TM671

Creating efficient mapp View HMI applications

PERFECTION IN AUTOMATION
A MEMBER OF THE ABB GROUP

Prerequisites and requirements

Training modules	TM611 – Working with mapp view
Software	The prerequisite for the execution of this training module is the mapp View project created in TM611. Automation Studio 4.4.4 Automation Runtime 4.44 Technology Package – mapp View 5.3
Hardware	ARsim X20 CPU with ETAL611.1T10-1 **www.br-automation.com/eta-system**

Table of contents

1 Introduction

mapp View includes a wide range of combinable and easy-to-configure standard components. The exercises in this training module are designed to deepen your existing basic knowledge of mapp View. The additional information, configuration options and combination of various widgets open up entirely new possibilities for designing an HMI application.

1.1 Learning objectives

This training module uses exercises to expand and reinforce your existing basic knowledge of mapp View.

- Participants will learn how to use the event and action system with different use cases.
- Participants will be able to apply the Automation Studio roles and rights system on widgets for visibility and operation.
- Participants will learn about procedures and options for the animation of widgets.
- Participants will be able to implement the different types of variables in the mapp View HMI application for the correct tasks.
- Participants will complete additional exercises related to using the text system.
- Participants will be able to create HMI applications for the appropriate task on different end devices.
- Participants will get an overview of the implementation of several HMI applications.
- Participants will learn how a machine process is represented visually.

1.2 Symbols and safety notices

Unless otherwise specified, the descriptions of symbols and safety notices listed in "TM210 – Working with Automation Studio" apply.

2 Dialog and message boxes

A dialog box and a message box are elements of an HMI application that can be displayed over a page.

2.1 Overview of dialog and message boxes

A dialog box is a window created by the developer of the HMI application that, similar to a page, is designed using a layout and the content referenced in its areas.

All widgets can be placed and configured in a piece of content that is used in the dialog box.

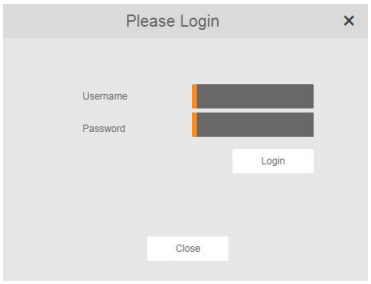


Figure 1: Example of a dialog box

Areas of application of dialog boxes

Dialog boxes are displayed to collect input from the user. Modal dialog boxes lock the rest of the user interface while the dialog box is being displayed. Modeless dialog boxes continue to allow interactions outside of the dialog box.

2.2 Creating and opening a dialog box

The goal of this exercise is to create a dialog box for user authentication in the HMI application. For the dialog box, a layout with an area is used which is assigned a piece of content with the size of the layout.

Exercise: Creating a dialog box

Using the following description, a dialog box should be created in which a "Login" widget and a "Button" widget for closing the dialog box are displayed. The dialog box is opened by clicking on the image in the header.

Figure 2: Open the dialog box by clicking on the image

The following steps are necessary for this:

1) Create a layout
2) Create content
3) Create a dialog box
4) Reference dialog boxes in the visualization object
5) Event binding for opening the dialog box

6) Event binding for closing the dialog box

7) Reference the event binding file in the HMI application.

HMI application / mapp View / Engineering / Dialog boxes

2.2.1 Creating a layout

The layout is added to the Logical View. The layout with the ID "DialogLayout" is configured with width="450" and height="300" in the open XML editor.

Property name	Value
id	DialogLayout
Height	300
width	450

Table 1: Properties of the layout file

An area the same size as the layout is configured.

Id	Height	width	left	top
AreaDialog	300	450	0	0

Table 2: Properties of an area

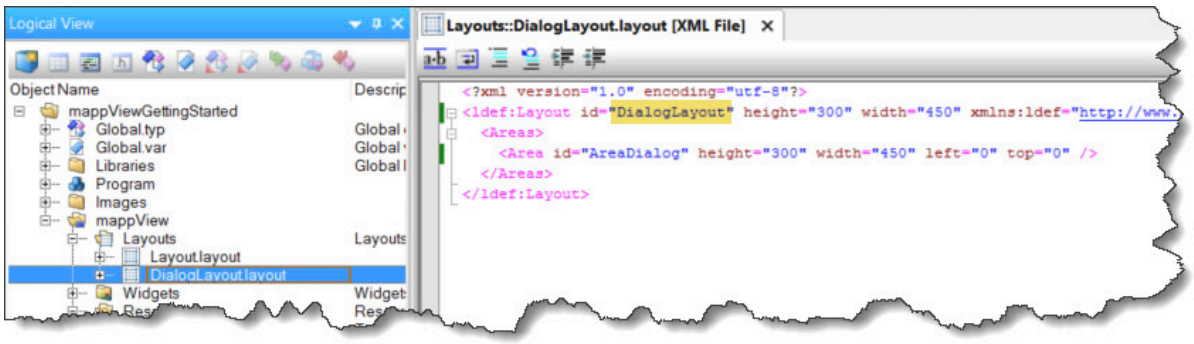


Figure 3: Add the layout file in Logical View and edit the XML file

2.2.2 Creating content

A piece of content is added in the package "Dialog boxes / AreaConents". After adding the content file in the Logical View, the name must be changed to "ContentLogin".

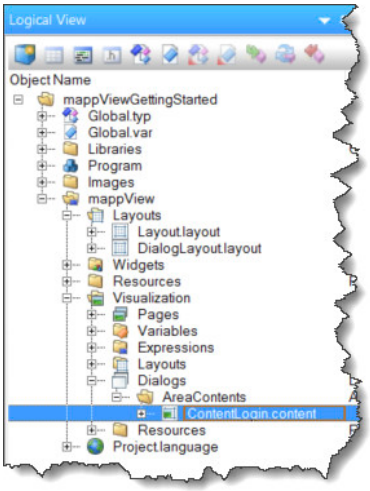


Figure 4: Content file added

Editing content

A piece of content is edited in the visual editor. In the Properties window, a unique name as well as the width and height of the content are entered.

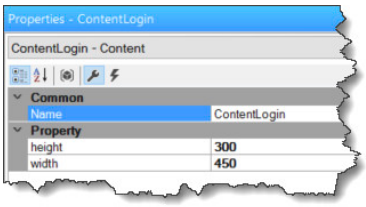


Figure 5: ContentLogin properties

A "Login" widget is in the content. The dialog box should also be closed automatically after a successful login.

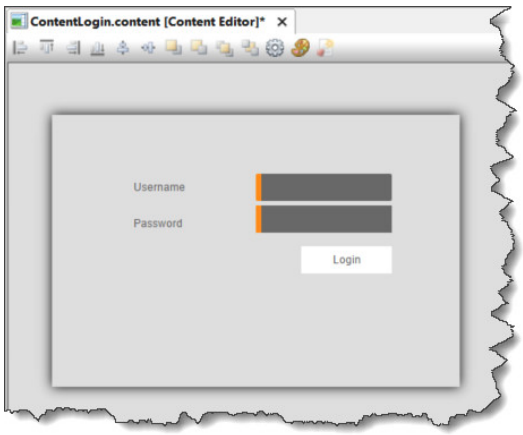
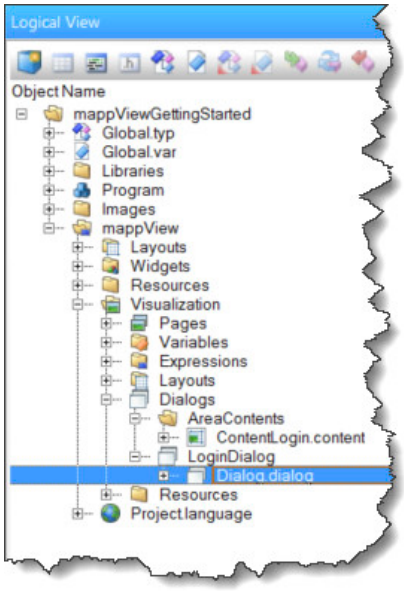


Figure 6: Widgets configured in the content

2.2.3 Creating a dialog box



A dialog box is added from the Object Catalog to the "Dialog" object in an existing "Dialogs" package using drag-and-drop or by double-clicking on it. The "Dialogs" package is part of a mapp View visualization package.

The next step is to create a dialog box with the name "LoginDialog" where a layout and a piece of content can then be referenced.

The following steps cannot be differentiated from the configuration of a page. Dialog boxes are managed in the "Dialogs" package.

Figure 7: Inserted dialog box

In order to uniquely identify the dialog box, a unique "ID" must be specified. To use the previously defined layout, the "ID" of the layout "DialogLayout" must be specified.

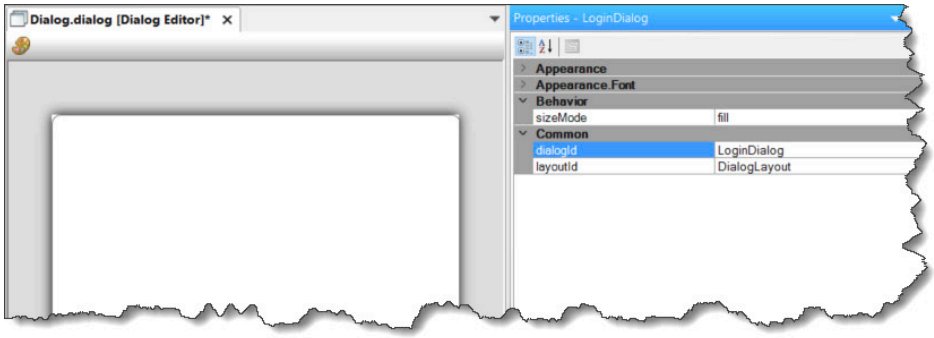


Figure 8: Dialog box editor with referenced layout and dialog box ID.

As is done with a page, the content with the ID "ContentLogin" must be assigned to the dialog box area.

2.2.4 Referencing a dialog box in the HMI application (.vis)

The dialog box must be referenced in the HMI application (.vis) of the Configuration View so that it is also transfered as a part of the HMI application.

The dialog boxes that are part of the HMI application are defined in the element <Dialogs>.

The image shows the relevant content for practicing with this visualization object (.vis).

```
<Dialogs>
    <Dialog refId="LoginDialog"/>
</Dialogs>
```

2.2.5 Event binding for opening the dialog box

In ContentTop, the name of the widget is changed to "ImageOpenDialog".

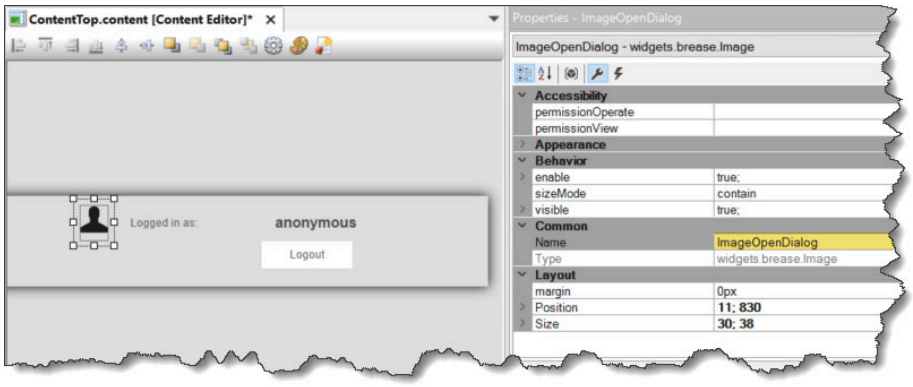


Figure 9: To open the dialog box, rename the "Image" widget.

In the next step, a new event binding for the click event is created in the event view for the "Image" Widget using the file assigned to the content.

The event is automatically recorded; the subsequent action must be defined. The possible attributes are described in Automation Help.

