value"/>
        </button>
        <button>-
            <action event="click" type="dec" name="value"/>
        </button>
    </body>
</document>
```

Table 23: Attributes of the dec action

Attribute	Description	Remarks
name	Name of the variable that is decreased	Required
value	The amount of which the variable is decreased. Default value is 1.	Optional

Delete-overlay

This action deletes a named overlay. For more details about overlays, see Section 5.6.

Table 24: Attributes of the delete-overlay action

Attribute	Description	Remarks
name	Name of the overlay	Required

Delete-all-overlays

This action deletes all overlays. For more details about overlays, see Section 5.6. This action does not accept any attributes.

Erase-all-highlights

This action removes all highlights. See Highlight. This action does not accept any attributes.

Focus

This action focuses on an element given as a parameter. The id of the element is given as parameter id. Example 44 shows how the focus action can be used.

Using the focus action

Example 44

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <body xmlns:h="http://www.w3.org/1999/xhtml">
        <h:input id="foo" />
        <button>Click me!
            <action event="click" type="focus" id="foo" />
        </button>
    </body>
</document>
```

Highlight

This action highlights an SVG element. Table 25 lists the attributes of the highlight action.

© 2021 Hitachi Power Grids. All rights reserved.

Table 25: Attributes of the highlight action

Attribute	Description	
id	An id of the element to be highlighted. If not given, then the component owning the action element is highlighted.	Optional
color	Color to be used for highlighting	Optional. Default is yellow.

Using the highlight action

Example 45

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0"
nature="http://www.w3.org/2000/svg">
    <body xmlns:s="http://www.w3.org/2000/svg">
        <s:circle id="foo" cx="100" cy="100" r="50" fill="red" />
        <s:rect x="200" y="200" width="300" height="300" fill="black">
            <action event="click" type="highlight" color="green" />
            <action event="click" type="highlight" color="blue"
id="foo" />
        </s:rect>
    </body>
</document>
```

Inc

This action increments a value of the variable. Example 43 shows how the inc and dec actions can be used to increment and decrement a value of the variable. Table 26 lists the attributes of the inc action.

Table 26: Attributes of the inc action

Attribute	Description	
name	Name of the variable that is decreased	Required
value	The amount of which the variable is decreased. Default value is 1.	Optional

Invoke

This action invokes an SCIL script. For more information, see Section 4.3.

Invoke-service

This action invokes a service. For more information, see Section 10.

Js-invoke

This action invokes a JavaScript function. The function must be either in the view's user context or in a component. Table 27 lists the attributes of the js-invoke action. The arguments to the method can be given using the parameter element. Parameters do not have names. Example 46 shows how a method can be called from the user context. The method needs to be assigned to the user context object. Example 47 shows how to invoke a method on a widget.

© 2021 Hitachi Power Grids. All rights reserved.

Table 27: Attributes of the js-invoke action

Attribute	Description	Remarks
name	Name of the method to be called	Required
on	Name of the widget, if the method of a widget is called	Optional

Invoking a JavaScript method from the user context

Example 46

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <script type="client">
view.greet = function(msg) {
alert(msg);
}
        </script>
    </head>
    <body>
        <button>Click me!
            <action event="click" type="js-invoke" name="greet">
                <parameter value="'Hello world'" />
            </action>
        </button>
    </body>
</document>
```

Invoking a JavaScript method on a component

Example 47

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="v" />
    </head>
    <body>
        <edit variable="v" id="e" type="text" />
        <button>Click me!
            <action event="click" type="js-invoke" name="focus" on="e" />
        </button>
    </body>
</document>
```

Navigate

This action closes all opened non-permanent views and navigates to a new view. Parameters for the opened view are defined with the parameter elements. Table 28 lists the attributes of the navigate action.

Table 28: Attributes of the navigate action

Attribute	Description	Remarks
name	Name of the view to be opened	Required

Nothing

This action does nothing. This action does not accept any attributes.

Open-modal

This action opens a modal dialog box. Parameters for the opened view are defined with the parameter elements. Table 29 lists the attributes of the open-modal action.

© 2021 Hitachi Power Grids. All rights reserved.

Table 29: Attributes of the open-modal action

Attribute	Description	Remarks
name	Name of the view to be opened	Required
close-on-escape	If true, the modal can be closed by pressing ESC.	Optional. Default false.
close-on-lost-focus	If true, the modal is closed when the modal dialog box loses focus.	Optional. Default false.
modal-style	If false, then modal does not uses the webui-modal class. This turns off certain CSS styles that are defined for a modal dialog box.	Optional. Default true.

Open-popover

This action opens a popover dialog box. Parameters for the opened view are defined with the parameter elements. Table 30 lists the attributes of the open-popover action.

Table 30: Attributes of the open-popover action

Attribute	Description	Remarks
name	Name of the view to be opened	Required
title	Title of the popover	Optional
timeout	Time in milliseconds after the popover is automatically closed. If 0, then the popover is not closed automatically.	Optional. Default value is 0.
show-title	If false, popover's title is not displayed.	Optional. Default value is true.
dismissible	If false, user can close popover by clicking outside the popover dialog box.	Optional. Default value is true.
show-close-button	If true, popover will have a close button.	Optional. Default value is true.
darken-background	If true, the area outside the popover is darkened.	Optional. Default value is false.
darken-background-content-highlight	Defines area that is not darkened. The area is defined as an object with properties left, top, width, and height. The properties value are the screen coordinates.	Optional
result	Variable where the result of the view is stored	Optional

Overlay

This action opens a given view as an overlay. Table 31 lists the attributes of the overlay action. Parameters for the opened view can be given with the parameter elements. For more information, see Section 5.6.

Table 31: Attributes of the overlay action

Attribute	Description	Remarks
name	Name of the view to be opened	Required

© 2021 Hitachi Power Grids. All rights reserved.

Redirect

This action redirects the current view to a new view. The difference between redirect and navigate is that if redirect is used inside a view's widget, redirect affects only that view's widget. Table 32 lists the attributes of the redirect action. Parameters for the opened view can be given with the parameter elements.

Table 32: Attributes of the redirect action

Attribute	Description	Remarks
name	Name of the view to be opened	Required

Set

This action sets a value to the variable.

Table 33: Attributes of the set action

Attribute	Description	Remarks
name	Name of the variable that is set	Required
value	Expression for the new value	Required

Show-confirmation-box

This action shows the confirmation box as a popover. The attributes are same as in the open-popover action (see Table 30). Table 34 lists the parameters of the confirmation box view. Example 48 shows how to use the show-confirmation-box action.

Table 34: Parameters of the configuration box view

Parameter	Description	Remarks
message	Question asked from the user	Required
buttons	Array of button definitions. Each button definition is an object with keys type and text. Type can be either OK or Cancel.	Required
extraParams	Extra parameters that are passed to the results	Optional

Using the show-confirmation-box action

Example 48

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable name="r" type="internal" />
    </head>
    <body xmlns:h="http://www.w3.org/1999/xhtml">
        <h:p>Result: {r.confirmed}</h:p>
        <button>Click me!
            <action type="show-confirmation-box" result="r">
                <parameter name="message" value="'Acknowledge alarm?'" />
                <parameter name="buttons" value="[{type: 'ok', text:
'yes'}, {type: 'cancel', text: 'no'}]" />
            </action>
        </button>
    </body>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

show-view-in-tab

This action opens a view in a tab. Table 35 lists the attributes of the show-view-in-tab action. The view parameter can be defined with the parameter elements. This action only works if page is opened from desktop-layout view.

Table 35: Attributes of the show-view-in-tab action

Attribute	Description	Remarks
name	Name of the view to be opened	Required
new-tab	New view is to be opened in a new tab. Both new-tab and active-tab should not be defined.	Optional. Default false.
active-tab	New view is to be opened in the active tab. Both new-tab and active-tab should not be defined.	Optional. Default true.

open-right-pane

This action opens the **control** dialog box to the right pane. This action does not accept any parameters, but there are certain parameters that must be provided to the **control** dialog box. These are provided with the parameter element. Table 36 lists the parameters of the **control** dialog box. This action only works if page is opened from desktop-layout view.

Table 36: Parameters of the control dialog box

Name	Description
ObjectReference	Object reference as string
ControlDialogName	Name of the control dialog box to be opened
SymbolName	Name of the symbol to be displayed with the control dialog box

close-right-pane

This action closes the right pane. This action does not accept any parameters. This action only works if page is opened from desktop-layout view.

5.5 Icons

Workplace X contains Workplace X font icon, Font awesome icons, and Bootstrap Glyphicons. There are a few ways to use these icons. Example 49 shows how font icons can be used in a view.

View using font icons

Example 49

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <body>
        <icon name="fa/play" />
        <icon name="webui/ui_alarm_bell" />
    </body>
</document>
```

It is also possible to use custom icons. The icon's URI is defined using the icon element in the head section. It is practical and efficient to use data URIs for representing icons, but also normal URIs can be used. Example 50 shows how custom icons can be used. Note that the example is not complete as the data URI is stripped.

© 2021 Hitachi Power Grids. All rights reserved.

View using a custom icon

Example 50

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <icon name="pause">
uI
wF4pT92AAAAB3RJTUUH4wkGDCoXkd1OqgAAABl0RVh0Q29tbWVudABDcmVhdGVkIHdpdGggR0l
NUFeB
DhcAAABASURBVEjHY/z//z8DLQETA43BqAUDbwELVlHe0tuYgp
+7VYmRHY2DUQtGLRi1YNSCUQtGLSAHMI627EYtoBgAAJWkDDsoMjjZAAAAAElFTkSuQmCC</
icon>
    </head>
    <body>
        <icon name="pause" />
    </body>
</document>
```

List of available icons can be seen in the reference https://<computer name>/blank/reference/generic/icon.

5.6 Overlays

Overlays are the method to open views in the same root node. With SVG content, the effect is that views are opened on top of each other. The HTML content with default CSS rules, views are displayed in a row. With CSS, different placing of views can be achieved. Example 51 shows that overlays can be opened and closed using actions.

View using overlay actions

Example 51

```
a.xml:
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <body>Hello from A</body>
</document>

b.xml:
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <body>Hello from B</body>
</document>

main.xml:
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <body>
    <button>Show a<action type="overlay" name="a" /></button>
    <button>Show b<action type="overlay" name="b" /></button>
    <button>Hide a<action type="delete-overlay" name="a" /></button>
    <button>Hide a<action type="delete-overlay" name="b" /></button>
  </body>
</document>
```

Overlays can also be manipulated with JavaScript using the *API.Views.Overlays* interface. This interface can be implemented using the view's *getOverlays* function.

© 2021 Hitachi Power Grids. All rights reserved.

Section 6 Modules

Sometimes parts of a functionality are shared between different views. This shared functionality can be implemented in modules which can be used from many views. Modules are XML documents. The content of the module resembles views and the root elements contain the same XML namespace as the view's root element.

6.1 Defining and using modules

A simple display can show the values of various signals. These signals can be shown in different colors, based on the status of the signal as shown in the Example 52. If there are multiple similar displays, reuse the mapping definitions.

Using mapping function for conditional styling

Example 52

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <mapping name="az-color">
            <step value="'red'" when="1" />
            <step value="'red'" when="2" />
            <step value="'#fc0'" when="3" />
            <step value="'#fc0'" when="4" />
            <otherwise value="'green'" />
        </mapping>
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <details n:LN="ESTHA1_MEC">
            <summary>Interesting value</summary>
            <div>
                <span s:fontWeight="bold" s:color="{`az-
color`(`20.AZ`)}">{`20.OV`}</span>
{`20.ST`} {`20.OX`}
            </div>
        </details>
    </d:body>
</document>
```

Example 53 displays a simple module that defines color mapping function. The module can contain elements that are defined in the head of a view, for example, scripts and styling information.

Simple module

Example 53

```
<?xml version="1.0"?>
<module xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
       <mapping name="color">
          <step value="'green'" when="0" />
          <step value="'red'" when="1" />
          <step value="'red'" when="2" />
```


© 2021 Hitachi Power Grids. All rights reserved.

```
            <step value="'#fc0'" when="3" />
            <step value="'#fc0'" when="4" />
        </mapping>
    </head>
</module>
```

The modules are stored in the modules folder. The name of the module is the name of the file containing the module. This module is represented as \sc\apl\<myapplication>\views \modules\my-module.xml file.

This module can be used in a view, using a use element. For example, see section

The content of the head element is replaced with single-use element. Example 54 displays a view using the module.

View using a module

Example 54

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <use name="my-module" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <details n:LN="ESTHA1_MEC">
            <summary>{`20.IL`.STA} {`20.IL`.BAY} {`20.IL`.DEV}</summary>
            <div>
                <span class="measurement"
s:color="{color(`10.AZ`)}">{`10.OV`}</span> {`10.ST`} {`10.OX`}</div>
            <div>
                <span class="measurement"
s:color="{color(`20.AZ`)}">{`20.OV`}</span> {`20.ST`} {`20.OX`}</div>
            <div>
                <span class="measurement"
s:color="{color(`21.AZ`)}">{`21.OV`}</span> {`21.ST`} {`21.OX`}</div>
        </details>
    </d:body>
</document>
```

Using a module from another module is also possible. This works similar to the module from a view.

6.2 Blocks

The head section helps to re-use parts in the module. To re-use parts of the body, blocks can be used. Blocks can be included using the include element. Example 55 is a module with a single block. Modules can contain many arbitrary blocks.

Module with a block

Example 55

```
<?xml version="1.0"?>
<module xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <block name="my-block">
    <head>
      <variable name="hello" type="internal" default="'Hello world!'" />
    </head>
    <body xmlns="http://www.w3.org/1999/xhtml">
      <p>{hello}</p>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    </body>
  </block>
</module>
```

Example 56 contains a view that uses the block defined in the Example 55. The head section of the block is added to the view in the place of the module's use element, but only if the block is used in the view. Example 56 displays the module as my-module.

View using a block

Example 56

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <use name="my-module2" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <p>Greetings from a module:</p>
        <d:include name="my-block" />
    </d:body>
</document>
```

6.3 Splitting view to different files

Modules can be combined from multiple files, as similar to views. Instead of file, module can be a directory. To split the module into files, create a marker file; module.json. This must contain a JSON object. An empty object is allowed.

In the modules the names of all the files are considered unlike view files. The XML file must have the same name as the module, that is the name of the folder. Styles and scripts for the head section must also have the same name. Styles and scripts of blocks must have the name of the block.

For example, if there is a module called my-module with JavaScript and CSS in a head section and a block called my-block with a JavaScript and a CSS file, then the module consists of the file as listed in Table 37.