The entire event binding for opening a dialog box looks as follows:

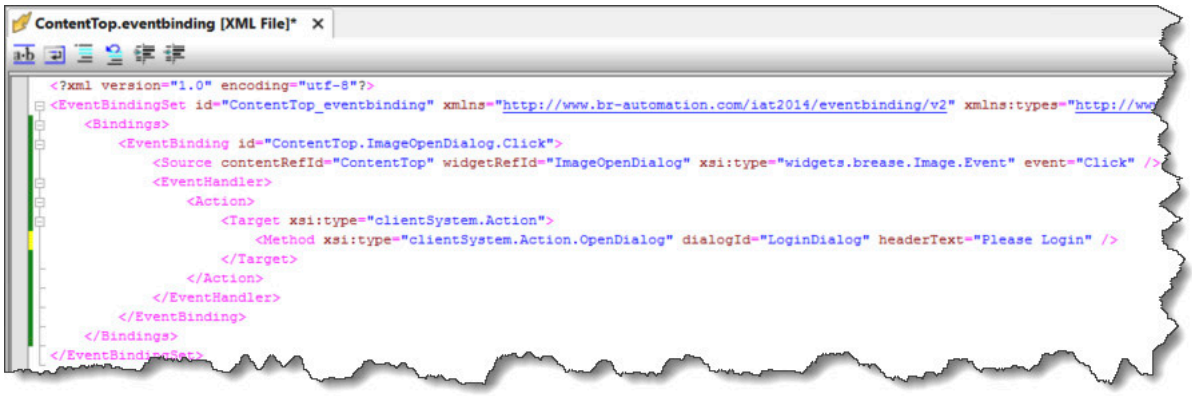


Figure 10: Completed event binding for opening the dialog box

Expected result

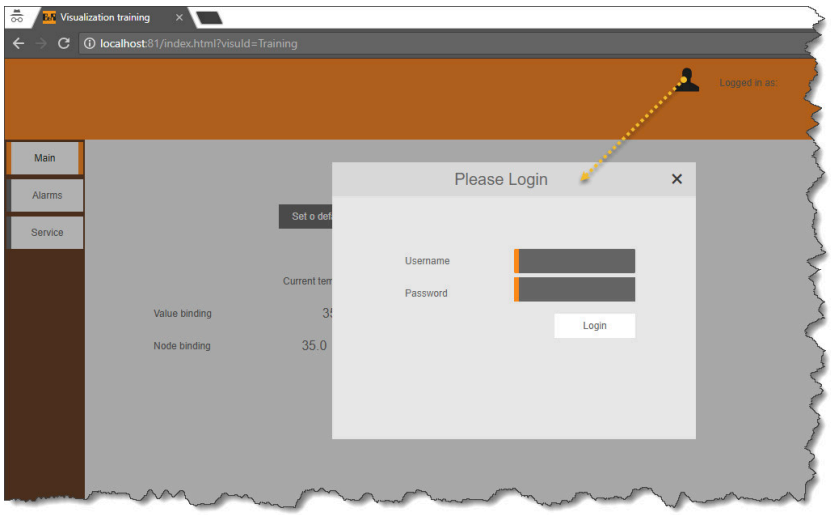


Figure 11: HMI application in the browser with an opened dialog box after clicking on the user image

2.2.6 Event binding for closing the dialog box

The "Login" widget provides an event that can be responded to if login is successful. If the widget is selected, ContentLogin switches to the event view and an event binding is configured for event "LoginSuccess".

```
<Source xsi:type="widgets.brease.Button.Event" contentRefId="ContentLogin"
widgetRefId="ButtonClose" event="Click" />
```

After the event has been defined, the subsequent action must be defined. The parameters that can be selected are defined in Automation Help.

Because a new event binding file with an EventBindingSet ID has been created for the active content, it must be referenced again in the HMI application.

> HMI application / mapp View / Engineering / Events and actions / Action / Client system actions / CloseDialog

The action for closing the dialog box is as follows:

```
<Target xsi:type="clientSystem.Action">
    <Method xsi:type="clientSystem.Action.CloseDialog" dialogId="LoginDialog"/>
</Target>
```

Expected result:

The dialog box is automatically closed after successful login.

2.3 MessageBox overview

A message box is shown via a page. The user must confirm the message box, which may execute an additional action depending on what is selected.

Figure 12: MessageBox example

A message box consists of a content area, header and button area.

The content area displays the message text that has to be shown and, optionally, an image.

A text can be displayed in the header

The button area can contain one or more buttons corresponding to the message box type.

Figure 13: Areas of a message box

2.3.1 Message box types

When calling a message box, stating the type specifies which buttons are shown. The type is selected according to the confirmation that is expected as a result of the event.

Message box type	Description
AbortRetryIgnore	Displays buttons for "Abort", "Repeat" and "Ignore"
OK	Displays an "OK" button
OKCancel	Displays buttons for "OK" and "Abort"
RetryCancel	Displays buttons for "Repeat" and "Abort"
YesNo	Displays buttons for "Yes" and "No"
YesNoCancel	Displays buttons for "Yes", "No" and "Abort"

Table 3: Message box types

2.3.2 Message box return value

If a message box of a certain type is shown, then confirmation is expected from the HMI application operator. Depending on the button pressed, the result can be evaluated in the event binding and the desired action can be configured.

Here, every button is assigned an ID that allows it to be evaluated in a condition.

Button ID	Value
Yes	1
No	2
OK	4
Cancel	8
Abort	16

Table 4: Message box button ID

Button ID	Value
Retry	32
Ignore	64

Table 4: Message box button ID

2.3.3 Areas of application

Message boxes are displayed to collect input from the user. A message box requires confirmation, which influences the process sequence according to the selection made by the user. Unlike a dialog box, the structure of a message box cannot be changed by the user. A message box is a system component.

2.4 Open message box and evaluate confirmation

The goal of this exercise is to configure an "OKCancel" message box which is opened on a certain event. An OPC UA variable with a defined value is described by pressing the "OK" button.

The message box is configured via the event binding.

Exercise: Configure an "OKCancel" message box

The following description explains how to configure a message box in which the OPC UA variable "Set-Temperature" is set to a defined value by pressing the "OK" button. The message box is closed without any additional action by selecting the "Cancel" button.

The following steps are necessary:

1) Use the button on the ContentMainPage to open the message box
2) Create an event binding from the "Button" widget
3) Call message box
4) Configure result of pressing the "OK" button using an OPC UA action

HMI application / mapp View / Engineering / Events and actions / Action / Client system actions / ShowMessageBox

2.4.1 Button widget for opening the message box

The "Button" widget placed in the ContentMainPage is used to open the message box. The name of the button has to be changed to "ButtonMsgBox" in this case. In addition, text for labeling the button has to be defined.

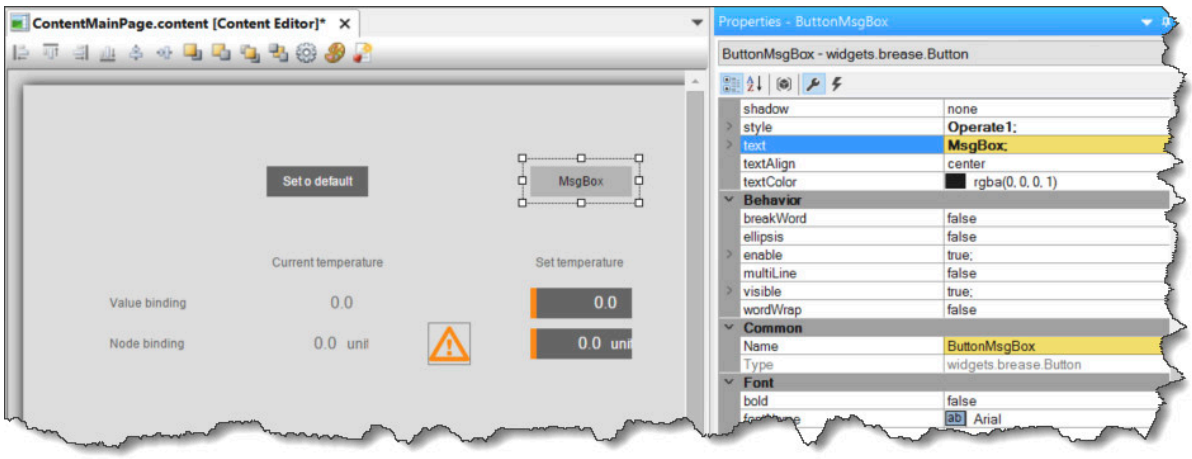


Figure 14: Button for opening a message box

2.4.2 Creating an event binding

A new <EventBinding> is created in the available event binding file.

As a source, the click event of the Button widget placed in the ContentMainPage with the name "Button-MsgBox" is used for opening the message box.

The creation process is started in the event view of the Properties window for the click event of the "Button" widget, just like with the previous exercises.

After the event has been defined, the subsequent action must be defined. The parameters that can be selected are defined in Automation Help.

> HMI application / mapp View / Engineering / Action / Events and actions / Client system actions / ShowMessageBox

The action for opening a message box of the type "OkCancel" looks as follows:

```
<Target xsi:type="clientSystem.Action">
    <Method xsi:type="clientSystem.Action.ShowMessageBox"
        type="OKCancel"
        message="Are you sure?"
        header="Reset SetTemperature"
        icon="Warning"  />
</Target>
```

> The message text (message=) and the text header (header=) can be referenced by specifying a localized text ID from the text system (e.g. $IAT/MessageText).

The next step is to configure the action for the expected result of a pressed button. The OPC UA variable "::AsGobalPV:SetTemperature" is set to a value = "30".

For this, a <Result> element with a <ResultHandler> that is available for every button must be configured using the respective <Action>. In the <ResultHandler>, the "OK" button with the button-Id=4 is reacted to as a return value in the condition.

```
<Result>
    <ResultHandler condition="result = 4">
        <Action>
            <Target xsi:type="opcUa.NodeAction"
            refId="::AsGlobalPV:SetTemperature" >
                <Method xsi:type="opcUa.NodeAction.SetValueNumber"
                value="30" />
            </Target>
        </Action>
    </ResultHandler>
</Result>
```

The entire event binding for opening a message box looks as follows:

```
<EventBinding>
    <Source xsi:type="widgets.brease.Button.Event"
    contentRefId="ContentMainPage" widgetRefId="ButtonMsgBox" event="Click"/>
    <EventHandler>
        <Action>
            <Target xsi:type="clientSystem.Action">
                <Method xsi:type="clientSystem.Action.ShowMessageBox"
                    type="OKCancel"
                    message="Are you sure?"
                    header="Reset SetTemperature"
                    icon="Warning"/>
            </Target>
            <Result>
                <ResultHandler condition="result = 4">
                    <Action>
                        <Target xsi:type="opcUa.NodeAction"
                            refId="::AsGlobalPV:SetTemperature" >
                            <Method xsi:type="opcUa.NodeAction.SetValueNumber"
                            value="30" />
                        </Target>
                    </Action>
                </ResultHandler>
            </Result>
        </Action>
    </EventHandler>
</EventBinding>
```

Expected result

When the button "MsgBox" is pressed, a dialog box is opened.

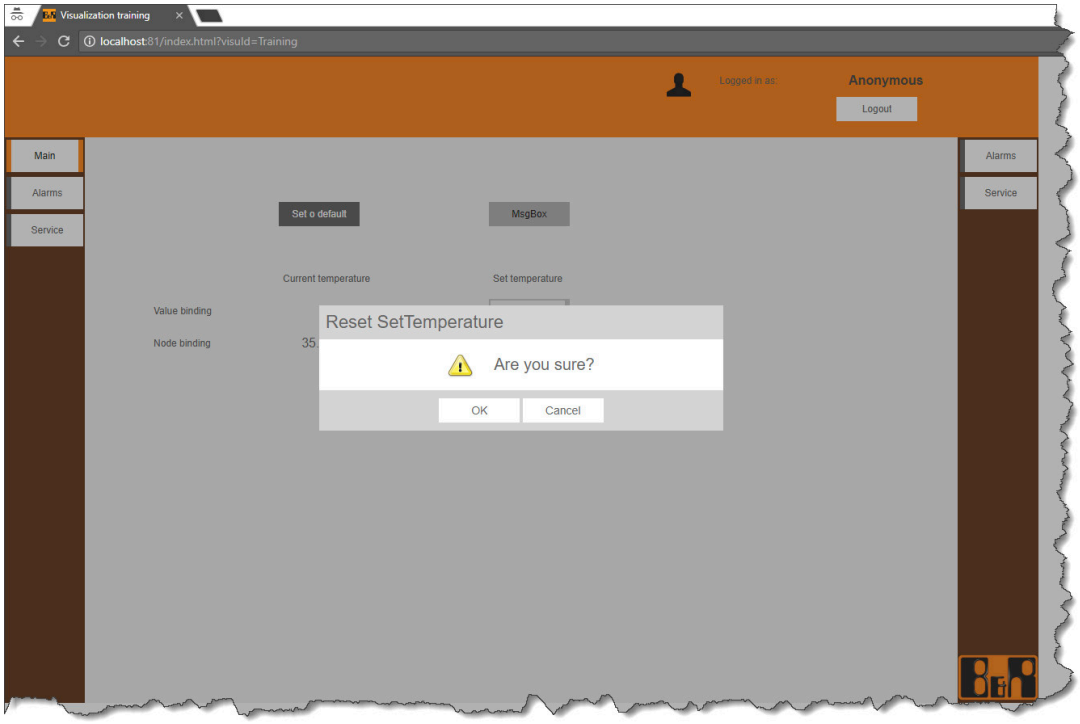


Figure 15: The message box is displayed in the foreground

What's expected when pressing the OK button is that the value of the OPC UA variable "SetTemperature" is set to the value "30".

Since login is carried out with the "Anonymous" user, write access to the OPC UA variable is not permitted in this case. If a user who has write access to the OPC UA variable "SetTemperature" logs in, then the value is also written.

A corresponding error entry "BadUserAccessDenied" is entered in the Automation Runtime logger.

2	Error	2016-12-14 10:35:33,890...	-1061091064	B&R	mapp View Server	OPC UA variable "::AsGlobalPV:SetTemperature" write failed. Status: BadUserAccessDenied

Figure 16: AR logger when write access is denied

3 Using roles and rights on widgets

3.1 Overview of roles and rights on widgets

There are two options available for controlling visibility or usability (enable state of a widget), where there is a distinction between a process-dependent and role-based functionality.

For example, if it is necessary to control visibility or usability of a widget or a group of widgets via OPC UA variables (state image of the process), then binding to the "visible" or "enable" property of a widget is required. There are some exercises for practicing this provided in this training module.

As already shown in TM611, write access to OPC UA variables can be restricted for individual roles. For input-dependent widgets, this has an effect on their enable state.

In addition, the visibility or usability of a widget can also be restricted as a function of the role assigned by the logged-in user.

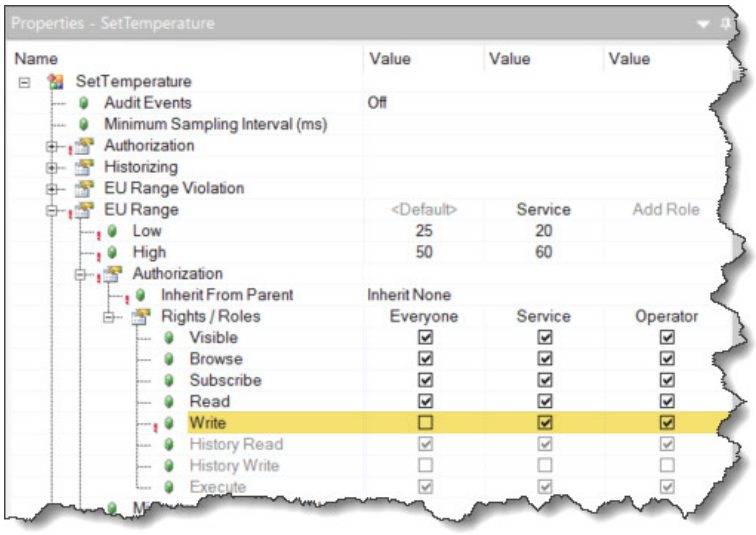


Figure 17: Restricted write access of an OPC UA variable for the role "Everyone"

HMI application / mapp View / Guides / FAQ / Widget applications / Restricting visibility and operation

3.2 Role-based visibility and usability of widgets

3.2.1 Configure roles on the property "permissionView"

All roles with users who are permitted to view content have to be entered in the property "permissionView" for the NavigationButton widget responsible for navigating to the ServicePage.

Configuration is carried out by entering the roles in the following syntax:

['Role'[,'Role*']]

Exercise: Configure access rights for visibility on the NavigationButton widget in "ContentLeft"

The goal of this exercise is to restrict the visibility of the "NavigationButton" widget for the role "Everyone" when navigating to the ServicePage.

Navigating to the ServicePage is only possible in the HMI application for users assigned "Operator" and "Service" roles. This means that after a user logs out, the navigation button is not visible.

The following steps are necessary for this:

1) Add roles to the property "permissionView" on the NavigationButton widget

If no role is entered, then the visibility for all roles is given, i.e. there is no restriction.

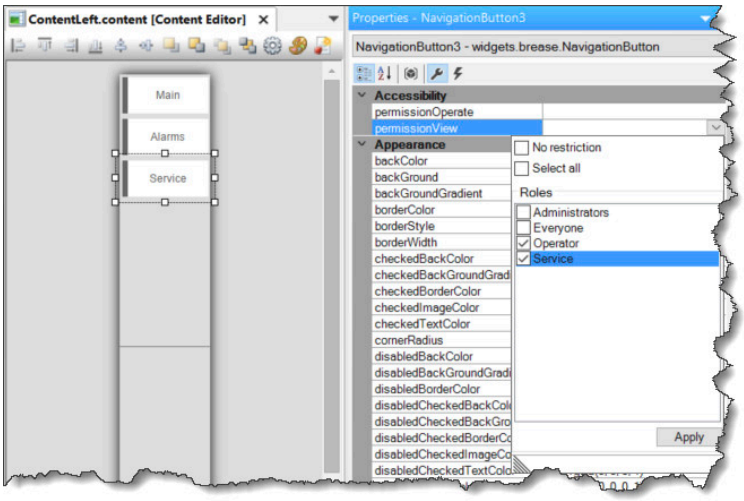


Figure 18: Configure the visibility controller on the property "permissionView"

Visibility cannot be restricted for NavigationButtons of the automatic navigation.

Expected result

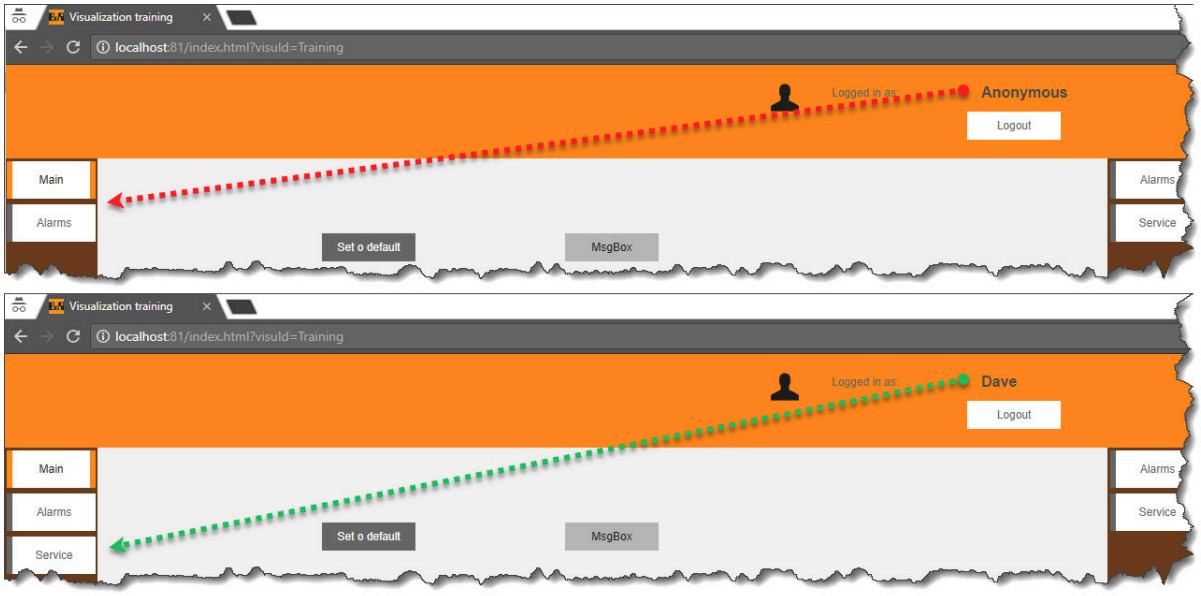


Figure 19: NavigationButton is only visible to users assigned the "Service" and "Operator" role

3.2.2 Role-based operation of widgets

All roles with users who are permitted to operate the system have to be entered in the property "permissionOperate" for the NavigationButton widget responsible for navigation to the ServicePage.

Configuration is carried out by entering the roles in the following syntax:

['Role'[,'Role*']]

Exercise: Configure access rights for operation on the NavigationButton widget in "ContentLeft"

The goal of this exercise is, in addition to the visibility of the NavigationButton widget, to restrict usability for the "Operator" role when navigating to the ServicePage.

In the last exercise, the visibility of the NavigationButton widget was already restricted. Specifying the role for the operation is done on the property "permissionOperate".

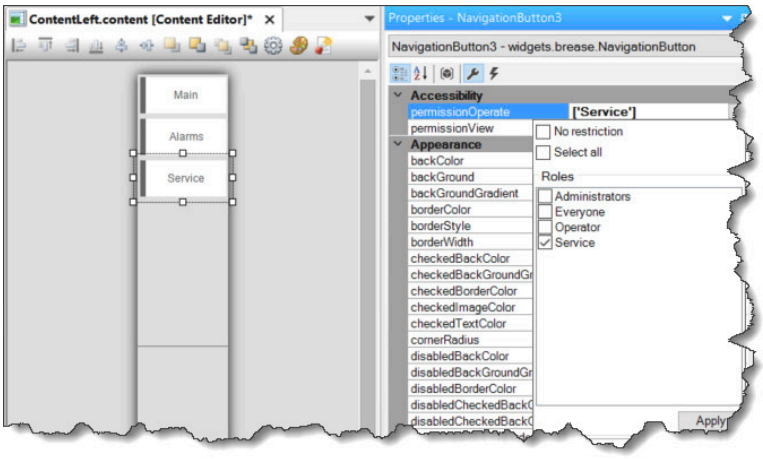


Figure 20: Configure access rights for the property permissionOperate

Expected result:

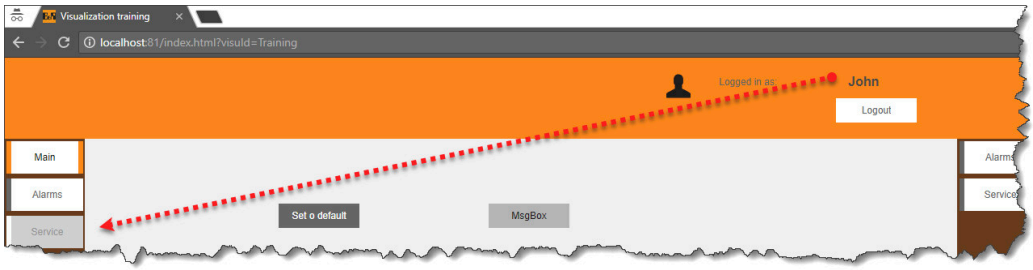


Figure 21: Users assigned the role "Operate" can see the NavigationButton but cannot use it

4 Animations in the HMI application

The configuration of static screen elements with a color scheme and appearance that can be determined in the content editor during configuration already makes it possible to create many elements of an attractive HMI application.