Table 37: Files in an example module

File name	Format	Description
<library>/modules/my-module/module.json	JSON	Marker file
<library>/modules/my-module/my-module.xml	XML	Main file of the module. See Example 57 .
<library>/modules/my-module/my-module.js	JavaScript	JavaScript content that is appended to the head section of the module.
<library>/modules/my-module/my-module.css	CSS	CSS content that is appended to the head section of the module.
<library>/modules/my-module/my-block.js	JavaScript	JavaScript content that is appended to a block called my-block.
<library>/modules/my-module/my-block.css	CSS	CSS content that is appended to a block called my-block.

Example 57 displays a possible content of the main file, my-module.xml.

Shell of for a module that is combined from separate files

Example 57

© 2021 Hitachi Power Grids. All rights reserved.

```
<?xml version="1.0"?>
<module xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <!-- my-module.js and my-module.css will go here. -->
  </head>
  <block name="my-block">
    <head>
      <!-- my-block.js and my-block.css will go here. -->
    </head>
  </block>
</document>
```

Scripts can either be **JavaScript** or **TypeScript** files and styles can either be **CSS** or **Less** files. The head section or a single block can have any number of scripts or style files.

© 2021 Hitachi Power Grids. All rights reserved.

Section 7 Extensions

Extension provides a way to configure the existing content of the user interface. This section describes the extension mechanisms. The process to extend the existing user interface is described in Section 11. Extending the user interface more specifically is described in Section 11.5.

Extensions are based on defining extensions, that is to define the extension process of views and providing implementation to extensions. Two types of extensions can be defined: Extensions points and Options. Extensions points are places that allow to add more functionalities. Options provide a way to disable or enable content.

7.1 Defining extension points and options

Extension points and Options are defined using the extension namespace. Name of the extension namespace is http://www.webui-ap.com/schemata/extension/1.0. Example 58 shows a simple view with an extension point and an option. Extension points are defined with point elements. Each extension point has a name.

Hello World example with extensions

Example 58

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:h="http://www.w3.org/1999/xhtml"
    xmlns:x="http://www.webui-ap.com/schemata/extension/1.0">
    <body>
        <h:div x:option="greet">Hello world!</h:div>
        <x:point name="my-extension-point" />
    </body>
</document>
```

In Example 58 there are two extensions, first one is an option inside div element called greet. This allows that the div element can be disabled with the extension. The second extension is a point called my-extension-point that can be used to add content. Extensions allow someone else to modify the view in a controlled method.

7.2 Implementing extensions

Extensions can be implemented using the extension files. These are stored in extensions folder. Extension documents are XML files. Example 59 displays a simple extension that is an extension to the Example 58. The extension disables the default greeting using an option element and adds a new one with an implementation element.

Simple extension document

Example 59

```
<?xml version="1.0"?>
<x:extension xmlns:x="http://www.webui-ap.com/schemata/extension/1.0"
    xmlns="http://www.w3.org/1999/xhtml">
    <x:apply mask="helloworld" />
    <x:implementation name="my-extension-point">
        <div>Hello from extension!</div>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    </x:implementation>
    <x:option name="greet" enabled="false" />
</x:extension>
```

The apply element defines the scope of the extension. The extension is only applied to document with a given name. This extension works only if the original document is stored as helloworld.xml. The extension mechanism modifies the original document as displayed in Example 60.

End result after applying the extension

Example 60

```
<?xml version="1.0"?>
<document
  xmlns="http://www.webui-ap.com/schemata/view/1.0"
  xmlns:h="…”>
   <body>
      <h:div>Hello form extension!</h:div>
   </body>
</document>
```

With extension points the content of the body element can be modified. The header section of a view can be extended using the head element in the extension namespace. Example 61 displays an extension document with a head extension. The extension adds a simple JavaScript snippet of the helloworld document.

Extension document with a head element

Example 61

```
<?xml version="1.0"?>
<x:extension xmlns:x="http://www.webui-ap.com/schemata/extension/1.0"
    xmlns:d="http://www.webui-ap.com/schemata/view/1.0">
    <x:apply mask="helloworld" />
    <x:head>
        <d:script type="client">alert("Hello!");</d:script>
    </x:head>
</x:extension>
```

The head element of the extension namespace may contain any elements that are permitted in the head element of the document namespace. This should be used with caution, since it is easy to make extensions that are not compatible with the rest of the system. For example, adding a new mandatory external variable breaks the system, because this new variable is not provided when the view is opened from the existing code.

Extensions are used in situations where view writer allows someone else to extend the view. It is possible to have the extension implementation in a different library than the view itself. For example, SYS600 provides several extension points that can be implemented in different projects. See Section 11.5.

© 2021 Hitachi Power Grids. All rights reserved.

Section 8 Libraries

Views and other content are organized in libraries. Each library can contain different types of resources. Different resource types are stored in different subdirectories.

There are three types of libraries.

- *Public libraries*: The content of these libraries is served to authorized and non-authorized clients. Currently, there is one public library called public.
- *System libraries*: The content of these libraries is served only to authorized clients. System libraries can be used for implementing new reusable features.
- *Application specific libraries*: The content of these are served to the client logged in to the application. There can only be one application library for each application, and these are stored in the **/sc/apl/<myapplication>/views** folder.

The possible content of a library is listed in Table 38.

Table 38: The content of a library

Name	Description
dialogs	Folder containing all views.
js	Folder containing all automatically loaded JavaScript files.
styles	Folder containing all automatically loaded CSS files.
resources	Folder containing all other resources, the files that the client can load.
rights	Authorization right definitions.
translations	Folder for translation files.
manifest.json	File that describes the library.

Each library must contain a manifest.json file. The manifest contains a JSON object. The structure is explained in the Table 39. If many libraries have the same resource, the used one is considered from the library with the highest priority. Third-party libraries can be integrated to the system.

A third-party component can be integrated by placing component's JavaScript, CSS, and other resource files into js, styles, and resources folder.

Table 39: The content of a library manifest

Field	Description
Workplace X Library	A marker field to indicate a library. The value must be 1.0. If incompatible changes to the library format are introduced, then the number may change.
public	True or false. Defines if the content of the library is accessible without logging in or not.
css-order	Defines the loading order of CSS files in the library. With CSS files, the loading order matters. The content is an object. Keys are file names and values are number. Files are loaded in ascending numerical order. If some CSS file is not listed, the default value is set as 1. If the value is -1, then the CSS file is not loaded by default. By default, all css files in styles folder are loaded.
js-order	Similar as css-order, but for JavaScript files in scripts folder.
priority	Priority of the library. System libraries have a default priority of 100. The application library has a priority of 200.

© 2021 Hitachi Power Grids. All rights reserved.

Libraries can be stored either as a folder or as a ZIP file. The content of the folder can be edited while the system is running, but modifications to ZIP files are not followed. The folder approach is good for developing the system, but ZIP is better for distributing the content.

8.1 Installing libraries

There are three types of content stored in the WebUI libraries:

- System content distributed with the SYS600
- Content created using View Builder
- Content created by users

System content is part of SYS600 installation. System content can be extended by various WebUI extension mechanisms, but it cannot be directly modified.

User created libraries can be installed to **/sc/mod** folder. Such content can also be used in various projects.

Project specific content is part of the specific project only. This content is available in **/sc/apl/<application>/views** folder. This content is typically generated by View Builder.

© 2021 Hitachi Power Grids. All rights reserved.

Section 9 Localization

This section describes the used localization and internationalization system. The basic principle is that views are written in English and with language resources data, these can be translated into any language.

Localization is the operation that gives the program all the needed information so that this can handle both input and output in a method that is correct for the native language and for the cultural habits of the user. Localization includes translating, applications' text to the language of the user. This also includes things like text orientation, date and time formats, the calendar used, and even color scheme and user interface layout of the application.

Internationalization is the operation that enables a program to support multiple languages. This is a generalization process, by which the program is untied from calling only English strings or other English language specific habits. In Workplace X, internationalization is done automatically. Views can be written in English and translatable texts are extracted automatically from the view.

Some tools are provided for adjusting date and time format for local preferences. Customers can localize Workplace X applications further by customizing the views of the application.

Workplace X uses the Gettext system for internationalization. This has the advantage that many parts of the internationalization toolchain, like editors for creating translations, are already available and mature. Key concepts of localization are listed in Table 40.

Table 40: Glossary of localization terms

Term	Explanation
Context	The meaning of the text can differ depending on the context. This is more frequent with single words and short fragments. For example, the English word Letter can mean either postal mail or a single character, depending on usage. The Gettext solves this problem by allowing a context attribute added to any translatable string.
Plural Forms	Plural forms become an issue whenever text contains a number that can change run time. Plural forms are complicated because different languages have different rules for pluralization, different number of plural forms and different number of rules for pluralization (for example, Arabic has 6 different plural forms). Gettext solves this problem by having a system that allows a translator (or the tool used for translation) to define pluralization rules and different translations for all plural forms in translation file.
Gettext	An internationalization and localization system commonly used for writing multilingual programs.
Harvesting	The process of extracting translatable strings from the application. This process is explained in Section 9.2.
i18n	Shorthand for internationalization. I18n is constructed from first and last letters and from the length of the word internationalization.
l10n	Shorthand for localization. Constructed in the same way as i18n.
MO file	Machine Object file. The binary version of the PO file. Not used in Workplace X.
PO file	Portable Object file. A human readable translation file.
POEdit	Editor for creating PO files. POEdit is the recommended translations editor for Workplace X.
POT file	Portable Object Template. Template file that contains all harvested strings of the application. POT files are used as a starting point for PO files.

© 2021 Hitachi Power Grids. All rights reserved.

9.1 Translation process

The translation process consists of two steps:

- harvesting, where translatable texts are collected from source files
- translation, where harvested strings are translated to target language(s)

Harvested texts are collected in a translation template, essentially a translation file without any translations for collected strings. In translation, the template file is copied, and appropriate translations are added. The process is outlined in Figure 23.

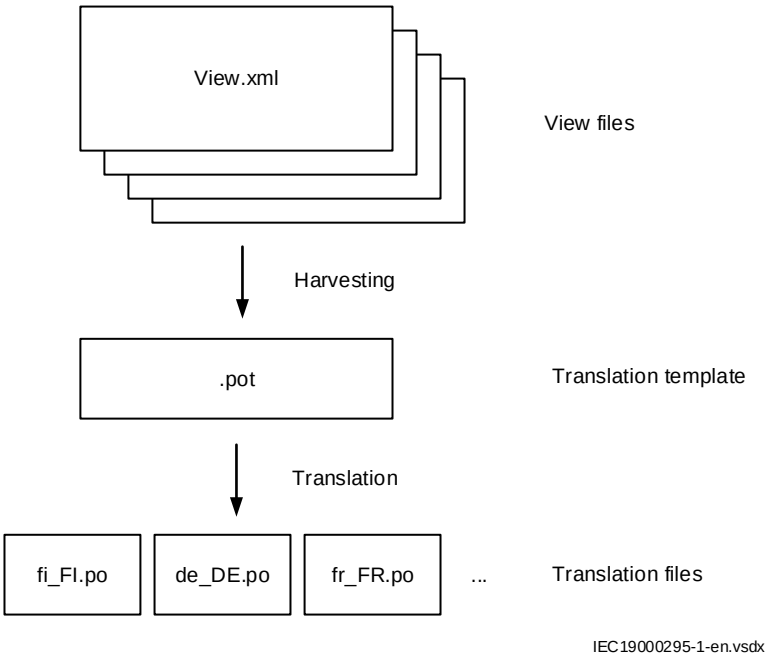


Figure 23: Translation process

9.2 Harvesting

Harvesting is done with the command line tool harvest_application.bat. The tool takes only a singleargument, the name of the application that should be translated. For example:

harvest_application.bat <myApplication>

The harvesting tool creates myApplication.POT file in the application directory. Harvesting tool is located in folder c:\Program Files\ABB\MicroSCADA Pro\Workplace X\

9.3 Translation

Translation is performed by copying the template (.POT) file and filling in translated texts. This process can be done manually, but a translation tool makes the process much faster and easier.

The recommended tool for translations is POEdit. Open source version is available for download from https://poedit.net/download . The translation file must be placed in the library directory and named as <country code>_<language variant>.PO. The <country code>

© 2021 Hitachi Power Grids. All rights reserved.

and <language variant> are two letters, ISO 639-1 language codes. For example, Austrian German translation file is named as de_AT.PO.

9.3.1 Creating translation file

Translation files should be placed under directory named translations in the application view directory, for example, c:\sc\apl\<myapplication>\views\translations.

Example 62 displays, a localisationDemo.xml, located in c:\sc\apl\<myApplication>\views \dialogs.
Text element localization

Example 62

```
<?xml version="1.0" encoding="utf-8"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:h="http://www.w3.org/1999/xhtml">
    <head>
        <title>Localisation Demo</title>
    </head>
    <body>
        <h:p>This text is localized</h:p>
    </body>
</document>
```

Harvester can be executed with the following command:

C:\Program Files\ABB\MicroSCADA Pro\Workplace X>harvest_application.bat <myApplication>
Harvester scans the view file and creates following translation template (.POT) as displayed in Example 63. The template file will be in application directory, i.e. c:\sc\apl\<myapplication>\.

Text element localization template (POT) file

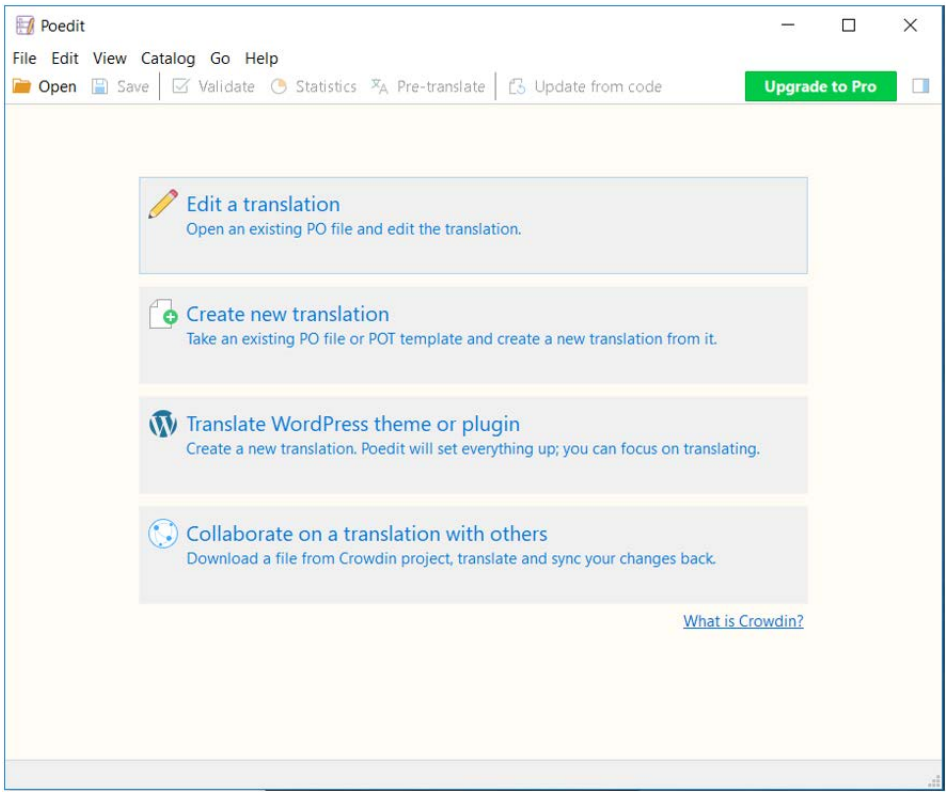
Example 63

```
msgid ""
msgstr ""
"Project-Id-Version: ABB SYS600 Web UI Library demoLibrary\n"
"Content-Type: text/plain; charset=UTF-8\n"
"POT-Creation-Date: 2018-08-14T11:25:40.038Z\n"
"Language-Team: \n"
"MIME-Version: 1.0\n"

#. Substitute the translated name of the language instead of "Default
english" below.
#. it will be shown in language selection setting as the entry for the
translation.
msgctxt "WebUI Framework - settings"
msgid "Language name"
msgstr "Default english"

#: demoLibrary/localisationDemo.xml: /null/document/head/title
msgid "Localisation Demo"
msgstr ""

#: demoLibrary/localisationDemo.xml: /null/document/body/p
msgid "This text is localized"
msgstr ""
```

To create a translation file:

1. Use **POEdit** to open a POT file.

© 2021 Hitachi Power Grids. All rights reserved.



IEC19000296-1-en.vsdx

Figure 24: POEdit opening screen

2. Select **Create new translation** and select the .POT file to access the language selection screen.

© 2021 Hitachi Power Grids. All rights reserved.

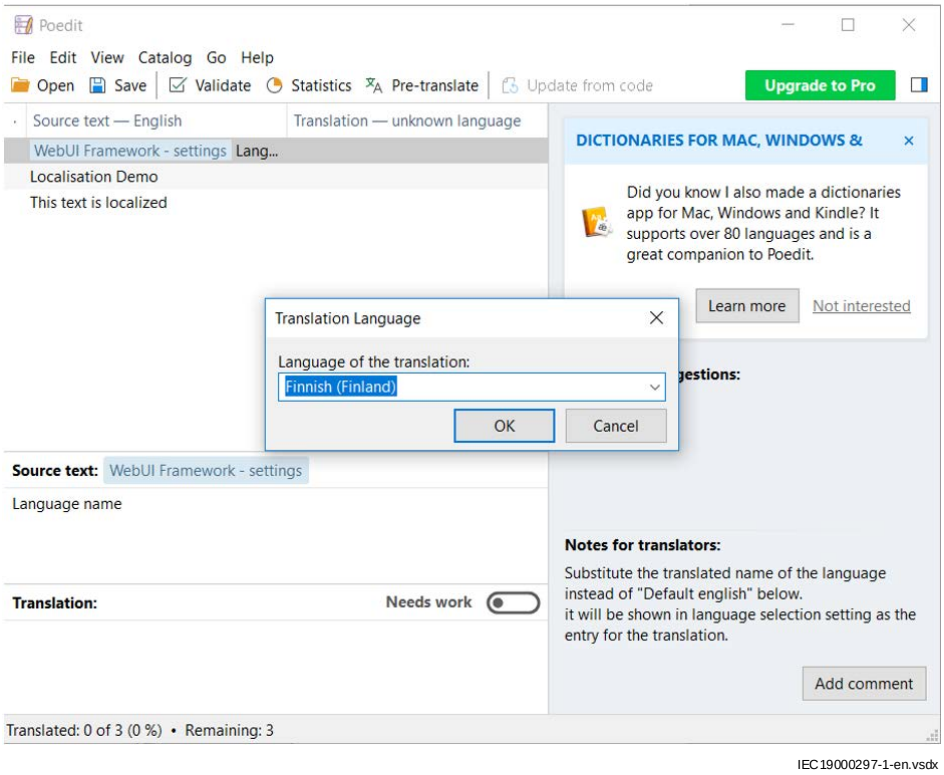
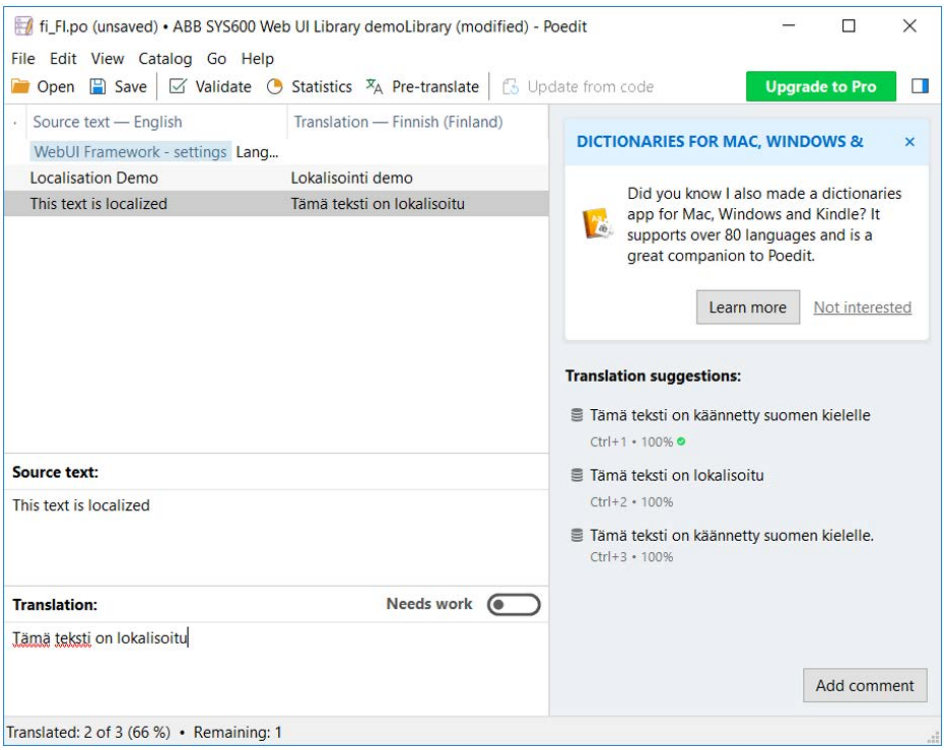


Figure 25: Choosing a language for translation

3. Select **language** to start the translation.

© 2021 Hitachi Power Grids. All rights reserved.



IEC19000298-1-en.vsdx

Figure 26: Translating texts with POEdit

4. Select a line from the upper section, and type translations in the **Translation** box.
   POEdit selects the language appropriate for plural rules and number of plural forms automatically.
5. After writing translations, save the file.
   POEdit automatically suggests the correct file name for the translation file.
   The translated (.PO) file, is received as displayed in Example 5.
   Translation file

Example 64

```
msgid ""
msgstr ""
"Project-Id-Version: ABB SYS600 Web UI Library demoLibrary\n"
"Content-Type: text/plain; charset=UTF-8\n"
"POT-Creation-Date: 2018-08-14T11:25:40.038Z\n"
"Language-Team: \n"
"MIME-Version: 1.0\n"
"PO-Revision-Date: \n"
"Content-Transfer-Encoding: 8bit\n"
"X-Generator: Poedit 2.1.1\n"
"Last-Translator: \n"
"Plural-Forms: nplurals=2; plural=(n != 1);\n"
"Language: fi_FI\n"

#. Substitute the translated name of the language instead of "Default
english" below.
#. it will be shown in language selection setting as the entry for
the translation.
msgctxt "WebUI Framework - settings"
msgid "Language name"
msgstr ""

#: demoLibrary/localisationDemo.xml: /null/document/head/title
msgid "Localisation Demo"
```


© 2021 Hitachi Power Grids. All rights reserved.

```
msgstr "Lokalisointi demo"

#: demoLibrary/localisationDemo.xml: /null/document/body/p
msgid "This text is localized"
msgstr "Tämä teksti on lokalisoitu"
```

The new translation is automatically available for users in Workplace X/WebMonitor.

9.3.2 Translating scripts

String constants in script blocks are not localized automatically. These are encased in Gettext calls.

i18n._("This string is translated");

Sometimes it is useful to mark a string constant as available for translation but do the actual translation later For example, when a string constant is defined and stored at the initialization time of an application and is displayed later when user action prompts this. Typically, this is not required to localize the strings at initialization time because the user might change language settings during run time.

Workplace X framework provides an explicit feature for handling this issue by using the _noop functions.

These functions have the same syntax as other Gettext functions but are not used in the actual translation. These functions provide a hook for the harvester to find the localizable strings.

i18n._noop("This text will be harvested but not translated");

A variable and script block is shown in Example 65.

View with localized text in script block

Example 65

```
<?xml version="1.0" encoding="utf-8"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:h="http://www.w3.org/1999/xhtml">
    <head>
        <variable type="internal" name="text_plain" default="'This text
is not localized nor harvested'" />
        <script type="client">
self.set("text_plain", i18n._('This text is localized in script'));
        </script>
    </head>
    <body>
        <h:p>{text_plain}</h:p>
    </body>
</document>
```

In Example 65 the head block with the variable default is neither harvested nor localized. A variable value can be localized inside the script block, if required.

After harvesting the view, a .POT file with an entry for the script text is received, as shown in Example 66.

Localization template entry for script text

Example 66

```
<?xml version="1.0" encoding="utf-8"?>
<document xmlns = "http://www.webui-ap.com/schemata/view/1.0">
    <head></head>
    <body>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
        <button text="Press button to do something" />
        <edit type="string" label="Random musings" placeholder="Write
something here" />
    </body>
</document>
```

This .POT file can be translated as shown in Example 65.

9.3.3 Component translation

Workplace X component attributes that contain user-visible texts are also harvested and localized. The complete list is available in Workplace X Developer Documentation, section *Components*. To access documentation, see section 2.5.

A simple view with the button and an editor can be considered. Button has text and editor has label and placeholder texts. All these texts are visible to the user, so these are localizable. Example 67 displays the view file.

Workplace X components with localizable texts

Example 67

```
<?xml version="1.0" encoding="utf-8"?>
<document xmlns = "http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="internal" name="v" />
    </head>
    <body>
        <button text="Press button to do something" />
        <edit variable="v" type="string" label="Random musings"
placeholder="Write something here" />
    </body>
</document>
```

From the view, harvester generates a .POT file with entries for the localizable component attributes. Example 68 displays the entries.

Localization template entries for component texts

Example 68

```
#: demoLibrary/componentLocalisationDemo.xml: /null/document/body/button
msgid "Press button to do something"
msgstr ""

#: demoLibrary/componentLocalisationDemo.xml: /null/document/body/edit
msgid "Write something here"
msgstr ""

#: demoLibrary/componentLocalisationDemo.xml: /null/document/body/edit
msgid "Random musings"
msgstr ""
```

The button text, the editor label and the placeholder are harvested and can be translated like the other texts.

9.3.4 Variable substitution

Variable values can be inserted inside localizable strings. Use the variable reference in text elements, and the Harvester harvests and inserts a reference to the run time value for the variable.

For scripts, the Gettext functions uses sprint format:

© 2021 Hitachi Power Grids. All rights reserved.

i18n._("Variable value inside string: %1$s. Another: %2$s", variable1, variable2);

Example 69 displays a view file with variables in both script and text element.

Using variables in localized strings

Example 69

```
<?xml version="1.0" encoding="utf-8"?>
<document
    xmlns = "http://www.webui-ap.com/schemata/view/1.0"
    xmlns:h="http://www.w3.org/1999/xhtml">
    <head>
        <variable type="internal" name="some_number" default="42" />
        <variable type="internal" name="script_text" default="''" />
        <script type="client">
var scriptVariable = 34;
self.set("script_text", i18n._('The number used in script is: %1$s',
scriptVariable));
        </script>
    </head>
    <body>
        <h:p>The number is: {some_number}</h:p>
        <h:p>{script_text}</h:p>
    </body>
</document>
```

After harvesting, a POT file with entries for strings with variable references is received, as displayed in Example 70.

Template entries for localized strings with variables

Example 70

```
#: demoLibrary/variableSubstitutionDemo.xml: /null/document/head/script
msgid "The number used in script is: %1$s"
msgstr ""

#: demoLibrary/variableSubstitutionDemo.xml: /null/document/body/p
msgid "The number is: %1$s"
msgstr ""
```

The harvester finds both strings and changes variables into sprint style references. When translating, the references must be used as is, but the respective places in the string can be changed, as displayed in Example 71.

Translation with variable substitution

Example 71

```
#: demoLibrary/variableSubstitutionDemo.xml: /null/document/head/script
msgid "The number used in script is: %1$s"
msgstr "%1$s on ohjelmassa määritetty numero"

#: demoLibrary/variableSubstitutionDemo.xml: /null/document/body/p
msgid "The number is: %1$s"
msgstr "Numero on: %1$s"
```

9.3.5 Plurals

Showing variable values in localized strings means that the use plural of form for the string is required. Workplace X localization has mechanisms to perform this function.

For markup elements, use new attribute i18n:plural to inform harvester that a plural form is needed:

{people} person has passed here

© 2021 Hitachi Power Grids. All rights reserved.

For scripts, use the Gettext function i18n._p, which takes the plural and singular forms as arguments

i18n._p(' %1$s people have passed here', '%1$s person has passed here', people);

Example 72 displays a view with some plural forms.

Using plural forms in localization

Example 72

```
<?xml version="1.0" encoding="utf-8"?>
<document
    xmlns = "http://www.webui-ap.com/schemata/view/1.0"
    xmlns:i18n = "http://www.webui-ap.com/schemata/i18n/1.0"
    xmlns:h="http://www.w3.org/1999/xhtml">

    <head>
        <variable type="internal" name="script_text" default="''" />
        <variable type="internal" name="fox_count" default="" />
        <script type="client">
var foxes = 0;
this.addFox = function() {
foxes = foxes + 1;
self.set("script_text", i18n._p('%1$s fox jumped over a lazy dog',
'%1$s foxes jumped over a lazy dog', foxes));
self.set("fox_count", foxes);
}
        </script>
    </head>
    <body>
        <h:p i18n:plural="{fox_count} foxes jumped over a lazy dog">
{fox_count} fox jumped over a lazy dog
        </h:p>
        <h:p>{script_text}</h:p>
        <button text="Press button to add a fox">
            <action type="js-invoke" name="addFox"/>
        </button>
    </body>
</document>
```

- Add xmlns:i18n="http://www.webui-ap.com/schemata/i18n/1.0" to the document element to use i18n attributes.
- Plural forms can use only one variable substitution in the string because the combination of singular and plural forms is not supported in the system.

After harvesting, a .POT file with entries for the plural forms is received, as displayed in Example 73.

Template file entries with plural forms

Example 73

```
#: demoLibrary/pluralsLocalisationDemo.xml: /null/document/head/script
msgid "%1$s foxes jumped over a lazy dog"
msgid_plural "%1$s fox jumped over a lazy dog"
msgstr[0] ""
msgstr[1] ""

#: demoLibrary/pluralsLocalisationDemo.xml: /null/document/body/p
msgid "%1$s fox jumped over a lazy dog"
msgid_plural "%1$s foxes jumped over a lazy dog"
msgstr[0] ""
msgstr[1] ""
```


© 2021 Hitachi Power Grids. All rights reserved.