In addition, it's possible to change the visibility or usability of a widget or of a widget group as a function of process states or to change the visual appearance of a widget during operation.

Additional exercises show how, for example, an HMI application created for a certain size can be adjusted for an end device with a different resolution, and how a page is configured with a background image.

4.1 Simple visibility control for widgets

The following exercise shows how the visibility of a widget is controlled based on a process state, which is present as an OPC UA variable.

HMI application / mapp View / Guides / FAQ / Widget applications / Restricting visibility and operation

Exercise: Change visible state of a widget

The visibility of a widget should be controlled using a Boolean OPC UA variable.

The following steps are necessary for this:

1) Variable "State1" from data type "BOOL" is used.
2) Enable this variable as an OPC UA variable
3) Bind the OPC UA variable to a ToggleButton widget in the content "ContentMainPage"
4) Bind the OPC UA variable to the visible property of NumericOutput widget "NumericOutput1"

Steps 1-3 can now be carried out independently. Ensure that write access is also enabled for every role for the OPC UA variable.

4.1.1 Binding to the visible property

A binding to the OPC UA variable "State1" must be set up in the visible property of the NumericOutput widget.

To avoid flickering of the visibility of a widget during initial display, the default value must be set to "false".

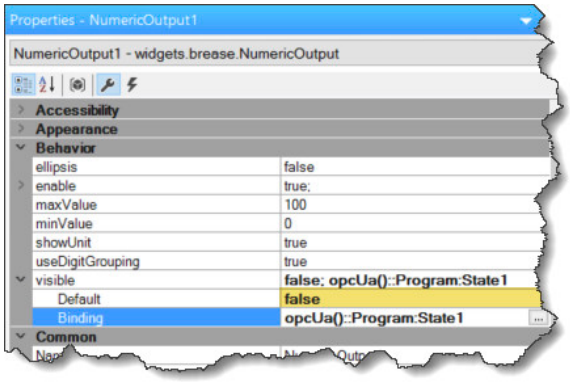


Figure 22: Visible binding of an OPC UA variable

Expected result

The ToggleButton "simulates" the process state for the visibility control between the state "true" or "false".

This state has an effect on the visibility of the NumericOutput widget.

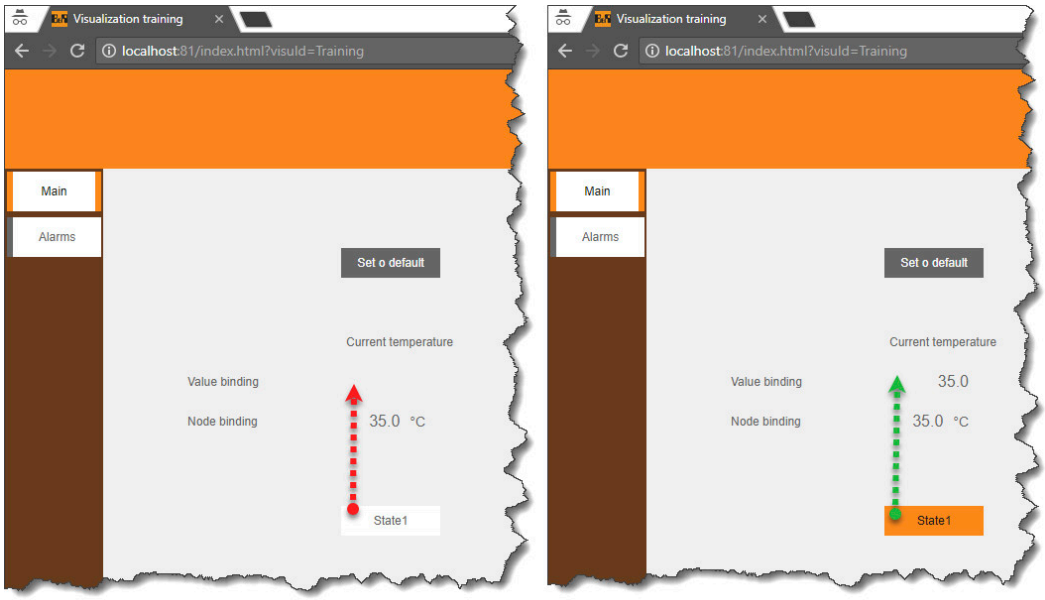


Figure 23: Visibility of the widget with a pressed or released ToggleButton

4.2 Visibility control calculated using multiple process values

This exercise shows how the visibility of a widget can be determined in the HMI application based on three process states that are present as OPC UA variables. When doing so, a calculation in the control program is not required.

The calculation of the result is configured in the HMI application using expressions. An expression is a construct that returns a value after it has been evaluated.

Result = InA (Bool) AND InB (BOOL) AND (InC (INT) > 30)

HMI application / mapp View / Engineering / Variables and data / Expressions

Exercise: Create an expression and link the result to a visible property

The following description shows how to configure an expression with a result that is connected to the visible property of a NumericOutput widget.

The following steps are necessary for this:

1) The variables "State1" and "State2" of data type "BOOL" and "State3" of data type "REAL" are used.
2) Enable these variables as OPC UA variables
3) Bind the OPC UA variables "State1" and "State2" each to a "ToggleButton" widget in the content "ContentMainPage"
4) Bind the OPC UA variable "State3" to a BasicSlider widget in the content "ContentMainPage"
5) Add and configure an expression and an ExpressionType in the Logical View
6) Bind the OPC UA variables to the operands of the expression
7) Bind the expression results to the visible property of a NumericInput widget

Steps 1-4 can now be carried out independently. Ensure that the OPC UA variables are also permitted write access for every role.

4.2.1 Configure expression type

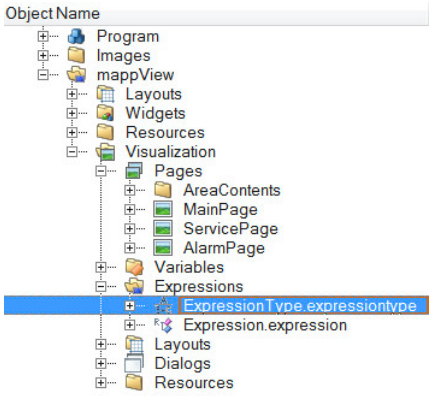


Figure 24: ExpressionType and expression file in the Logical View

An expression is configured in two steps. The first step is to create an expression type as a template of an expression instance. For this, an "Expression Type" file from the Object Catalog is added in the Expressions node for the visualization package.

After opening the file, a unique ExpressionTypeSet ID with the name "myExpressionTypeSet" must be added.

This ID must not be referenced in the HMI application (.vis).

```
<?xml version="1.0" encoding="utf-8"?>
<ExpressionTypeSet id="myExpressionTypeSet"
xmlns="http://www.br-automation.com/iat2016/expressionType/v1">
    <ExpressionTypes>

    </ExpressionTypes>
</ExpressionTypeSet>
```

The next step is to configure an ExpressionType with name "myCalculation" and data type "BOOL".

```
<ExpressionType name="myCalculation" datatype="BOOL">

</ExpressionType>
```

The operands and the linking operation are defined within the ExpressionType.

```
<Operands>
    <Operand name="State1" datatype="BOOL" />
    <Operand name="State2" datatype="BOOL" />
    <Operand name="State3" datatype="ANY_REAL" />
</Operands>
<Operation>
    (State3 &gt; 30 AND State1) AND State2
</Operation>
```

The operation defines how the operands are linked. This calculation results in a return value of "true" or "false" based on the data type specified in the ExpressionType.

The finished configuration of the ExpressionType looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<ExpressionTypeSet id="myExpressionTypeSet"
xmlns="http://www.br-automation.com/iat2016/expressionType/v1">
    <ExpressionTypes>
        <ExpressionType name="myCalculation" datatype="BOOL">
            <Operands>
                <Operand name="State1" datatype="BOOL" />
                <Operand name="State2" datatype="BOOL" />
                <Operand name="State3" datatype="ANY_REAL" />
            </Operands>
            <Operation>
                (State3 &gt; 30 AND State1) AND State2
            </Operation>
        </ExpressionType>
    </ExpressionTypes>
</ExpressionTypeSet>
```

4.2.2 Configuring an expression

The second step is to configure an instance of an expression. For this, an "Expression" file from the Object Catalog is added in the Expressions node of the visualization package. After opening the file, a unique ExpressionSet ID with the name "myExpressionSet" must be added.

```
<?xml version="1.0" encoding="utf-8"?>
<ExpressionsSet id="myExpressionSet"
xmlns="http://www.br-automation.com/iat2015/expression/engineering/v3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance">
 <Expressions>
 </Expressions>
</ExpressionsSet>
```

This ID must be referenced in the element <ExpressionsSets> of the HMI application (.vis).

```
<ExpressionsSets>
        <ExpressionsSet refId="myExpressionSet"/>
</ExpressionsSets>
```

Finally, the expression itself is added under <Expressions>.

```
<Expression xsi:type="content" id="myExprResult"
contentRefId="ContentMainPage" type="myCalculation" />
```

Once the configuration is completed, the expression looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<ExpressionsSet id="myExpressionSet"
    xmlns="http://www.br-automation.com/iat2015/expression/engineering/v3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Expressions>
        <Expression xsi:type="content" id="myExprResult"
        contentRefId="ContentMainPage" type="myCalculation" />
    </Expressions>
</ExpressionsSet>
```

4.2.3 Binding of the OPC UA variables to the operands of an expression

The required OPC UA variables "State1", "State2" and "State3" can be connected to the corresponding operands so that the expression can also be calculated during operation. The binding occurs manually in a binding file. An OPC UA variable (source) is connected via a "oneWay" binding to the operand of an expression.

```
<Binding mode="oneWay">
    <Source xsi:type="opcUa" refId="::Program:State1" attribute="value" />
    <Target xsi:type="expression" refId="myCalculation" attribute="State1" />
</Binding>
```

The expression ID " myExprResult " is referenced in the <Target>, and the operand name is transferred as an attribute. Once completed, the binding for all operands looks like this:

```
    <Binding mode="oneWay">
        <Source xsi:type="opcUa" refId="::Program:State1" attribute="value" />
        <Target xsi:type="expression" refId="myExprResult" attribute="State1" />
    </Binding>
    <Binding mode="oneWay">
        <Source xsi:type="opcUa" refId="::Program:State2" attribute="value" />
        <Target xsi:type="expression" refId="myExprResult" attribute="State2" />
    </Binding>
    <Binding mode="oneWay">
        <Source xsi:type="opcUa" refId="::Program:State3" attribute="value" />
        <Target xsi:type="expression" refId="myExprResult" attribute="State3" />
    </Binding>
```

In the last step, the calculated result must be connected to the visible property of a widget.

4.2.4 Binding the expression results to the visible property

The calculation of the expression returns a Boolean return value. This return value, which was configured in the expression by the ID "myExprResult", can now be connected to the visible property of a widget. Binding to the visible property is done in the variable selection dialog box of the "NumericOutput2" widget. The source of the type "expression" and of the attribute "result" is connected to the corresponding widget property.