```
#: demoLibrary/pluralsLocalisationDemo.xml: /null/document/body/button
msgid "Press button to add a fox"
msgstr ""
```

When translation starts, **POEdit** automatically finds the plural form rules for the selected language and adds the appropriate number of plural form pages to the translation section. The Figure 27 displays the **POEdit translation** screen for Polish, which has three plural forms. The screen displays the plural form tabs in the **Translation** box. The tab titles show the rule for each plural form.

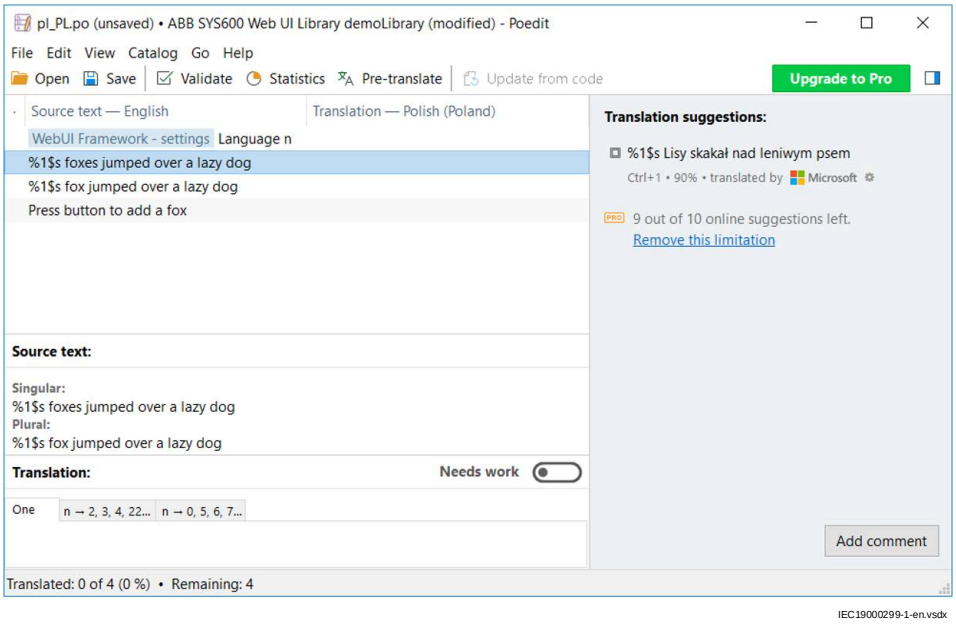


Figure 27: Translating plural forms in POEdit

When the translation is saved, **POEdit** creates a translation file with plural forms, as displayed in Example 74.

Translation file (Polish) with plural forms

Example 74

```
msgid ""
msgstr ""
"Project-Id-Version: ABB SYS600 Web UI Library demoLibrary\n"
"Content-Type: text/plain; charset=UTF-8\n"
"POT-Creation-Date: 2018-08-15T08:25:16.524Z\n"
"Language-Team: \n"
"MIME-Version: 1.0\n"
"PO-Revision-Date: \n"
"Content-Transfer-Encoding: 8bit\n"
"X-Generator: Poedit 2.1.1\n"
"Last-Translator: \n"
"Plural-Forms: nplurals=3; plural=(n==1 ? 0 : n%10>=2 && n%10<=4 && (n
%100<12 || n%100>14) ? 1 : 2);\n"
"Language: pl_PL\n"

#. Substitute the translated name of the language instead of "Default
english" below.
#. it will be shown in language selection setting as the entry for the
translation.
msgctxt "WebUI Framework - settings"
msgid "Language name"
msgstr ""
```


© 2021 Hitachi Power Grids. All rights reserved.

```
#: demoLibrary/pluralsLocalisationDemo.xml: /null/document/head/script
msgid "%1$s foxes jumped over a lazy dog"
msgid_plural "%1$s fox jumped over a lazy dog"
msgstr[0] ""
msgstr[1] ""
msgstr[2] ""

#: demoLibrary/pluralsLocalisationDemo.xml: /null/document/body/p
msgid "%1$s fox jumped over a lazy dog"
msgid_plural "%1$s foxes jumped over a lazy dog"
msgstr[0] ""
msgstr[1] ""
msgstr[2] ""

#: demoLibrary/pluralsLocalisationDemo.xml: /null/document/body/button
msgid "Press button to add a fox"
msgstr ""
```

The **POEdit** adds the appropriate rules to Plural-Forms -line.

After providing all the required plural forms in translation phase, the localization system applies the correct from based on the variable value automatically.

Using context

Similar to plurals, in text elements the context is defined with an i18n:context -attribute:

In scripts, a function with context parameter is used:

i18n._c(' mail, 'Write a letter');

Context and plurals can also be combined:

Write {count} letter

i18n._cp(' mail', 'Write %1$s letters', 'Write %1$s letter', count);

Example 75 displays a simple view of two text elements with different contexts.

Using contexts in localization

Example 75

```
<?xml version="1.0" encoding="utf-8"?>
<document
    xmlns = "http://www.webui-ap.com/schemata/view/1.0"
    xmlns:i18n = "http://www.webui-ap.com/schemata/i18n/1.0"
    xmlns:h="http://www.w3.org/1999/xhtml">
    <head></head>
    <body>
        <h:p i18n:context="Construction">This is a crane</h:p>
        <h:p i18n:context="Avian">This is a crane</h:p>
    </body>
</document>
```

Harvesting this view produces .POT file with two same strings, each with the respective context, as shown in Example 76.

Template file entries with context

Example 76

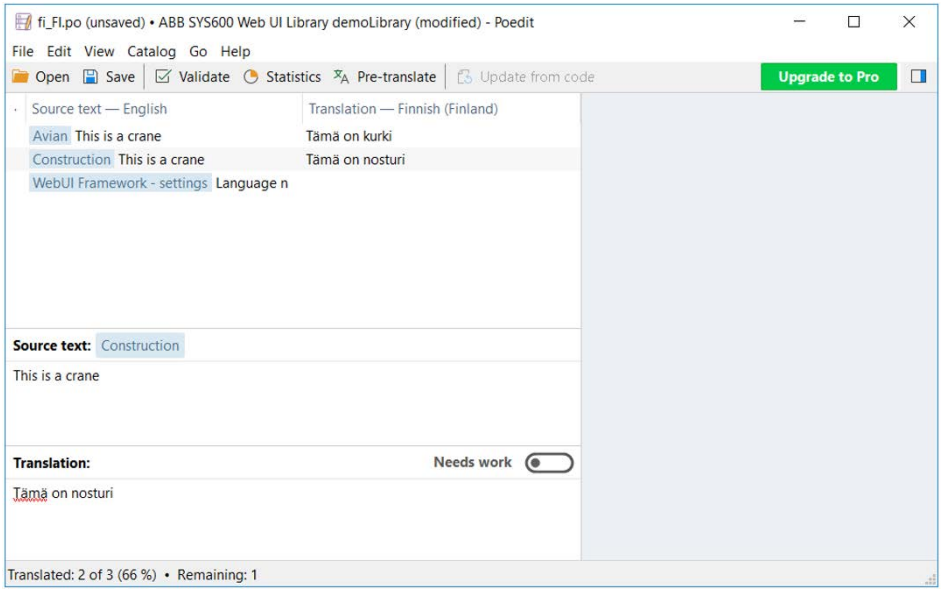
```
#: demoLibrary/contextLocalisationDemo.xml: /null/document/body/p
msgctxt "Construction"
msgid "This is a crane"
```


© 2021 Hitachi Power Grids. All rights reserved.

```
msgstr ""

#: demoLibrary/contextLocalisationDemo.xml: /null/document/body/p
msgctxt "Avian"
msgid "This is a crane"
msgstr ""
```

The exact same string is harvested twice, and both the strings have the respective context.

Figure 28 displays the **POEdit translation** screen, with context entries next to each translatable string.



IEC19000300-1-en.vsdx

Figure 28: Using context in POEdit

The translated .POT files have separate entries for the texts and for the respective translations, as displayed in Example 77.

Contexts in translation file

Example 77

```
#: demoLibrary/contextLocalisationDemo.xml: /null/document/body/p
msgctxt "Construction"
msgid "This is a crane"
msgstr "Tämä on nosturi"

#: demoLibrary/contextLocalisationDemo.xml: /null/document/body/p
msgctxt "Avian"
msgid "This is a crane"
msgstr "Tämä on kurki"
```


© 2021 Hitachi Power Grids. All rights reserved.

Section 10 Services

This section describes general mechanisms for invoking services. The list of available services is available in Workplace X Developer Document, section *Services*. To access documentation, see section 2.5.

Services are methods that can be called by the client. Each call can have one parameter, which is an object described in Services section in the reference documentation. Each call can return one or more responses. Last response is called final response and responses before the last response are called intermediate responses. Typically, services only return one response.

To be able to invoke a service, it needs to be declared using service element. Example 78 contains a view with a service that queries available attribute for the event list.

View querying list of available event list attributes

Example 78

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <service interface="EventStorageService"
method="QueryEventAttributes" result="{attributes: myvariable}" />
        <variable type="internal" name="myvariable" />
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <d:button>Click Me!
            <d:action type="invoke-service" name="EventStorageService/
QueryEventAttributes" />
        </d:button>
        <ul>
            <d:repeat for-each="myvariable" iterator="attr">
                <li>{attr.name}: {attr.description}</li>
            </d:repeat>
        </ul>
    </d:body>
</document>
```

Attribute of the script elements are listed in Table 41. List of available interfaces and methods are available in the separate reference. Script element itself does not cause invocation of the service and therefore, input parameters for the service are not defined in the service element.

Table 41: Attribute of the script elements

Attribute name	Description
interface	Name of the interface
method	Name of the method
result	Optional result expression describing how services result is handled
failure-policy	Either *throw* or *retry*

Failure policy defines what to do in case of connection failure. Value *retry* means that the call is retried. This means that client tries to do the call at least once. It is possible that the connection breaks after the call is already sent to the server, but the response is not received. In this case, with failure policy of *retry*, the call will be executed more than once. This mode is suitable for methods that query data since it does not matter if query is executed more than once.

© 2021 Hitachi Power Grids. All rights reserved.

If value of failure policy is omitted or is *throw*, it means that call will signal an error. In this mode, call is executed one time. If error is not handled, an error message will be shown to the user.

It is possible to attach actions to services. This happens by adding action elements under service elements. Actions has three possible event types: *final*, *intermediate*, and *error*. If event type is omitted, *final* is used. Actions can access the result of the service call using a variable called result.

Example 79 contains a view with an action. When queryViews call returns a value, its result is shown in a popup.

View that invokes queryView service

Example 79

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <service interface="WebServer" method="queryViews" />
        <variable type="internal" name="response" default="" />
        <script type="client">
self.showPopup = (value) => alert(JSON.stringify(value));
        </script>
    </head>
    <d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <d:button>Click again Me!
            <d:action type="invoke-service" name="WebServer/queryViews">
                <d:action event="final" type="js-invoke" name="showPopup">
                    <d:parameter value="response" />
                </d:action>
            </d:action>
        </d:button>
    </d:body>
</document>
```

Services can be invoked either with action element, as in Example 78, or using JavaScript. Example 79 contains a view that invokes a service using JavaScript.

View that invokes queryViews using JavaScript

Example 80

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
<head>
<role name="demo" />
<service interface="WebServer" method="queryViews" />
<script type="client">
self.showPopup = (value) => alert(JSON.stringify(value));
</script>
</head>
<d:body
xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
xmlns="http://www.w3.org/1999/xhtml">
<d:button>Click Me!
<d:action type="invoke-service" name="WebServer/queryViews">
<d:action event="final" type="js-invoke" name="showPopup">
<d:parameter value="response" />
</d:action>
</d:action>
</d:button>
</d:body>
</document>
```


© 2021 Hitachi Power Grids. All rights reserved.

Error handling can be done either using action element with event type error or with error callback in JavaScript. If service causes an error and there is no error callback, an error message is shown.

There are two types of errors:

1. It is not possible to do the call, for example, because there is an error in the network connection or some similar condition.
2. The call is made, but the operation failed. If operation failed, then normal response callback is invoked, and the error can be detected from the payload of the response.

© 2021 Hitachi Power Grids. All rights reserved.

Section 11 Desktop layout

This section describes the content provided with the SYS600 installation. It is possible to extend the SYS600 so that the new content is seemingly part of the new UI.

11.1 Menu structure

The desktop layout contains two menu items, which are configurable. Content can be added to the left side menu tree by adding a driver view. The driver view contains only definitions, not any visible elements. The driver view is loaded by the system and the menu structure is generated based on the definitions in the driver views. The view must contain a single variable called items. The view must contain a script that sets the variable to contain an object that describes the menu item. Fields of the object are listed in Table 42.

Table 42: Fields of the object describing a menu item in the left menu

Field	Description
type	Either *queryViews*, *queryFilters*, *queryFunction*, or *staticPath*. This defines the process of content generation on the menu. • *queryViews* means all the views with a given role are added to the menu. • *queryFilter* means menu items are generated based on the available filters. This is applicable only for alarms and events. • *queryFunction* means that the menu content is generated using a JavaScript function. • *staticPath* means that the menu item contains only one static entry defined by the path attributes.
order	The position of the menu item in the main tree. The process menu item has a position of 100, alarms have 200 and so on.
role	The role used for querying views. This is applicable only if the type is queryViews.
icon	Icon of the menu item. This is style name for either Workplace X font icon or Font awesome icon. See section 5.5 for more information about icons.
title	Title of the icon. Use the underscore function to translate the title of the icon.
path	A view that is opened if the main item is clicked.
filterKey	Parameters used for querying filters. This is applicable only if the type is queryFilters.
id	Suffix for element id of the menu item. The used prefix is webui-nav-.
function	Name of the function that is invoked. This is applicable only if the type is queryFunction. See Example 81.
linkable	If value is **true**, then content is available in Workplace X follower window.