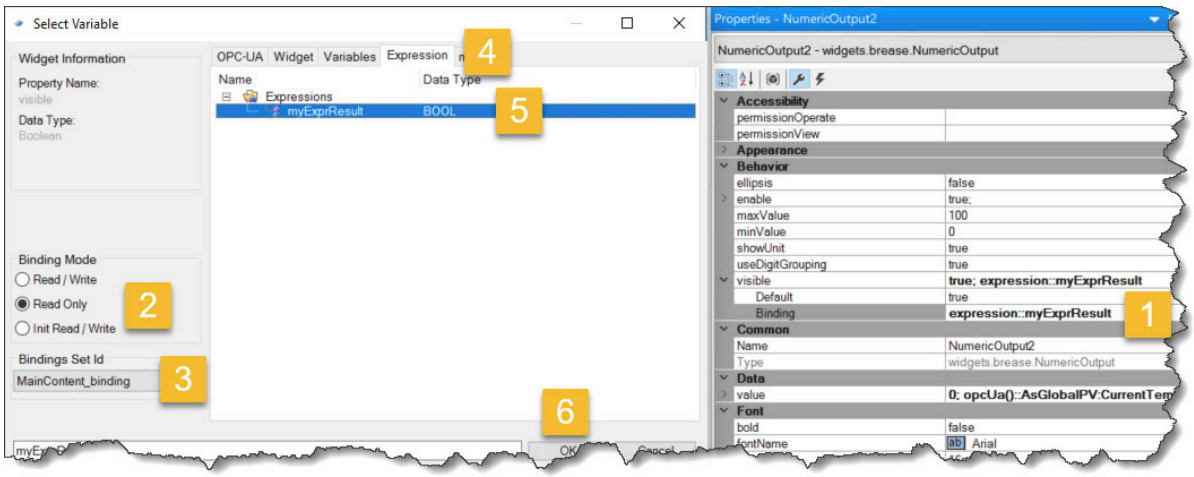


Figure 25: Selecting the expression results

Expected result

A TRUE or FALSE value is returned through the operation of the three operands. The result is used for controlling the visibility of a widget.

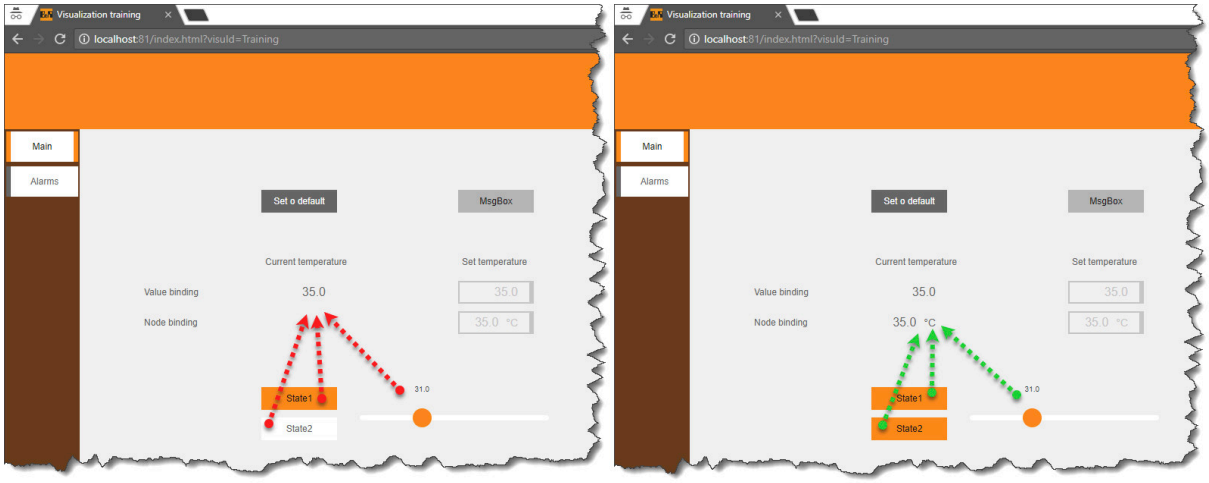


Figure 26: Visibility of the NumericOutput widget based on the expression result

4.3 Background image for a page

In the previous exercises, the background color of the corresponding area was also defined when assigning a piece of content to a page.

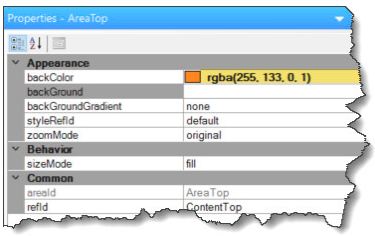


Figure 27: Specify the background color of an area in the Page editor

Exercise: Display a background image on the MainPage

The goal of this exercise is to display an image over the entire page instead of the area colors. The following chapter is used to configure an image to a page. The following steps are necessary for this:

1) Select an image with a page ratio of 16:9
2) Save the image in the media package
3) Reference the image of the .page file of the MainPage
4) Change the background color of the areas

Steps 1-2 can now be carried out independently. A random image can be used.

4.3.1 Reference the background image of the MainPage

After inserting the image into the media package, this can be referenced on the page. The reference is defined in property backGround="Media/Images/Background.png".

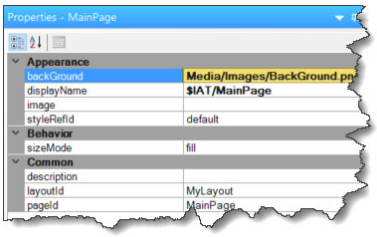


Figure 28: Specify the background for a page in the Page editor

When the project is transfered, an outcome is not yet displayed. Only by setting a transparent background color for the area with backColor="transparent" is the image also shown during operation.

The background color for an area can be reset by selecting "Reset" in the shortcut menu of property "backColor".

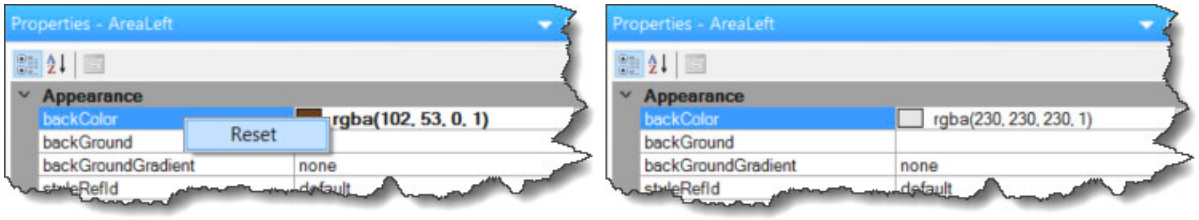


Figure 29: Reset the background color using Shortcut menu / Reset

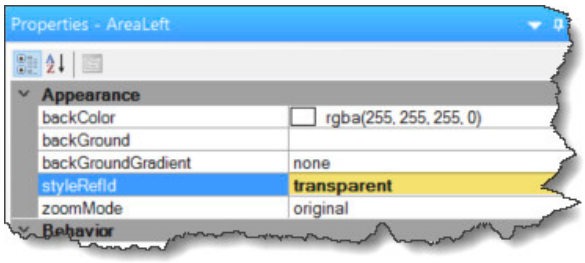
After resetting, the default style for the area is applied, which is determined by the theme that is defined.

Default style with gray background color:

```
<Style id="default" xsi:type="system.brease.Area"
       backColor="#e6e6e6" />
```

Transparent style that will be applied to the area.

```
<Style id="transparent" xsi:type="system.brease.Area"
       backColor="transparent" />
```



In order to use a transparent background, the transparent style must be defined for property "styleRefId" for each area.

Figure 30: Apply "transparent" style to area

The background image is fully shown on the the MainPage in the first result of this exercise.

Expected result

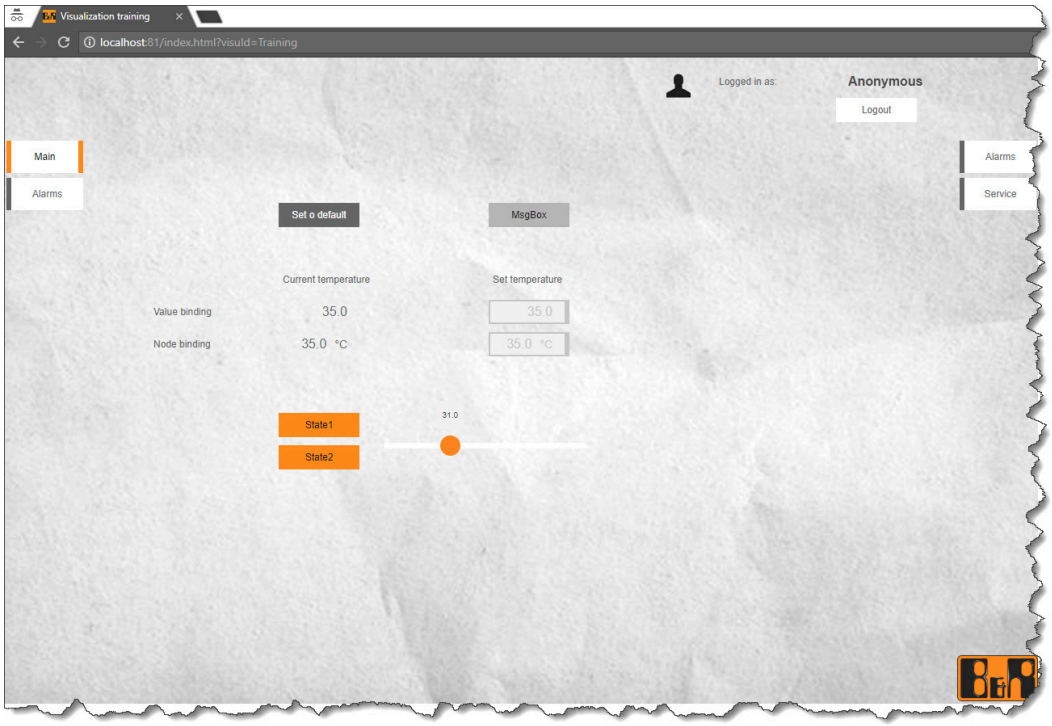


Figure 31: Display a background image on a page

4.4 Relative and absolute positioning in Container widgets

By default, widgets are depicted on the content with an absolute top/left position. As previously shown in TM611, the "NavigationButton" widgets in the ContentLeft were relatively positioned in a "NavigationBar" widget. If a "Container" widget offers the property "childPositioning", then you can choose between absolute and relative positioning of its "Child" widgets here. The goal of this exercise is to position random widgets row by row in a GroupBox. For this, a transparent GroupBox is added as a line and a binding to its visible property is set up.

Exercise: Hide lines in a GroupBox

The following chapter shows how to configure a GroupBox in which lines can be shown and hidden with the binding to the visible property. This can be necessary due to a certain machine configuration, for example.

The following steps are necessary for this:

1) Add a new event binding file with the "ServiceContent_binding" binding set ID.
2) Reference the binding set ID in the HMI application (.vis)
3) Add a GroupBox at a relative position to the content ContentServicePage
4) Add the transparent GroupBoxes as child boxes for the first GroupBox (= line)
5) Add widgets to the respective GroupBoxes
6) Bind the OPC UA variable "::Program:State1" to the visible property of the first child GroupBox in the new binding file

4.4.1 Add and configure GroupBoxes

A GroupBox is added in the "ContentServicePage" and the property "childPositioning" configured to the value "relative".

Property name	Value
childPositioning	relative
Position	25;480
Size	580;270

Table 5: Properties of the GroupBox

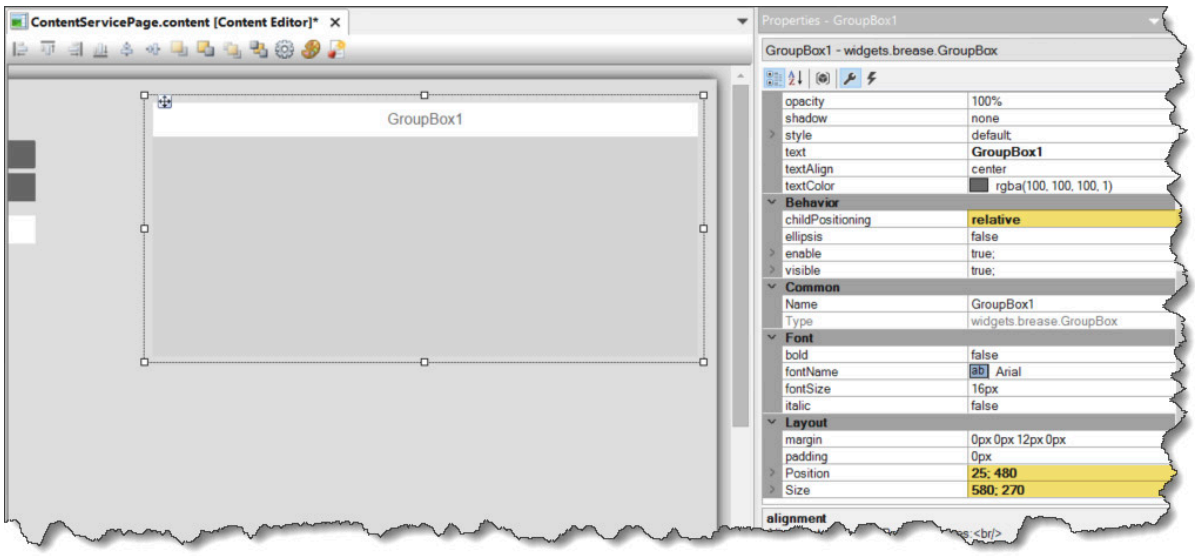


Figure 32: Parent GroupBox with relative positioning

Three additional GroupBoxes are added in this GroupBox and the size is adjusted to the width of the parent. The header of the GroupBoxes is hidden by deleting the default text in the "text" property, and it is shown without a border and background color by referencing style="transparent".

All three GroupBox widgets are configured with the following properties:

Property name	Value
Style	Transparent
Text	
Position	0;0
Size	550;55
Margin	10px

Table 6: Properties of the GroupBox widgets

This is automatically labeled top/left="0" when dragging a GroupBox within the relatively positioned parent GroupBox. A Label, Button and NumericInput widget with absolute positions are added into every GroupBox.

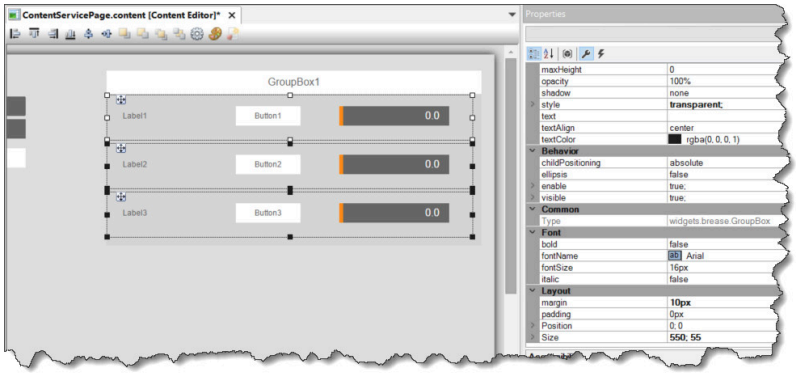


Figure 33: Completely configured GroupBoxes – Display with multi-select

Subsequently, the first child GroupBox is configured as a value binding to the OPC UA variable "::Program:State1 via the visible property.

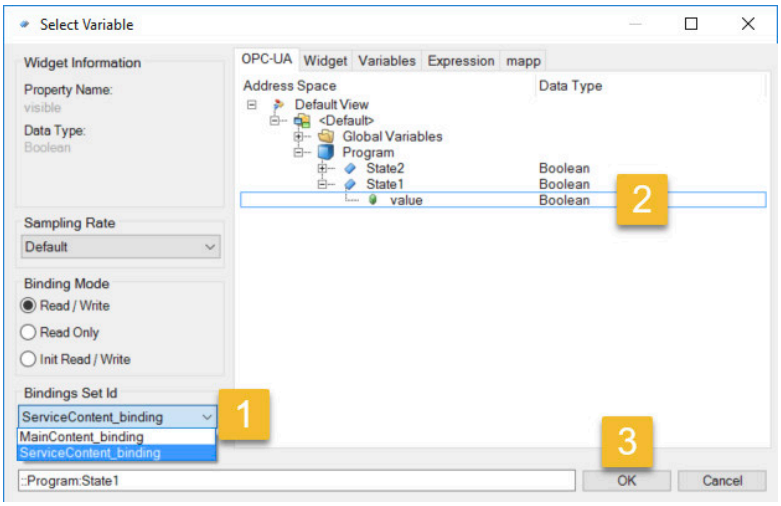


Figure 34: Select the BindingsSet ID in the variable selection dialog box

Expected result

If the ToggleButton for changing the OPC UA variable "::Program:State1" is changed on the MainPage, then the first child GroupBox is shown/hidden through relative positioning. The following GroupBox widgets fill the empty space in the document.

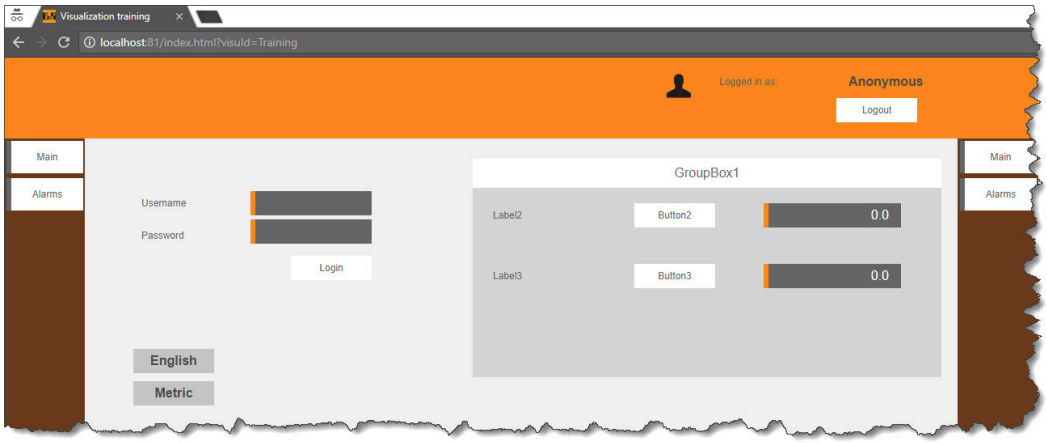


Figure 35: Hidden first line in a GroupBox

4.5 Scaling the HMI application to the size of the browser window

An HMI application is created for the resolution of a display. The size of the page is determined via the layout that is referenced in a page. If it's necessary to display the same HMI application on different size displays (e.g. HD Ready and Full HD), this can be set up by configuring automatic zoom.

Ensure that the HMI application can be operated on both end devices with different diagonal sizes. In addition, the aspect ratio on the end devices must be the same (e.g. 4:3 or 16:9).

Exercise: Automatic adjustment of the HMI application to the browser window

The zoom setting is configured for HMI application in the element <Configurations>. Automatic zoom is enabled with the configuration entry key="zoom" with value="true".

```
<Configuration key="zoom" value="true" />
```

The zoom setting is adjusted once when the HMI application is provided in correspondence with the size of the browser window. A dynamic change of size newly depicts the page; some widgets cannot react to the size change, however, and maintain their initial size or input field.

Expected result

Resizing the browser window and restarting the HMI application scales the HMI application accordingly. The piece of content is scaled so that it is fully visible at maximum size.

5 Tasks with different variable types

In previous exercises, we have used the following types of variables for use in an HMI application:

Variable type	Scope
OPC UA variables	Controller program
Expressions	Visualization

Table 7: Overview of the previously used types of variables

These variables were connected by a value binding or node binding to a widget instance for a piece of content. This chapter applies an additional variable and binding type for new tasks with corresponding exercises.

Variable type	Scope
Session variables	Visualization

Table 8: Additional variable type

HMI application / mappView / Engineering / Variables and data

Session variables

A session variable is a variable whose scope is a session. In other words, its values apply within a specific client-server connection. This makes it possible to store the status of the HMI application for each client. Session variables can only be read and written using value binding.

List binding

For list binding, an element is selected from the list of variables using a selector with which the binding between <Source> and <Target> should be executed.

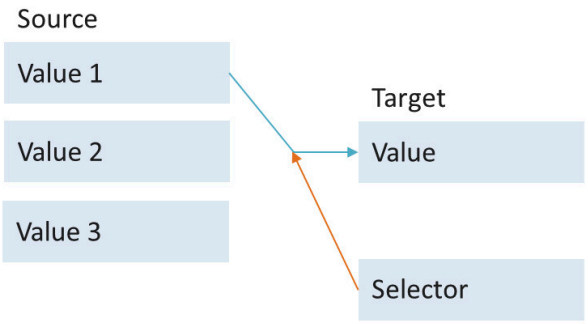


Figure 36: Binding to an element of a list

5.1 Exchange information with different widgets

Properties of widgets can be connected within a piece of content using Brease-Brease binding.

In doing so, the variable type for <Source> as well as for <Target> is defined by xsi:type="brease". The scope of a Brease-Brease binding is limited to the same content.

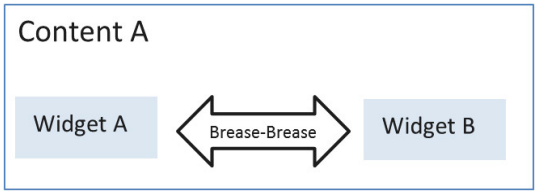


Figure 37: Widget information exchange within the same content

Since a state or value change is not saved for navigation, a state change of Content A would only have an effect if Content B is also displayed on a page at the same time.

If a page that does not contain Content B were navigated to, the state change to Content A by the binding would be invalid (source and target are not simultaneously active) and would therefore be lost. Session variables are used for caching state changes or value changes of a widget. These also retain the value if the corresponding content is not shown.

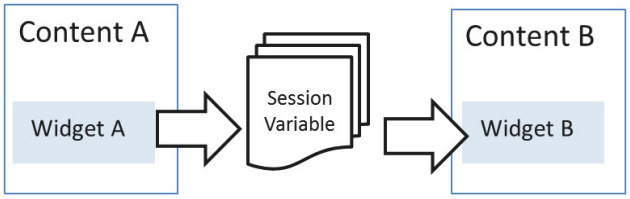


Figure 38: Widget information exchange via content borders

5.1.1 Widget information exchange within the same content

The goal of the exercise is to show the value of a widget on a piece of content without having to use an OPC UA variable on another widget for the same piece of content.

Exercise: Display value of a BasicSlider widget on a RadialGauge widget

The following chapter shows the movement of a BasicSlider widget as a pointer deflection in a Radial-Gauge widget. In the process, the connection of the "value" property occurs directly between the widgets.

The following steps are necessary for this:

1) Create a new "TestPage" page and "ContentTestPage" content and expand automatic navigation
2) Reference the TestPage in the HMI application (.vis)
3) Add a new binding file with the "TestContent_binding" binding set ID.
4) Reference the binding set ID in the HMI application (.vis)

5) A BasicSlider widget and a RadialGauge widget are added in the ContentTestPage of the new page.

6) The value property of the RadialGauge widget is linked with the value property of the BasicSlider widget.

Steps 1 to 4 can now be done without assistance.

5.1.2 Brease-Brease binding

The binding of the property "value" of the "RadialGauge" widget must be executed since the value change of the "BasicSlider" widget (source) should be shown on the "RadialGauge" widget (target).