Example 81 contains a definition for the menu that lists motors. The function attribute defines the names of the function that is called to receive the content of the menu. The function can return the structure like in the Example 81 or it can return a JavaScript promise object. Promise can be used if the list is not static and can be queried from the backend.

Generating a menu structure with JavaScript

Example 81

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
```


© 2021 Hitachi Power Grids. All rights reserved.

```
<head>
<role name="main-menu-item" />
<variable type="external" name="menuItem" />
<script type="text/javascript">
this.set("menuItem", {
"type": "queryFunction",
"order": 1500,
"function": "getMotors",
"icon": "webui_icon_24/ui_motor",
"title": i18n._("Motors")
});
self.getMotors = function(path) {
return [{
"title": "Overview",
"path": "motor-overview"
}, {
"name": "motor1",
"title": "Motor 1",
"path": "motor",
"parameters": {"id": 1}
}, {
"name": "motor2",
"title": "Motor 2",
"path": "motor",
"parameters": {"id": 2}
}];
}
</script>
</head>
</document>
```

For each menu item, a CollapsibleNode component is instantiated. This can be used for adding icons, programmatically. Example 82 contains an extension to the left pane. This extension subscribes a dummy datapoint that displays if the motor contains a warning and toggles the warning On and Off based on the information.

An extension that adds a warning icon to a navigation tree item

Example 82

```
<?xml version="1.0"?>
<x:extension xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
xmlns:x="http://www.webui-ap.com/schemata/extension/1.0">
<x:apply mask="left-pane" />
<x:head>
<d:script type="client">
//self.bindName("urn:Domain:MyMotor.1/warning", res => {
self.menu.findNodesByAttribute("name", "motor1").forEach(n => {
n.node.warning = res == true;
});
//});
</d:script>
</x:head>
</x:extension>
```

The return value of the function is a structure that contains a list of views. The fields with the explanations are listed in Table 43.

© 2021 Hitachi Power Grids. All rights reserved.

Table 43: Field of the structure defining the menu in the left pane

Field	Datatype	Descriptions
Name	String	Name of the entry. If the title is not given, then the name is used as the title. Name must be unique.
Title	String	Visible title of the entry which is optional. If not given, the name is used instead.
Path	String	Name of the view that is opened when clicked. If not given, no view is opened. This is useful entry as children.
Params	Object	Parameters supplied to the view.
Children	Array	Child entries. The array contains objects with fields as this object.

Top menu can also be extended using extensions. There are two extension points in the top menu: x-top-menu-header and x-top-menu-items. See Section 13.3 to view an example of using x-top-menu-header. Example 83 displays the process of adding a simple menu item to the top menu. Menu item will open a view called *helloworld* as a modal dialog. Extension works only if there exists a view with that name. Top menu items cannot be rearranged.

An extension adding an item to the top menu

Example 83

```
<?xml version="1.0"?>
<x:extension
xmlns:x="http://www.webui-ap.com/schemata/extension/1.0"
xmlns:d=" http://www.webui-ap.com/schemata/view/1.0">
<x:apply mask=”top-menu” />
<x:implementation name="x-top-menu-items">
<li>
<span>My item</span>
<d:action type="open-modal" name="helloworld" />
<d:action type="js-invoke" name="closeAction" />
</li>
</x:implementation>
</x:extension>
```

11.2 Control dialogs

Control dialogs consist of two sections: the primary section and the secondary section. The control dialog can only be opened with the primary section or with both the primary and the secondary section. Example 84 displays the process to open the primary section of the control dialog. The object reference must be correct. Object reference in the example does not work if there is no disconnector modeled with ViewBuilder with the name AA1.C1.Q07.QB1.

Opening only primary section of a control dialog

Example 84

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="process" />

    </head>
    <body>
        <button>Click me!
            <action type="open-popover" name="control/power/disconnector">
                <parameter name="ShowCompactView" value="true" />
                <parameter name="ObjectReference"
value="'AA1.C1.Q07.QB1'" />
                <parameter name="SymbolName" value="'Disconnector'" />
```


© 2021 Hitachi Power Grids. All rights reserved.

```
            </action>
        </button>
    </body>
</document>
```

Existing control dialogs can be extended, and new control dialogs can be created. Control dialogs are created by creating a new view. The control dialog of tap changer is shown in Example 85.

Control dialog for a tap changer.

Example 85

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <title>Tap changer</title>

        <use name="control/control-dialog" />

        <meta name="control-dialog-content" content="common, tap-
changer" />
        <meta name="indication-datapoint" content="tap_position" />
        <meta name="large-symbol" content="true" />
    </head>
</document>
```

The control dialog main view has only head section. Dialog needs to use the control/control-dialog module. This module will dynamically query the content and build the control dialog. Possible meta elements are listed in table 44.

Table 44: Meta elements for the main control dialog view.

Name	Remarks
control-dialog-content	List on content to be loaded to the control dialog. Mandatory.
indication-datapoint	Main datapoint for showing the data indication. This is used for loading titles of the controlled device. Mandatory.
device-symbol	Symbol to be shown in the header.
large-symbol	Is content of *large-symbol* is *true*, then more space is reserved for the device-symbol.

Control dialog are built from sections. Top part of the control dialog has three sections: header, primary, and messages. Different tabs in lower parts are different sections. Lower part contains five sections: overview, measurements, actions, alarms, and events. It is advisable to use at most three sections in the lower part since more than three tabs might not fit to the screen in all cases.

A simple example that will add a Hello world -message to disconnector control dialog is shown in Example 86. Blocks will have predefined set of external variables. They are explained in table 45.

A simple control dialog block that will add a hello world -message to disconnector control dialog.

Example 86

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="control-dialog-disconnector" path="messages"
priority="50" />
```


© 2021 Hitachi Power Grids. All rights reserved.

```
        <variable type="external" name="ObjectReference" default="''"/>
        <variable type="external" name="Title" default="''" />
        <variable type="external" name="DeviceSymbol" default="''" />
        <variable type="external" name="ShortName" default="''" />
        <variable type="external" name="LongName" default="''" />
        <variable type="external" name="ShowCompactView"
default="false" />
        <variable type="external" name="IndicationDataPoint"
default="''" />
        <variable type="external" name="CommandDataPoints" default=""/>

        <style>
            body {
                padding: 1em;
                background-color: #eeddcc;
            }
        </style>
    </head>

    <body>
            Hello world!
    </body>
</document>
```

Table 45: Parameters provided to control dialog sections.

Parameter	Remarks
ObjectReference	Content of the object reference parameter of the main control dialog.
Title	Title of the main control dialog. Applicable only for header, primary, and message sections.
DeviceSymbol	Control of the *DeviceSymbol* parameter of the main control dialog.
ShortName	Name attribute of the object defined in *ObjectReference*.
LongName	Path attribute of the object defined in *ObjectReference*.
ShowCompactView	Defines if compact view is to be shown. In compact view only header, primary, and messages sections are shown. Applicable only for header, primary, and message sections.
IndicationDataPoint	Name of the main indication datapoint defined in the main control dialog. Applicable only for header, primary, and message sections.
CommandDataPoints	An object with keys primary and secondary. Primary will contain list of primary command data points and secondary will contain list of secondary command data points.

List of control dialogs in the system is listed in table 46. Table list the content of the control-dialog-content meta element.

Table 46: Control for control dialogs.

Control dialog	Content	Used in
alarm	alarm	Alarm list
bay	local-remote, bay	Process picture
circuit-breaker	common, power-switch, circuit-breaker	Process picture
Table continues on next page		

© 2021 Hitachi Power Grids. All rights reserved.

Control dialog	Content	Used in
diameter	local-remote, diameter	Process picture
disconnector	common, power-switch, disconnector	Process picture
double-sided-truck	common, power-switch, truck	Process picture
event	event	Event list
load-breaker-witch	common, power-switch, load-break-switch	Process picture
measurement	measurement	Process picture
section	local-remove, section	Process picture
tap-changer	common, tap-changer	Process picture

11.3 Context aware content

With Workplace X, it is possible to have content that follows the context of another window. In a follower window, navigation menu contains only content that is marked as linkable. Example 87 contains a menu item definition that sets the *linkable* property as true.

Menu item with linkable content

Example 87

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="main-menu-item" />
        <require right="ALARMS" />

        <variable type="external" name="menuItem" />

        <script type="text/javascript"><![CDATA[
            this.set("menuItem", {
                "type": "queryFilters",
                "order": 300,
                "id": "Alarms",
                "icon": "webui_icon_16/ui_alarm_bell",
                "title": i18n._("Alarms"),
                "path": "alarms/list",
                "filterKey": "WebUI_Alarms_Filters",
                "linkable": true
            });
        ]]></script>
    </head>
</document>
```

This allows follower windows to access the menu item. To filter the content, use the window variable called *global_coordinated_window_content needs*. The content of the variable is an object with two fields **scope** and **title**. **Scope** contains the object reference that defines the object types that should be shown in the linked view. **Title** contains the title of the view that defines the context. Typically, this is the name of a process picture. Example 88 is a trivial view that shows the content of the variable. This variable can be used to construct the context-sensitive display.

Trivial example that shows the content of the window content linking to the variable

Example 88

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <variable type="window" name="global_coordinated_window_content"
```


© 2021 Hitachi Power Grids. All rights reserved.

```
default="{}" />
    </head>
    <body>
        Content linked to: {global_coordinated_window_content.scope}
    </body>
</document>
```

11.4 Known metadata

The predefined metadata is listed below.

sideViewName

- When a view is loaded to a tab, if the view has *sideViewName* defined, the view is loaded to the side view. Click **Control** tab on above the left menu to access the side view. The sideview is opened with a single attribute, *localviewid*, which contains the id of the opened view. Example 89 contains a view defining the *sldViewName*. The view defines an addition internal variable which is then used by the side view. The side view fetches a reference to the main view using the *localviewid* variable.

  View with sideViewName

Example 89

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="process" />
        <meta name="sideViewName" content="my-side-view" />
        <variable type="internal" name="message" default="'Hello
world!'" />
    </head>
    <d:body
        xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns="http://www.w3.org/1999/xhtml">
        <p>Hello</p>
    </d:body>
</document>
```

Example 90 contains the corresponding side view.

Simple side view

Example 90

```
my-side-view.xml:
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <variable type="external" name="localviewid" />
    <script type="client">
       self.m = () => {
         const id = self.get("localviewid");
         const message =
runtime.getView(id).getBinder().get("message");
         alert(message);
       };
    </script>
  </head>
  <body>
    <button>Click me!
      <action type="js-invoke" name="m" />
    </button>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
    </body>
</document>
```

quickFilterPlaceholder

- Defines the placeholder text available in the search field.

contentViewVariables

- Defines the variables that are synchronized between the main view and a control dialog.

11.5 Extension points

The predefined extension points that can be used for customization are listed below:

Table 47: Extension points

Extension point	View	Description
x-control-header-icon	control	This extension point can be used for displaying icons in control dialogs. All icons are available in all the device types, so an implementation for this extension decides the icons that can be displayed. Example 91 displays the implementation of the auto-reclose icon. Note that in this example, the parameter value of the icon is not correct. The value should be a character in the Unicode private use section.
product-logo	desktop-layout	Possibility to add a custom logo next to the ABB logo in the top left corner. The ABB logo can also be disabled with **x-show-default-product-logo** option.
x-top-menu-header	top-menu	Content can be added to the top of the top menu. Example 92 displays the process to use this attribute.
x-top-menu-items	top-menu	More menu items can be added to the top menu. See Section 11.1 for this extension.

Table 48: Extension options

Option	View	Description
System has the following extension options that can be turned off.		
x-store-session	top-menu	Storing sessions from the top menu can be disabled.
x-user-role-change	top-menu	The role change functionality from the top menu can be disabled.
x-user-password-change	top-menu	The password change functionality from the top menu can be disabled.
x-show-default-product-logo	desktop-layout	The ABB logo from the top left corner of the screen can be disabled.

Implementing a control dialog icon

Example 91

```
<?xml version="1.0"?>
xmlns="http://www.w3.org/1999/xhtml"
<x:extension xmlns:x="http://www.webui-ap.com/schemata/extension/1.0"
  xmlns:d="http://www.webui-ap.com/schemata/view/1.0">
  <x:apply mask="control/header-section" />
  <x:implementation name="x-control-header-icon">
    <d:if condition="(`cda:autoreclosing_state`.value = 2)">
```


© 2021 Hitachi Power Grids. All rights reserved.

```
      <div class="status-icon" style="order:30">
        <d:view name="Emblem" xmlns="http://www.w3.org/1999/xhtml">
          <d:parameter name="disableZooming" value="true" />
          <d:parameter name="rotation" value="0" />
          <d:parameter name="color"
            value="logicalColors['Emblem (default)'].Value" />
          <d:parameter name="icon" value="'?'" />
          <d:parameter name="iconVerticalOffset" value="1.5" />
          <d:parameter name="blink"
value="`cda:autoreclosing_state`.value = 2" />
          <d:parameter name="emblemAddonColor" value="'rgba(255, 255,
255, 0)'" />
          <d:parameter name="emblemAddonText" value="''" />
          <d:parameter name="emblemAddonMultiSignal" value="false" />
        </d:view>
      </div>
    </d:if>
  </x:implementation>
</x:extension>
```

Using x-top-menu-header extension point

Example 92

```
<?xml version="1.0"?>
<x:extension xmlns:x="http://www.webui-ap.com/schemata/extension/1.0">
    <x:apply mask="top-menu" />
    <x:implementation name="x-top-menu-header">
        <div xmlns="http://www.w3.org/1999/xhtml"
            xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
            s:color="white">
            Hello world!
        </div>
    </x:implementation>
</x:extension>
```


© 2021 Hitachi Power Grids. All rights reserved.

Section 12 Troubleshooting

This section describes error situations and debugging views.

12.1 Typical problem situations

The namespaces of the views can be incorrect. If HTML elements are not in the XHTML namespace or if the document is in the XHTML namespace, the page remains blank. See Example 94 and Example 98.

Invalid view with body in the HTML namespace

Example 93

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
    xmlns="http://www.w3.org/1999/xhtml">
    <p>Hello world!</p>
  </body>
</document>
```

Invalid document with p element in the document namespace

Example 94

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <body>
    <p>Hello world!</p>
  </body>
</document>
```

One possibility is that the view is not well-formed. Check this with an XML syntax checker. For example, a script block containing less than the character is not inside a CDATA section. See Example 95 for such document.

Non well formed view document

Example 95

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <script type=”client”>
       for (let i = 0; i < 5; i++) alert(“Hello”);
    </script>
  </body>
</document>
```

Another possibility is that the initialization fails. In most cases, a red error banner is shown. But in certain cases, the error message is only visible in the console of the browser. Press F12 to access the console.