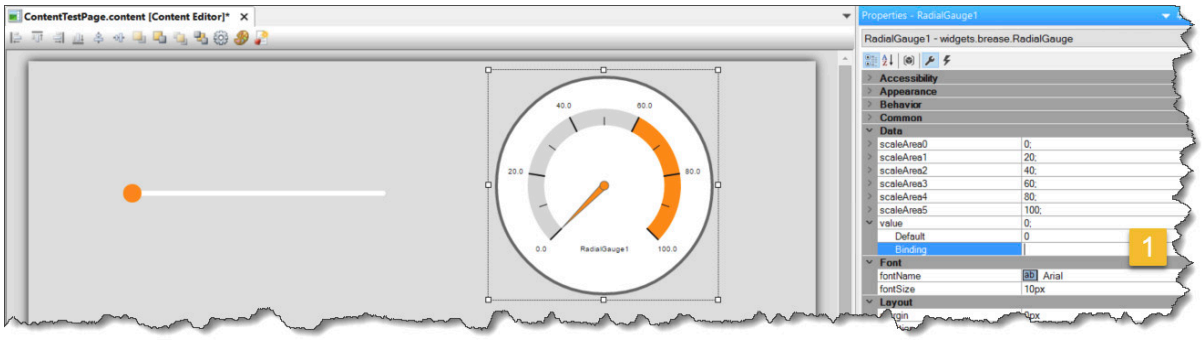


Figure 39: Setting up the binding on the value property of the RadialGauge widget

All widgets placed in the content are shown with their bindable properties in the "Widget" tab of the variable selection dialog box. Binding is done by selecting the "value" property for the widget value that is to be displayed. Since only read access occurs, the binding mode can be set to "read only".

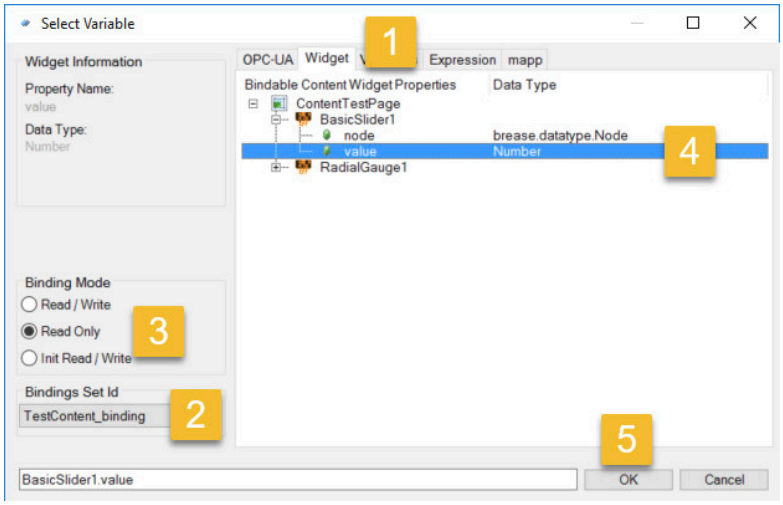


Figure 40: Value binding on a widget located in the same content

Expected result

A movement of the slider is shown by the pointer movement of the RadialGauge on the TestPage.

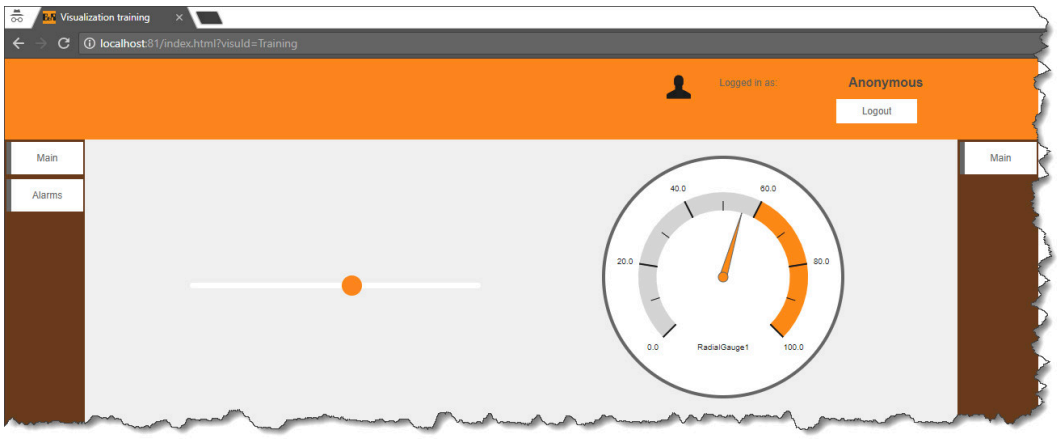


Figure 41: Changing the value of the slider moves the arrow pointer of the RadialGauge

5.1.3 Widget information exchange via content borders

The goal of the exercise is to show the value of a widget on a piece of content without the use of an OPC UA variable on another widget for another piece of content.

Exercise: Display value of a BasicSlider widget on a NumericOutput widget

Using the following chapter, the value of a BasicSlider widget in the content "ContentTestPage" is displayed on a NumericOutput widget in the content "ContentTop". The "value" property is connected here using a session variable.

The following steps are necessary for this:

1) Add a variable file to the visualization package
2) Define a session variable "SliderValue" of the type "ANY_REAL"
3) Bind the value property of a BasicSlider widget in the ContentTestPage with the session variable
4) Bind the value property of a NumericOutput widget in the ContentTop with the session variable

5.1.4 Session variables

Session variables are managed in the Logical View of a visualization package in a .svar file.

Visualisierung / mapp View / Engineering / Variablen und Daten / Session Variablen

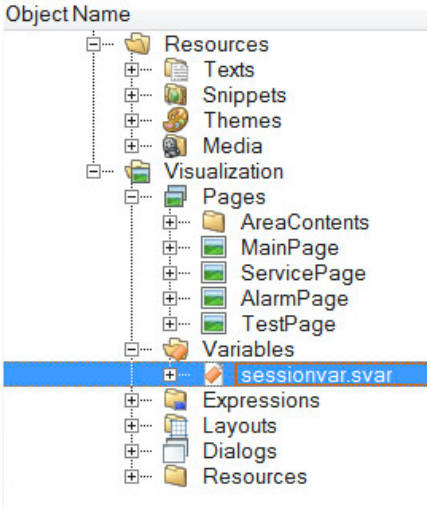


Figure 42: Session variable file in the Logical View

For this, a "session variable" file from the Object Catalog is added in the Variables node of the visualization package.

After opening the file, a unique VariableSet ID with the name "myVariables" is defined. This ID must be referenced in the element <VariablesSets> of the visualization object (.vis).

```
<VariablesSets>
    <VariablesSet refId="myVariables" />
</VariablesSets>
```

To finish, the variable "SliderValue" with data type "ANY_REAL" is added under <Variables>. The initial value of the session variable = "0".

```
<Variable name="SliderValue" xsi:type="ANY_INT" value="0" />
```