Script are not executed

If the type attribute of a script element is missing or incorrect, scripts are not executed. Also, one possibility is that the script element is executed, but the behavior is not expected. To

© 2021 Hitachi Power Grids. All rights reserved.

verify this, add a debugger statement in the top of the script block and over the view with JavaScript debugger enabled.

ParentElement, offsetTop, or SVG length attributes missing

When the script block is executed, the view is not placed in the visible DOM tree. Certain attributes, such as attributes related to the physical width of the element are available only after the element is attached to the visible DOM tree. Example 96 shows that a block of the script can be executed when an element is attached.

Running a JavaScript function after the root node is attached to the visible DOM tree

Example 96

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <script type=”client”>
       let n = view.getRootNode();
    console.log(“1: “ + n.parentElement);
    runtime.getComponentContext().event.attachInsertedEvent(n, self, ()
=> {
         console.log(“2: “ + n.parentElement);
       });
    </script>
  </head>
</document>
```

Memory consumption of the browser increase

It is possible to create memory leaks. For example, binding a DOM event manually or following a variable can cause a memory leak. Events created manually need to be unregistered in a function executed by the onclose event.

Removing an event listened when view is closed

Example 97

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
  <head>
    <script type=”client”>
      function doSomething() {}

      window.addEventListener(“deviceorientation”, doSomething);

      view.onclose.add(function() {
        window.removeEventListener(“deviceorientation”, doSomething);
      });
    </script>
  </head>
</document>
```

12.2 Debugging tools

There are a few debugging tools that can be used to troubleshoot the problems. When a view is opened with a regular browser, it is possible to use developer tools to debug the view. Press F12 in any browser to access the debugging tool.

© 2021 Hitachi Power Grids. All rights reserved.

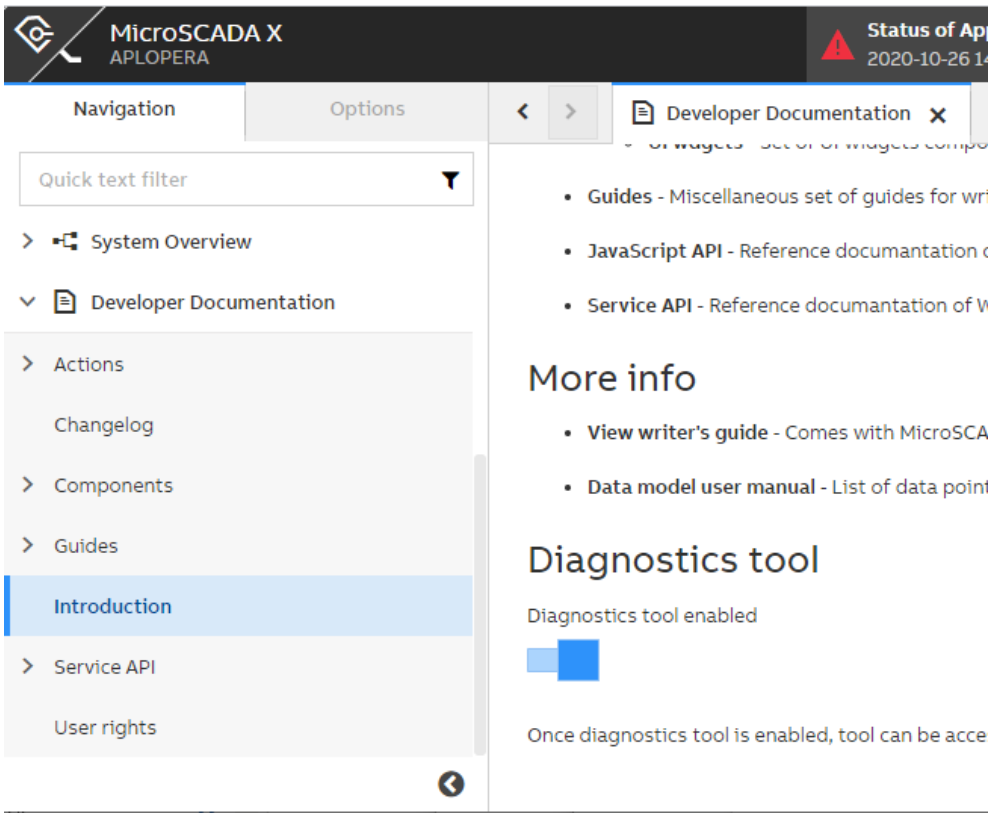


Figure 29: Accessing view diagnostics tool

Workplace X contains a view diagnostics tool. This can be accessed from the developer documentation's introduction page. There is a toggle for enabling and disabling the view diagnostics tool. Once enabled, the diagnostics tool can be accessed by clicking the icon in the top-left corner.

Diagnostics tool opens on the left side of the screen. Click the arrow icon to move the diagnostics tool to the right side of the screen.

Diagnostic tool lists all the currently opened views. Views are diagnosed either by clicking the view list or using the bull's eye tool. When the bull's eye tool is clicked, it is possible to click any element in the screen to see details of the view that owns the respective element.

View list can be filtered using the text input field. Different view roles are represented by the colored bars. All views with the same role will have a bar with the same color in front of the view name.

When a view is opened in the diagnostics tool, it is possible to inspect and edit variable content. If the value of the variable changes while the view is inspected, the change is highlighted. Note that only content that have a JSON representation can be edited. For example, it is not possible to edit a variable that contains JavaScript's date object.

© 2021 Hitachi Power Grids. All rights reserved.

Section 13 Examples

This section contains some examples which combine the techniques that are shown in the manual.

13.1 Screen splitter

This example implements a new feature to the system of opening a control dialog. When the user clicks a station level picture, the bay level picture automatically opens up alongside the station level picture as shown in Figure 30.

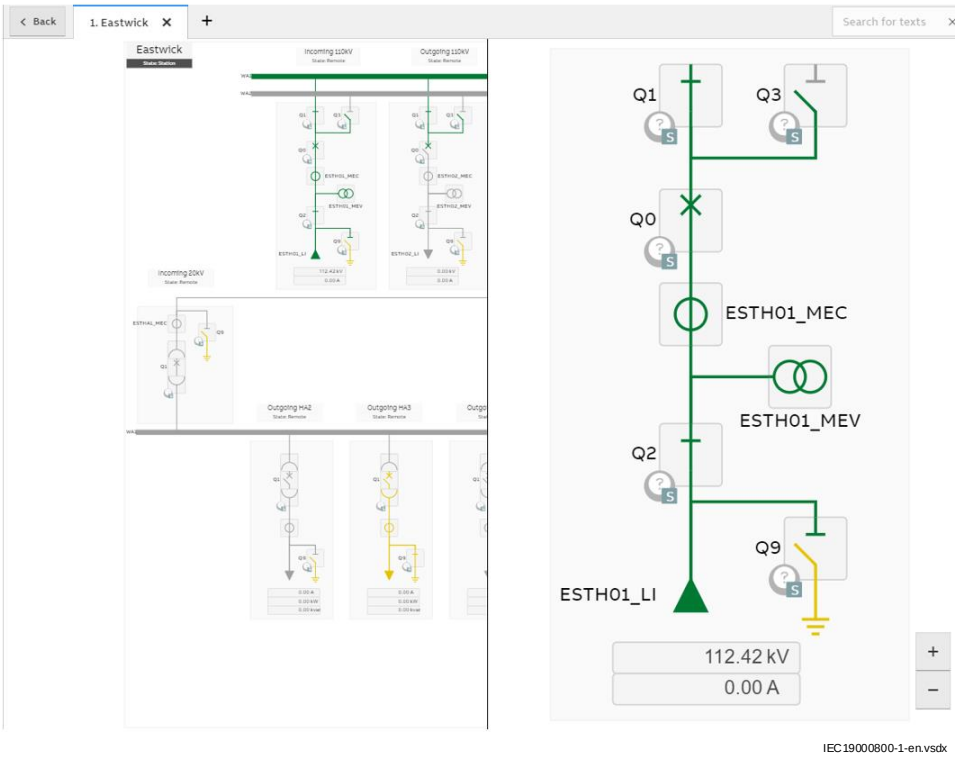


Figure 30: Station and bay Pictures

The full source is shown in Example 98. Line numbers are not part of the file. The feature is implemented as an extension. The extension applies only to the Bay view, as defined in line 4. The bay view is the grey rectangle behind the symbols in the bay. The extension adds a single JavaScript block to the head section of that view. Perform this using the head extension in lines 5 to 36.

The JavaScript contains two parts: a function called *handleClick* (lines 7 to 29) and a part that installs the handler (lines 31 to 34). *HandleClick* first detects the name of the view that contains the Bay symbol (line 8), then it checks the depth of view structure (lines 10 to 19). If the structure is not deep enough, the user can open a Bay level picture. There is also a detector, if it is already in a split bay picture (line 16). In that case, the extension does not do anything.

© 2021 Hitachi Power Grids. All rights reserved.

When a display needs to be split, the condition in the if statement in line 21 evaluates to true. In line 22, the click event propagation is canceled. This ensures that the symbol that was clicked never receives the event. In lines 24 to 26, a new element is created and placed under the DOM node of the root of the bottom-most view. In line 27, the bay display is instantiated to the newly created DOM node.

Screen splitting example

Example 98

```
 1: <?xml version="1.0"?>
 2: <x:extension xmlns="http://www.webui-ap.com/schemata/view/1.0"
 3:   xmlns:x="http://www.webui-ap.com/schemata/extension/1.0">
 4:   <x:apply mask="Bay" />
 5:   <x:head>
 6:     <script type="client">//<![CDATA[
 7:       function handleClick(e) {
 8:         let bayName = view.getParentView().getType().getName();
 9:
10:         let levels = 0;
11:         let p = view;
12:         while (p.getParentView()) {
13:           levels++;
14:           p = p.getParentView();
15:
16:           if (p.getRootNode(true).parentElement
                    .classList.contains("my-bay-view")) {
17:             return;
18:           }
19:         }
20:
21:         if (levels > 1) {
22:           e.stopPropagation();
23:
24:           let n = p.getRootNode(true);
25:           let c = UI.createElements(["div", {"class": "my-bay-view",
"style":
     "background: white; top: 0;right: 0;position: absolute;bottom:
0;width:
     50%;border-left: 1px solid black;"}]);
26:            n.appendChild(c);
27:           runtime.showDialog(view, bayName, {}, c);
28:         }
29:       }
30:
31:       runtime.getComponentContext()
              .events.attachInsertedEvent(view.getRootNode(), self, ()
=> {
32:         let p = view.getRootNode();
33:         p.parentElement.parentElement
                .addEventListener("click", handleClick, true);
34:       });
35:     //]]></script>
36:   </x:head>
37: </x:extension>
```

User can call the *handleClick* when needed. This is performed in lines 31 to 34. Line 31 uses a mechanism for calling a method when a DOM node is attached to the visible DOM tree. This is needed to ensure that the element has parent elements defined. Lines 32 to 34 executed when view *getRootNode()* is attached. In line 33, the *handleClick* function is attached as an event handler for the click event. It uses the capture phase of the DOM event model so that when the user clicks a symbol inside element, the symbol does not receive the click.

© 2021 Hitachi Power Grids. All rights reserved.

13.2 Power flow visualization

Example 99 adds a visualization of power flow in the power process. Example 99 shows the steps to add a third-party component, and use subscription and process information with an external library.

The visualization uses Sankey plugin for D3 library. The D3 library is part of the basic system, the Sankey plugin is not part of the basic system. The plugin can be downloaded from **Unpkg** (https://unpkg.com/d3-sankey@0.12.3/dist/d3-sankey.js).

The provided link points to a public source code repository. There is no guarantee that the link is valid in the future. Also, be aware of the license terms of any third-party components, both closed and open source ones. This component is distributed under the BSD license, which requires a certain obligation to be met.

The file can be stored in js folder, for example in \sc\apl\<myapplication>\views\js. This JavaScript file is loaded automatically.

To create a visualization, a view is needed. Since the use library is a JavaScript library, the view consists of JavaScript. Example 99 creates a trivial view with hard-coded data. The result is shown in Figure 31.

Power flow with hardcoded data

Example 99

```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
    xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
    <d:head>
        <d:title>Power Flow</d:title>
        <d:style>path.link { stroke: #ddd; } </d:style>
        <d:role name="process" />
        <d:script type="client">//<![CDATA[
            const svg = d3.select(view.getElementById("c"));
            const data = {
                nodes: [
                    {id: 0, title: "Total", color: "#77f"},
                    {id: 1, title: "Rivers", color: "#7f7"},
                    {id: 2, title: "Herwood", color: "#f77"},
                    {id: 3, title: "Eastwick", color: "#ff7"},
                ],
                links: [
                    {source: 0, target: 1, value: 3},
                    {source: 0, target: 2, value: 1},
                    {source: 0, target: 3, value: 1}
                ]
            };

            const width = 1000;
            const height = 500;

  const sankey =
d3.sankey().nodeAlign(d3.sankeyJustify).nodeWidth(15).nodePadding(10).siz
e([width, height]).nodes(data.nodes).links(data.links);
  const graph = sankey();

  svg.append("g")
      .attr("stroke", "#000")
    .selectAll("rect")
    .data(data.nodes).enter()
    .append("rect")
```


© 2021 Hitachi Power Grids. All rights reserved.

```
      .attr("x", d => d.x0)
      .attr("y", d => d.y0)
      .attr("height", d => d.y1 - d.y0)
      .attr("width", d => d.x1 - d.x0)
      .attr("fill", d => d.color)
        .append("title").text(d => `${d.title}\n${d.value} kW`);

  const link = svg.append("g")
      .attr("fill", "none")
      .attr("stroke-opacity", 0.5)
    .selectAll("g")
    .data(data.links).enter()
    .append("g");

  link.append("path")
      .attr("d", d3.sankeyLinkHorizontal())
      .attr("stroke", d => "#aaa")
      .attr("stroke-width", d => Math.max(1, d.width));

  link.append("title").text(d => `${d.source.title} â†’ ${d.target.title}
\n${d.value} kW`);

  svg.append("g")
      .style("font", "10px sans-serif")
    .selectAll("text")
    .data(data.nodes).enter()
    .append("text")
      .attr("x", d => d.x0 < width / 2 ? d.x1 + 6 : d.x0 - 6)
      .attr("y", d => (d.y1 + d.y0) / 2)
      .attr("dy", "0.35em")
      .attr("text-anchor", d => d.x0 < width / 2 ? "start" : "end")
      .text(d => d.title);
 //]]></d:script>

    </d:head>
    <d:body y:padding="2em" xmlns="http://www.w3.org/1999/xhtml"
        xmlns:s="http://www.w3.org/2000/svg">
        <h1>Power Flow</h1>
        <s:svg width="100%" height="100%" id="c" />
    </d:body>
</d:document>
```

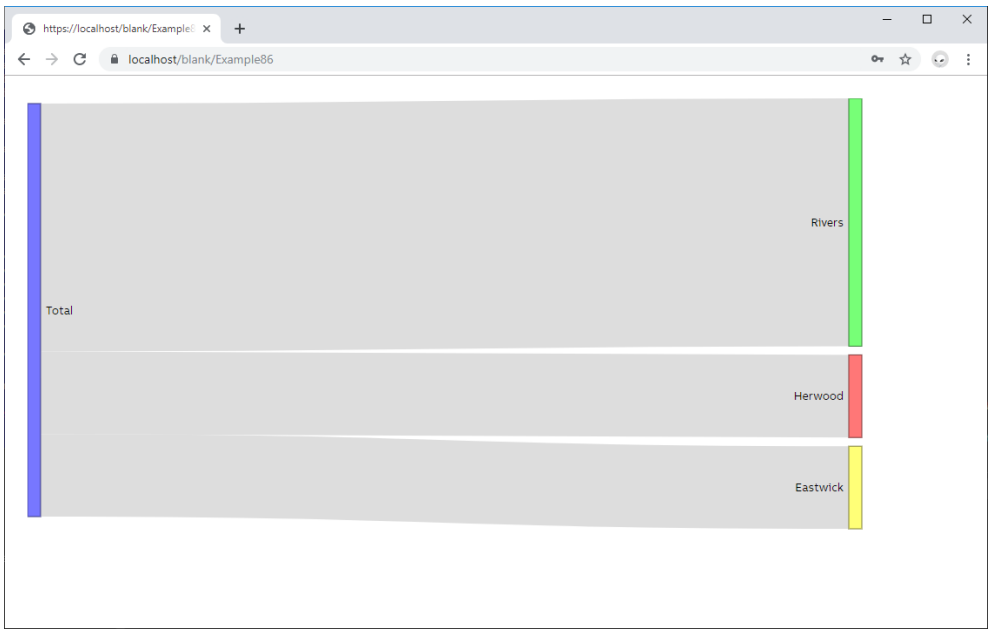


Figure 31: Trivial view with hard coded data

© 2021 Hitachi Power Grids. All rights reserved.

This can be connected to process data using *bindMultiple* function call. This version is available in Example 100.

Power Flow Diagram with Process Data

Example 100

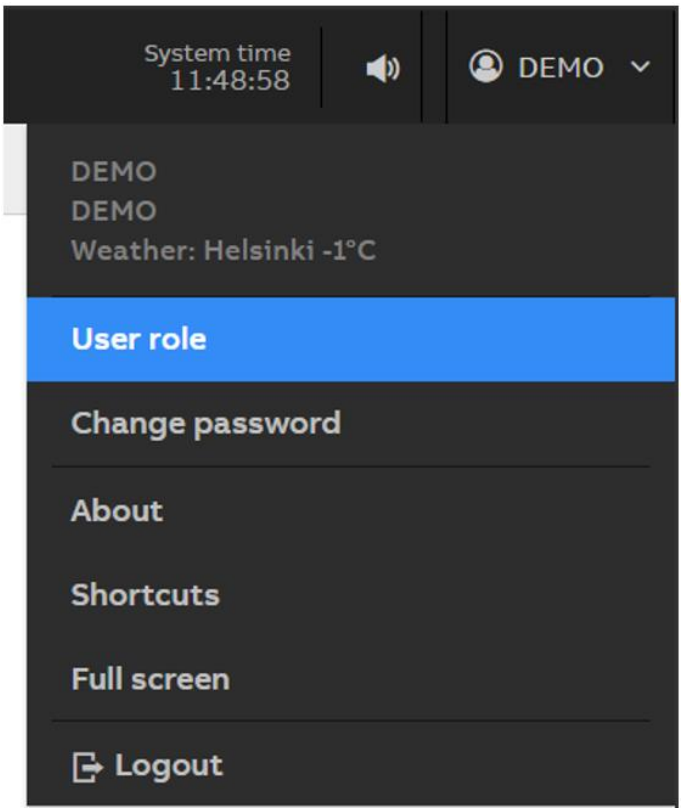
```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
  <d:head>
    …
    <d:script type="client">
        const svg = d3.select(view.getElementById("c"));

        view.bindMultipleWithPrevResults(["`\\P\\ESTH01_MEC\\20:OV`",
            "`\\P\\ESTH02_MEC\\20:OV`", "`\\P\\ESTH03_MEC\\20:OV`"],
values => {
            const data = {
                nodes: [
                    {node: 0, name: "Total", color: "#77f"},
                    {node: 1, name: "Rivers", color: "#7f7"},
                    {node: 2, name: "Herwood", color: "#f77"},
                    {node: 3, name: "Eastwick", color: "#ff7"},
                ],
                links: [
                    {source: 0, target: 1, value: values[0]},
                    {source: 0, target: 2, value: values[1]},
                    {source: 0, target: 3, value: values[2]}
                ]
            };
            const data = {
              nodes: [
                {node: 0, name: "Total", color: "#77f"},
                {node: 1, name: "Rivers", color: "#7f7"},
                {node: 2, name: "Herwood", color: "#f77"},
                {node: 3, name: "Eastwick", color: "#ff7"},
              ],
              links: [
                {source: 0, target: 1, value: 3},
                {source: 0, target: 2, value: 1},
                {source: 0, target: 3, value: 1}
              ]
          };

          const sankey = d3.sankey().nodeWidth(15).nodePadding(10)
            .size([959, 494]);
          …
       });
    </d:script>
  </d:head>
  <d:body y:padding="2em" xmlns:s="http://www.w3.org/2000/svg">
    <s:svg width="100%" height="100%" id="c" />
  </d:body>
</d:document>
```

13.3 Weather widget

This example adds a simple **weather widget** to the top menu. The result is shown in Figure 32.

This example uses a third-party system to get the weather forecast which may not work. This tutorial does not give the advice to use or not to use this particular system or any other third-party systems. The example provides information on the integration performed.

© 2021 Hitachi Power Grids. All rights reserved.



IEC19000801-1-en.vsdx

Figure 32: Top menu of the UI with the weather information

Figure 32 uses an extension mechanism and uses an extension point provided by the `top-menu`. This extension applies only to the `top-menu` view. That is declared on line 6. Lines from 7 to 9 shows the implementation to the extension point in the `top-menu`. A simple span element is added with a text. It assumes that the weather is stored in a variable.

Simple weather widget

Example 101

```
 1: <?xml version="1.0"?>
 2: <x:extension
 3:   xmlns:d="http://www.webui-ap.com/schemata/view/1.0”
 4:   xmlns=”http://www.w3.org/1999/xhtml”
 5:   xmlns:x="http://www.webui-ap.com/schemata/extension/1.0"
 6:   <x:apply mask="top-menu" />
 7:   <x:implementation name="x-top-menu-header">
 8:       <span style="display: block;">Weather: { weather }</span>
 9:   </x:implementation>
10:   <x:head>
11:      <d:variable type="internal" name="weather" default="'" />
12:
13:       <d:script type="text/javascript">//<![CDATA]
14:                          fetch("https://query.yahoopis.com/vi/public/
yql?q=select%20from%20weather.forecase%20where%20woeid%20in%20(select
%20woeid%20from%20qeo.places(a)%20text%3D%22helsinki%22)%20and%20u
%3D'c'&format=json&evn=store%3A%                                  2F
%2Fdatatables.org%2Falltableswithkeys")
15:          .then(function(response) {
16:            return response.json();
17:          })
18:          .then(function(json) {
19:             var str = json.query.results.channel.location.city + " "
```


© 2021 Hitachi Power Grids. All rights reserved.

```
+;
                  json.query.results.channel.item.condition.temp + "°" +;
                  json.query.results.channel.units.temperature;
20:             view.set("weather", str);
21:           });
22:     //]]></d:script>
23:   </x:head>
24: </x:extension>
```

The needed variable is declared in a head extension in line 11. The line from 13 to 22 contains a JavaScript block that fetches the weather from the server https://query.yahooapis.com/ and formats it to a string, which is then at line 20 stored to the variable. The used third-party system requires a long query parameter.

13.4 Dashboard

This example provides a simple dashboard view. The dashboard contains gauges and measurement values. The example shows how to use widgets. First, a new entry to the main menu is created for dashboards. Example 102 shows the view. This adds all views with role dashboard in the menu structure as explained in section 11.1.

Dashboard menu item view

Example 102

```
<?xml version="1.0"?>
   <document xmlns="http://www.webui-ap.com/schemata/view/1.0">
   <head>
       <role name="main-menu-item" />

       <variable type="external" name="menuItem" />

       <script type="text/javascript">//<![CDATA[
           this.set("menuItem", {
               "type": "queryViews",
               "order": 50,
               "id": "dashboards",
               "icon": "abb/ui_dashboard2",
               "title": i18n._("Dashboards")
           });
       //]]></script>
   </head>
</document>
```

Next, create an actual dashboard. This dashboard contains some widgets and some HTML markup which are connected to back-end data. As an example, use the measure widget. The widget has several attributes which can be hardcoded or connected to a data. Example 103 contains a trivial dashboard with hardcoded data.

Initial version of the dashboard

Example 103

```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
  <d:head>
    <d:title>Eastwick Power</d:title>
    <d:role name="dashboards" />
  </d:head>
  <d:body y:padding="2em">
    <d:measure type="radial"
      value="50" min="0" max="110"
      low-alarm-limit="0" low-warning-limit="0"
```


© 2021 Hitachi Power Grids. All rights reserved.

```
      high-warning-limit="80" high-alarm-limit="100"
      label="Q1 Active Power" unit="A" />
  </d:body>
</d:document>
```

The dashboard is not helpful since it contains just hard-coded data. But this data can be connected from the process. Example 104 contains a view which subscribes data from the back-end. The data uses SCIL syntax and naming namespace (see Section 5.4).

Measurement connected to the backend

Example 104

```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
  xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
  <d:head>
    <d:title>Eastwick Power</d:title>
    <d:role name="dashboards" />
  </d:head>
  <d:body y:padding="2em" xmlns:h="http://www.w3.org/1999/xhtml"
    xmlns:n="http://www.webui-ap.com/schemata/naming/1.0">
    <h:div n:LN="ESTH01_MEC" n:IX="20">
      <h:h3>{IL.STA}</h:h3>
      <d:measure type="radial"
        value="{OV}" min="{LI - 10}" max="{HI + 10}"
        low-alarm-limit="{LI}" low-warning-limit="{LW}"
        high-warning-limit="{HW}" high-alarm-limit="{HI}"
        label="{OX}" unit="{ST}" />
    </h:div>
  </d:body>
</d:document>
```

This example contains one measurement. It is easy to extend the example to include an arbitrary number of measurements and labels. Use repeat elements for multiplying the data content, based on the data as shown in Example 105. The end result is shown in Figure 33.

Dashboard with 9 measurements

Example 105

```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
  xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
  <d:head>
    <d:title>Eastwick Power</d:title>
    <d:role name="dashboards" />
  </d:head>
  <d:body y:padding="2em" xmlns:h="http://www.w3.org/1999/xhtml"
    xmlns:n="http://www.webui-ap.com/schemata/naming/1.0">
    <h:h1>Eastwick Power Dashboard</h:h1>
    <h:div y:display="flex" y:flex-direction="row">
      <d:repeat for-each="['ESTH01_MEC', 'ESTH02_MEC', 'ESTH03_MEC']"
iterator="ln">
        <h:div n:LN="{ln}" n:IX="10" y:display="inline-block"
y:padding="1em">
          <h:h3>{IL.BAY}</h:h3>
          <d:repeat for-each="[10, 20, 21]" iterator="ix">
            <d:measure n:IX="{ix}" type="radial"
              value="{OV}" min="{LI - 10}" max="{HI + 10}"
              low-alarm-limit="{LI}" low-warning-limit="{LW}"
              high-warning-limit="{HW}" high-alarm-limit="{HI}"
              label="{OX}" unit="{ST}" />
          </d:repeat>
        </h:div>
      </d:repeat>
    </h:div>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
  </d:body>
</d:document>
```

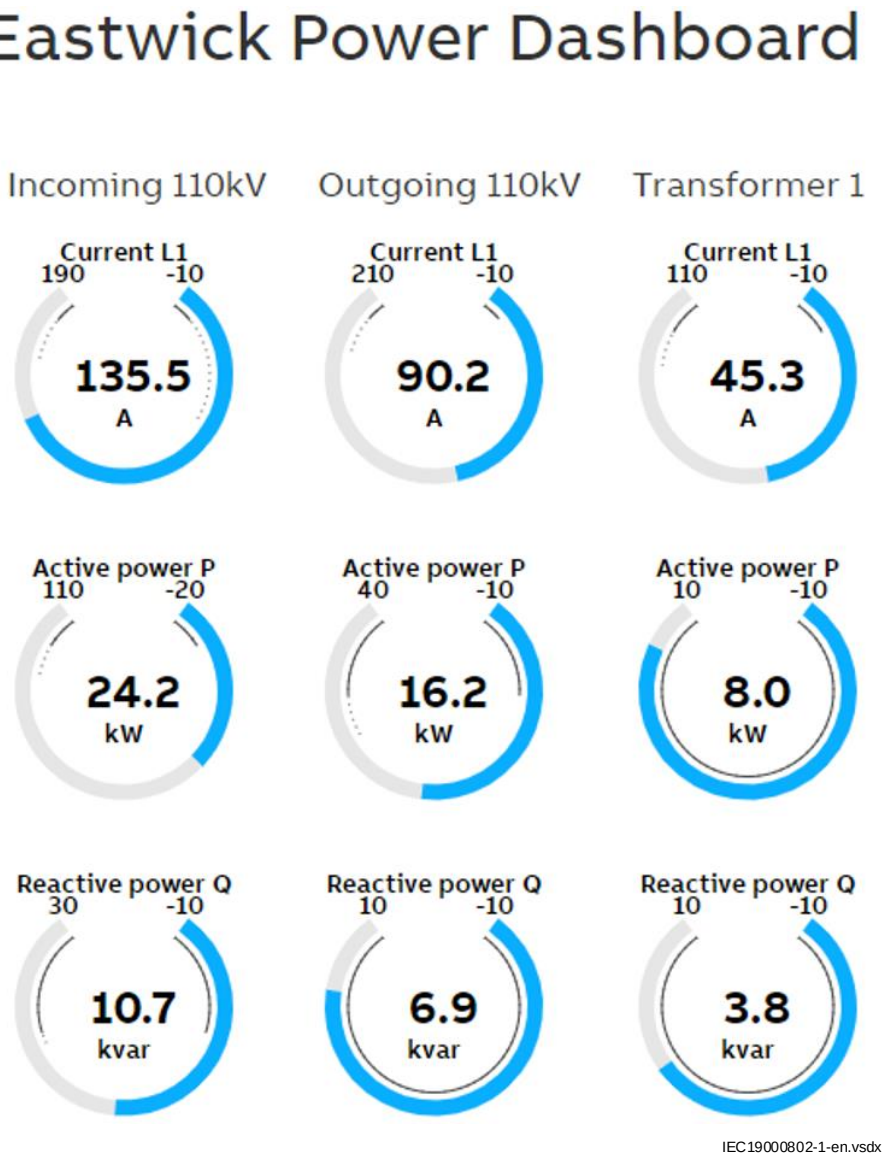


Figure 33: Power dashbaord with 9 measurements

13.5 System object editor

This example shows SCIL functions can be called from view files. A trivial SYS600 system object editor view is created. Figure 34 shows the result.

© 2021 Hitachi Power Grids. All rights reserved.

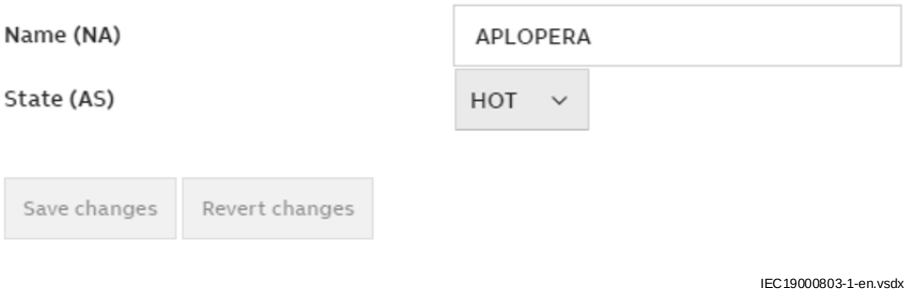


Figure 34: System object editor

Example 106 is the view of the system object editor. For example, the view can only be used for editing NA attribute and AS attribute of the application object, but the same mechanism can be used for arbitrary many attributes.

The view has one external attribute, the number of the application being edited. This is used in two scripts:

- One script is used for fetching the objects
- One for editing the application object

Variables NA and AS are bound to edit and select components. These components have an action attached to the change event. This event is fired when the content of the component is changed. These actions modify stores values in an internal variable called *changed*, which contains all attributes that the user has changed.

There are two buttons. Saving invokes the server script for storing the changes; clear the object for changed attributes; revering fetches the changes again and clears the object for changed attributes. Both buttons are enabled only if the object for changed attributes is filled.

System Object Editor

Example 106

```
<?xml version="1.0" encoding="utf-8"?>
<d:document xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:y="http://www.webui-ap.com/schemata/styling/1.0">
  <d:head>
    <d:title>System Object Editor</d:title>
    <d:variable type="external" name="AN" default="1" />
    <d:variable type="internal" name="InitialCX">
      <d:action type="set" name="CX" value="InitialCX" />
    </d:variable>
    <d:variable type="internal" name="InitialAS">
      <d:action type="set" name="AS" value="InitialAS" />
    </d:variable>
    <d:variable type="internal" name="CX" />
    <d:variable type="internal" name="AS" />
    <d:variable type="internal" name="changes" default="{}" />
    <d:script name="get-apl-attributes" type="server"
result="{CX:InitialCX, AS:InitialAS}" init="true" authorize="GENERAL >=
2">
        #return LIST(CX=APL{AN}:BCX, AS=APL{AN}:BAS)
    </d:script>
    <d:script name="set-apl-attributes" type="server" authorize="GENERAL
>= 2">
        @a = {changes}
        @an = {AN}
        @attrs = LIST_ATTR(%a)
        #loop_with i = 1 .. LENGTH(%attrs)
          @attr = %attrs(%i)
```


© 2021 Hitachi Power Grids. All rights reserved.

```
          #set APL'an':B'attr'=%a.'attr'
        #loop_end
    </d:script>
  </d:head>
  <d:body y:padding="2em" xmlns="http://www.w3.org/1999/xhtml">
    <table y:marginBottom="2em">
      <tr>
        <th y:width="20em">Comment (CX)</th>
        <td y:width="20em">
          <d:edit variable="CX">
            <d:action event="change" type="set" name="changes['CX']"
value="CX" when="CX != InitialCX" />
          </d:edit>
        </td>
      </tr>
      <tr>
        <th>State (AS)</th>
        <td>
          <d:select variable="AS" allow-clear="false">
            <d:option value="HOT" />
            <d:option value="WARM" />
            <d:option value="COLD" />
            <d:action event="change" type="set" name="changes['AS']"
value="AS" when="AS != InitialAS" />
          </d:select>
        </td>
      </tr>
    </table>
    <d:button disabled="{empty(changes)}">Save changes
      <d:action type="invoke" name="set-apl-attributes" />
      <d:action type="set" name="changes" value="{}" />
    </d:button>
    <d:button disabled="{empty(changes)}">Revert changes
      <d:action type="invoke" name="get-apl-attributes" />
      <d:action type="set" name="changes" value="{}" />
    </d:button>
  </d:body>
</d:document>
```

This example shows how the basic principles of an editor can be done. Important aspects such as error handling are not shown in the example.

13.6 Signal List

This example shows how signal list can be created. Signal list will be context-aware and shows values of signals from a picture.

First part of the signal list is the main menu item. The instructions on the content is defined in Section 11.1. This menu item has property **linkable** set to **true**. This means the content is available in Workplace X follower window. The complete menu item definition is available in Example 107.

Menu item for signal list

Example 107

```
<?xml version="1.0"?>
<document xmlns="http://www.webui-ap.com/schemata/view/1.0">
    <head>
        <role name="main-menu-item" />

        <variable type="external" name="menuItem" />

        <script type="text/javascript">//<![CDATA[
            this.set("menuItem", {
```


© 2021 Hitachi Power Grids. All rights reserved.

```
                "type": "staticPath",
                "order": 450,
                "id": "Signals",
                "icon": "webui_icon_16/ui_list",
                "title": i18n._("Signals"),
                "path": "signals/list",
                "linkable": true
            });
        //]]></script>
    </head>
</document>
```

In this manual, the main signal list example is split into two parts. First part contains the head of the view and the second part contains the body of the view. Head part is available in Example 108 and body part is available in Example 109. To use the example, the view defined in the above examples needs to be called **list.xml** and needs to be in the folder called **signals**. The reason for this is the **path** of the menu item points to this view.

First view is defined to the metadata fields. **Icon** defines the icon to be used in the tab header. **quickFilterPlaceholder** defines the text to be used in the view search bar. The view search field is shown only if the metadata **quickFilterPlaceholder** is defined.

View has a variable called **objectReference**. This variable contains the object reference that is used for querying the content to the display. When the content of the variable changes, service method called **QueryDatapointsInformation** is invoked. Results of this service call is stored to variables called **objects**. This variable is used in the body part to build the user interface.

There are two ways to populate the **objectReference** variable. If the display is opened from a follower window and the user has selected an SLD from the master window, then the information of the selected SLD is available in the variable called **global_coordinated_window_content**. In signal list, an action is attached to this variable. Then the **global_coordinated_window_content** variable is set, and the scope is automatically copied to the **objectReference** variable. This can be done by adding an action element as a child of the variable element.

If the window linking is not used, then the user has a possibility to select the display from a drop-down list. This functionality uses **selection**, **selectionValues**, and **selectionLabels** variables. **Selection** contains a selected object reference. This is automatically propagated to **objectReference** variable using an action element. **SelectionLabels** and **selectionValues** variables are by the selection control. They are populated with a JavaScript code in the view. The JavaScript code will invoke a **queryViews** service and process the data.

Head part of the signal list

Example 108

```
<head>
    <meta name="icon" content="ui_list" />
    <meta name="quickFilterPlaceholder" content="Search" />

    <variable type="internal" name="objects" />

    <variable type="internal" name="objectReference">
        <action type="invoke-service" name="DatapointInformationService/
QueryDatapointsInformation" when="not empty(objectReference)">
            <parameter name="object_reference" value="objectReference +
'.*'" />
            <parameter name="include_attributes" value="true" />
        </action>
    </variable>

    <variable type="internal" name="selectionLabels" />
    <variable type="internal" name="selectionValues" />
    <variable type="internal" name="selection">
        <action type="set" name="objectReference"
```


© 2021 Hitachi Power Grids. All rights reserved.

```
value="selectionValues[selection]"
when="empty(global_coordinated_window_content.scope)" />
    </variable>

    <style>
        body {
            padding: 1em;
        }

        div.top {
            margin-bottom: 1em;
        }

        .empty-text {
            font-size: 28px;
            margin-top: 3em;
            width: 100%;
            text-align: center;
        }

        td.measurement {
            position: relative;
        }

        span.value {
            position: absolute;
            top: 50%;
            transform: translateY(-50%);
            padding-left: 1em;
        }

        h1 {
            margin-top: .25em;
            margin-botto: .75em;
        }

        tr.match {
            background: rgb(255, 247, 204) ! important;
            border-color: rgb(255, 216, 0) ! important;
        }

    </style>

    <service interface="WebServer" method="queryViews" />
    <service interface="DatapointInformationService"
method="QueryDatapointsInformation" result="{objects: objects}"/>

    <script type="client">
      view.invokeService("WebServer", "queryViews", { condition: { meta:
"scope", operation: 1, value: "" }, include_meta: true }, (response) => {
            self.set("selectionLabels", response.views.map(r => r.name));
            const values = {};
            response.views.forEach(v => values[v.name] = v.meta["scope"]);
            self.set("selectionValues", values);
        });
    </script>


    <variable type="window" name="global_coordinated_window_content"
default="{}">
        <action type="set" name="objectReference"
value="global_coordinated_window_content.scope"
when="global_coordinated_window_content.scope" />
    </variable>
</head>
```


© 2021 Hitachi Power Grids. All rights reserved.

Body part of the view first must be selected so that the user can select the display. **Select** component is used for this purpose. Component is only shown if no window linking scope is available. If window linking scope is used, then that information is used to show a title instead.

There are two checks for checking if the content is available. If not, a corresponding message is shown to the user.

Last section of the body is the table that contains all signals. Table has fixed columns, but rows are created by using a **repeat** element. All signals are not shown. Interesting signals are filtered using an **if** element. In this example all signals that have a path and customer text attributes are shown.

Table rows are defined inside the **if** statement. The first column contains datapoints path, the second column contains datapoints customer text, and the third column contains datapoints state. For the view uses, see Section 5.3. The datapoint references domain is bound in the **body** element using **cda-domain** attribute. The datapoint references name is bound in the **tr** element using **cda-name** attribute. The only object left for the actual reference is to define the attribute name.

The fourth column is defined only for measurements. This column contains a single **measure** component and the value of the datapoint. The datapoint reference is fully defined using the **cda-domain** and **cda-name** attributes. A single dot can be used in the reference to indicate that there is no information to specify in the data point name.

For each row, class **match** is added, and the row matches the view search and it uses the **class** element. The **class** element will add the defined class to the parent element. In this case, the class is added to the **tr** element if the condition in the **class** elements **when** attribute is fulfilled.

Body part of the signal list

Example 109

```
<d:body xmlns:d="http://www.webui-ap.com/schemata/view/1.0"
        xmlns:s="http://www.webui-ap.com/schemata/styling/1.0"
        xmlns:n="http://www.webui-ap.com/schemata/naming/1.0"
        xmlns="http://www.w3.org/1999/xhtml"
        n:cda-topic="Domain">

    <div class="top">
        <d:if condition="empty(global_coordinated_window_content.scope)">
            <d:select variable="selection" values="selectionLabels" />
        </d:if>
        <d:if condition="not
empty(global_coordinated_window_content.scope)">
            <h1>{global_coordinated_window_content.title}</h1>
        </d:if>
    </div>

    <d:if condition="empty(selectionLabels)">
        <div class="empty-text">Pictures not found. Create a picture with
ViewBuilder to view signals.</div>
    </d:if>

    <d:if condition="empty(objectReference) and not
empty(selectionLabels)">
        <div class="empty-text">Select an item from dropdown list to view
signals.</div>
    </d:if>

    <d:if condition="not empty(selectionLabels) and not
empty(objectReference)">
        <table class="table table-compact table-strong-header table-
striped table-borderee">
            <thead>
```


© 2021 Hitachi Power Grids. All rights reserved.

```
                <tr>
                    <th>Signal</th>
                    <th>Text</th>
                    <th>State</th>
                    <th s:width="25em" />
                </tr>
            </thead>
            <tbody>
                <d:repeat for-each="objects" iterator="o">
                    <d:if condition="'path' in o.attributes and
'customer_text' in o.attributes">
                        <tr n:cda-name="{o.object_reference}.
{o.datapoint_type}">
                            <d:class name="match" when="(fuzzyMatch(`cda:/
path`.value, global_quickFilter) or fuzzyMatch(`cda:/
customer_text`.value, global_quickFilter) or fuzzyMatch(`cda:/
state`.value, global_quickFilter)) and not empty(global_quickFilter)" />
                            <td>{`cda:/path`.value}</td>
                            <td>{`cda:/customer_text`.value}</td>
                            <td>
                                {`cda:/state`.value}
                            </td>
                            <td class="measurement">
                                <d:if condition="'high_2_limit' in
o.attributes">
                                    <d:measure s:display="inline-block"
s:width="15em" type="bar" value="{`cda:.`.value}"
                                        min="{`cda:/low_2_limit`.value}"
max="{`cda:/high_2_limit`.value}" show-min-max-values="false"
                                        low-alarm-limit="{`cda:/
low_2_limit`.value}" low-warning-limit="{`cda:/low_1_limit`.value}"
                                        high-warning-limit="{`cda:/
high_1_limit`.value}" high-alarm-limit="{`cda:/high_2_limit`.value}"
                                        normal-color="#0aaeff" warning-
color="#ffd800"
                                        alarm-color="#f03040" border-
color="transparent"
                                        background-color="#ebebeb" size-
class="1"
                                        show-value="false" decimals="2"
unit="" unit-color=""
                                        limit-marker-style="orthogonal"
bar-stripes="false"
                                        value-position="center"/>
                                    <span class="value">
                                        {`cda:.`.value}
                                        {`cda:.`.unit}
                                    </span>
                                </d:if>
                            </td>
                        </tr>
                    </d:if>
                </d:repeat>
            </tbody>
        </table>
    </d:if>
</d:body>
```


© 2021 Hitachi Power Grids. All rights reserved.

Hitachi ABB Power Grids
Grid Automation Products
PL 688
65101 Vaasa, Finland

https://hitachiabb-powergrids.com/microscadax

Scan this QR code to visit our website

1MRK 511 506-UEN

© 2021 Hitachi Power Grids.
All rights reserved.