A finished variables file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<VariablesSet id="myVariables"
xmlns="http://www.br-automation.com/iat2015/session/engineering/v1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Variables>
        <Variable name="SliderValue" xsi:type="ANY_REAL" value="0" />
 </Variables>
</VariablesSet>
```

The next step is to add a BasicSlider widget to the ContentTestPage. A binding to the session variable is carried out on the "value" property. The session variables are shown for selection in the tab "Variables".

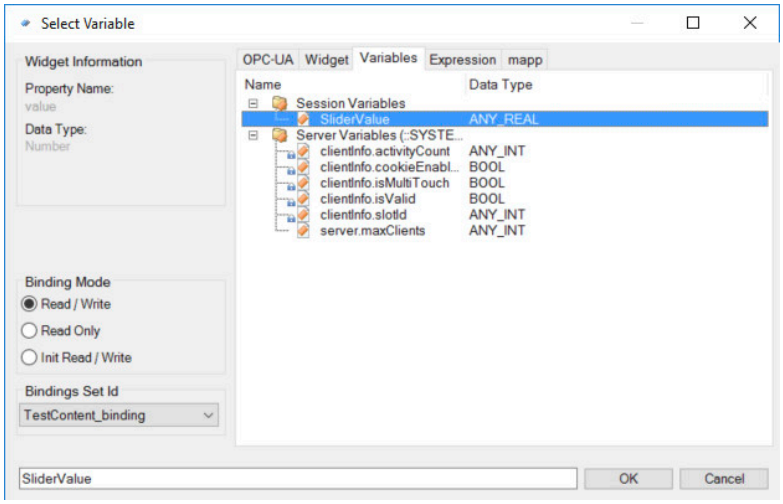


Figure 43: Selecting the session variable

To finish, a "NumericOutput" widget is added in the ContentTop and the same session variable connected to its value property.

Expected result

Moving the slider on the TestPage displays the value in the header. The value remains in the session variable even if you navigate in the HMI application.

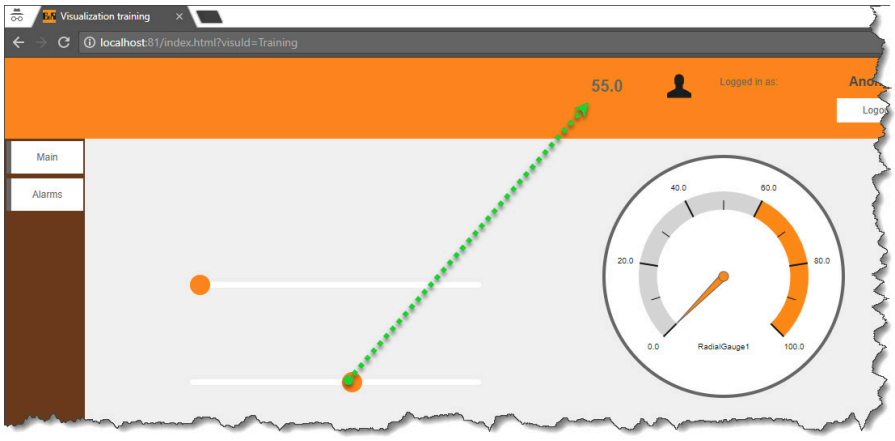


Figure 44: Binding spanning different pieces of content via a session variable

5.2 Changing the color of widgets

In this exercise, a reaction to the state change of the process sequence is used to change the visual design of a widget. This is how (e.g. when an error occurs in the process sequence) the HMI application operator can be made aware of the change of the display of a widget.

Figure 45: A state in the process sequence is displayed visually.

As already shown in TM611, the visual design is carried out with styleable properties or the style reference.

The process state is represented by the OPC UA variable "::Program:State2" which can be changed via a ToggleButton on the MainPage. Two styles are required that represent the visual appearance for the TRUE and FALSE status.

5.2.1 Define the style for the "TRUE" state

Style "Operate1" is referenced on the Button widget for opening the message box named "ButtonMsgBox".

A new style should be created based on this style that has a red border color and a defined frame width.

Exercise: Button style for the error state

The goal of this exercise is to create a new style with the name "Error" for widget type "Button".

Style "Operate1" is added together with BuRTheme1 and is included in the respective styles file "Button.styles".

By copying and pasting this style, you can begin changing the necessary styleable properties.

The following changes must be made in the file "Button.styles":

1) Copy the button style "Operate1"
2) After adding to a new line, change the ID to "Error".
3) Change the borderWidth and mouseDownBorderWidth to "2px 2px 2px 2px"
4) Change the attribute borderColor and mouseDownBorderColor as “#FF0000“ for a red frame

This exercise must be completed independently with the mapp View help documentation.

This differs from Automation Help: Use "Operate1" for the false state and use the OPC UA variable "::Program:State2"

HMI application / mapp View / Guides / FAQ / Event and action applications / Behavior / Validity of events

Expected result

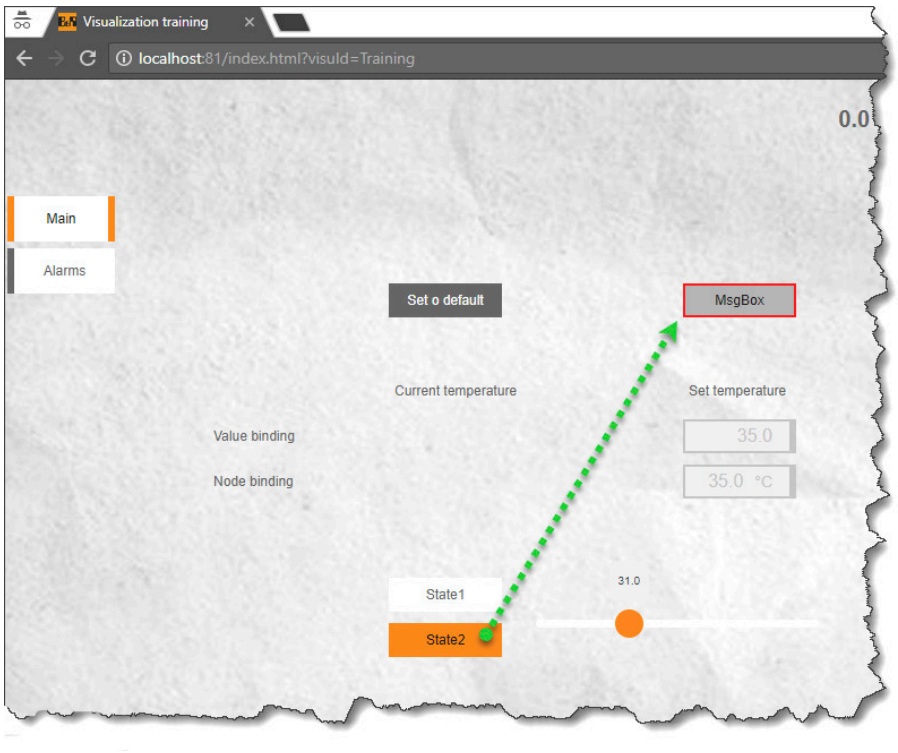


Figure 46: Style will change if status changes

5.3 Changing users and switching the page

This exercise shows how, at the end of a production shift, the active user logs out of the HMI application by clicking on a logout icon or logout button. While logging out, the HMI application should switch back to the MainPage in order to be able to log back in again.

Exercise: Log out by clicking on the image and navigating to the MainPage

The following description shows how to configure a logging out and navigating to a certain page.

The following steps are necessary for this:

1) Logging out is done by clicking on the B&R symbol in the content NavigationRight.

2) After logout, automatic navigation to the MainPage must be executed

This exercise must be completed independently with the mapp View help documentation.

HMI application / mapp View / Guides / FAQ / Event and action applications / Changing page with logout

Expected result

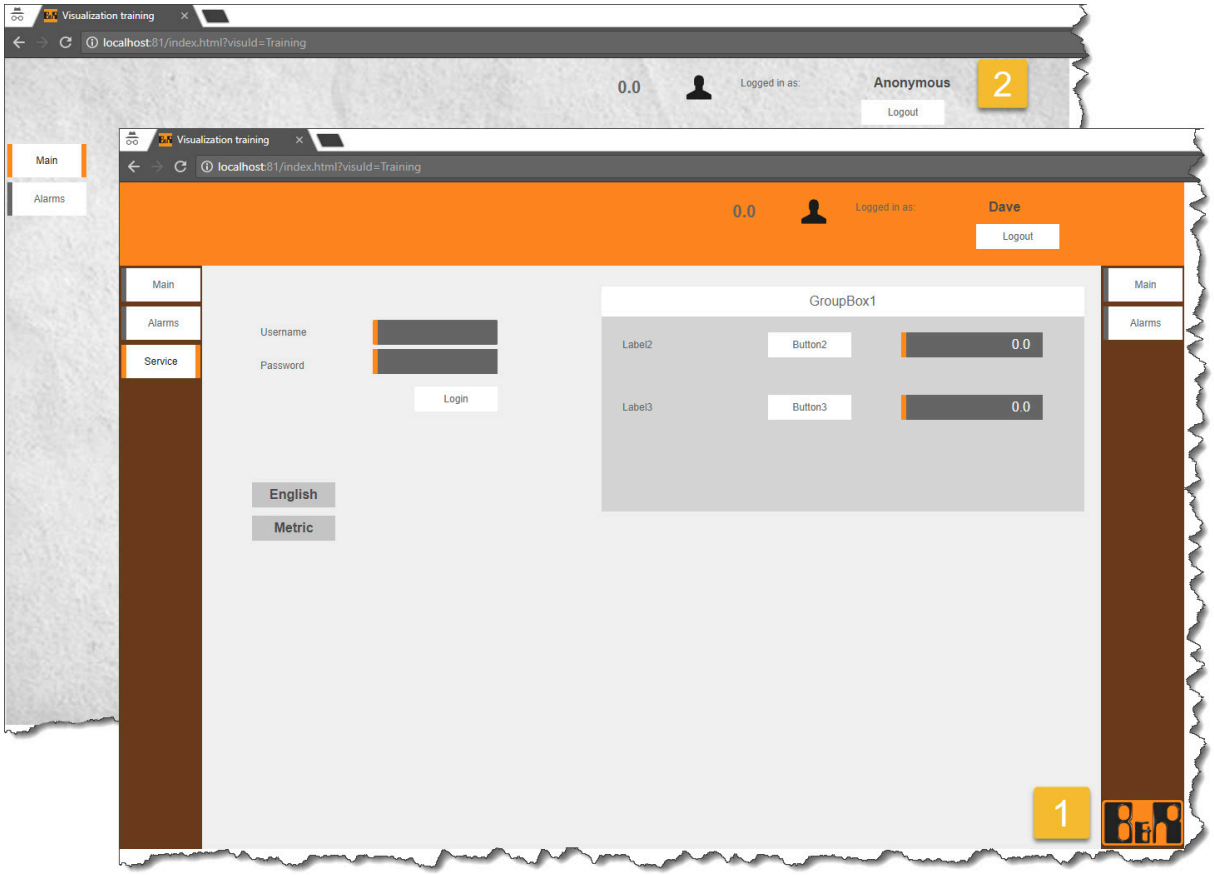


Figure 47: Logout with navigation to MainPage

5.4 Using system variables

System variables are permanently predefined variables that are filled with values by the mapp View server at runtime.

> HMI application / mapp View / Engineering / Variables and data / Session variables / System variables

System variables can be used to evaluate client information for every connected client in the control program. The goal of this exercise is to make it possible to evaluate the client information in the control program using the mapp View help documentation. This information is available in the program for every client with the proper permissions by binding the system variables to an OPC UA structure array.

Exercise: Map system variables on an OPC UA structure array

The following description shows how the client information of all connected clients is bound to an OPC structure array.

The following steps are necessary for this:

1) Generate a structure type for the variables that have to be transported
2) Generate a global variable "ClientInfo" with the above structure type and a length of [0..2]
3) Use variable "ClientInfo" in the existing program
4) Add a new binding file
5) Define binding ID and reference in the .vis file
6) Copy the list binding from the help documentation

This exercise is completed independently with the mapp View help documentation. Implementation is described in detail. The exercise only has to be carried out up to the chapter "Configuring a list binding of client information via OPC UA".

List binding is used for this exercise. The following XML code shows a excerpt for a system variable:

```
<Binding mode="oneWayToSource">
<Source xsi:type="listElement">
<Selector xsi:type="session" refId="::SYSTEM:clientInfo.slotId"
attribute="value" />
    <be:List xsi:type="be:opcUa" attribute="value" >
        <bt:Element index="0" refId="::AsGlobalPV:ClientInfo[0].userId" />
        <bt:Element index="1" refId="::AsGlobalPV:ClientInfo[1].userId" />
        <bt:Element index="2" refId="::AsGlobalPV:ClientInfo[2].userId" />
    </be:List>
</Source>
<Target xsi:type="session" refId="::SYSTEM:clientInfo.userId" attribute="value" />
</Binding>
```

The following image shows the effect of a "oneWayToSource" binding between <Target> and a list of OPC UA variables in the <Source> element. The client first logged onto the mapp View server is assigned the slotId = "0" by the server. Therefore this points to the OPC UA variable in the list "::ASGlobalPV.ClientInfo[0].userId". As a result, the name of the user is written by the system variable defined in the <Target> on the corresponding variable for the first element [0] of the structure array.

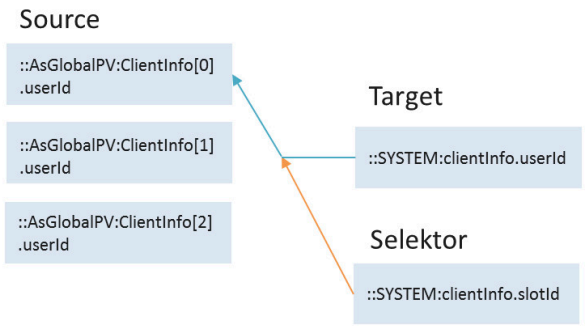


Figure 48: Schema of the list binding for the user ID

Expected result

The Watch window can be used to observe client information in the control program.

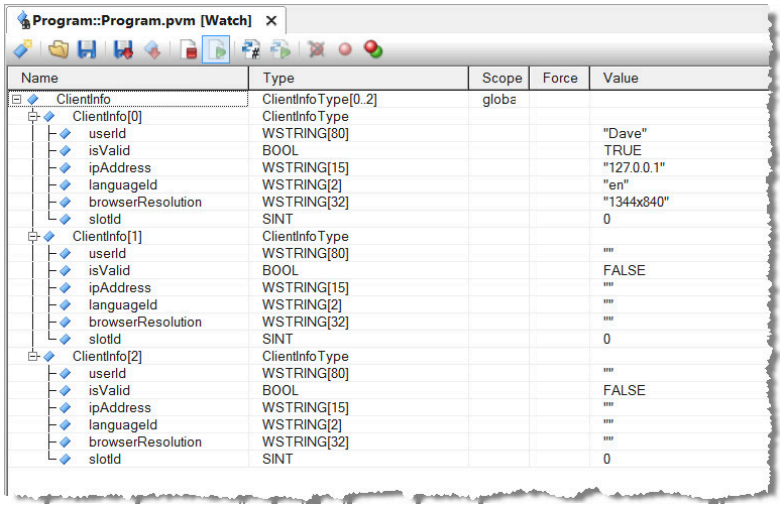


Figure 49: Client information for a connected client in the Watch window

6 Exercises with the text system

In the exercises for training manuals TM611 and TM671, the text system was already used several times to display localized texts on a widget in the selected language.

Snippets are used to embed formatted values as dynamic content in a text during operation.

HMI application / mapp View / Engineering / Variables and data / Session variables / Snippets

6.1 Displaying indexed texts

The goal of this exercise is to display a certain text from a list of texts when a value of a session variable changes. The texts are created in a text file (.tmx).

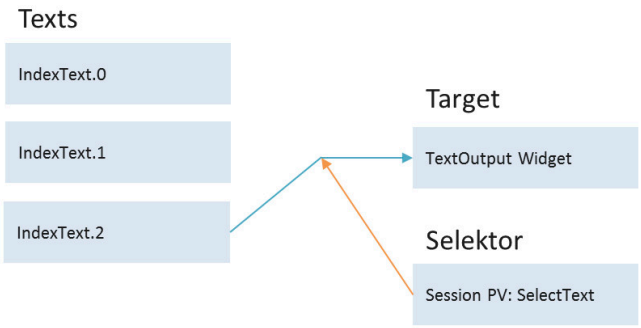


Figure 50: Schema for displaying indexed texts

Exercise: Display an indexed text

The following description shows how a certain text from a list of texts can be shown in a TextOutput widget as a result of a value change in session variable "SelectText".

The following steps are necessary for this:

1) Create a new session variable "SelectText" of the type "ANY_INT"
2) Configure a RadioButtonGroup with 3 RadioButton widgets in the ContentTestPage
3) Bind the session variable to the property "selectedIndex" of the RadioButtonGroup
4) Add a new text file (.tmx) in the Logical View
5) Reference the new text file in the text system configuration
6) Create three text IDs with a localized text each
7) Add a snippet file and create an IndexText snippet
8) Reference ID of the snippet in the HMI application (.vis)
9) Bind the session variable "SelectText" to the snippet
10) Bind the snippet to the TextOutput widget in the ContentTestPage

Steps 1 to 5 can now be carried out independently.

6.1.1 Configuring a text list in the text file

Add three text IDs to the new text file.

The text ID is entered with text and a value separated by a period. The text in front of the period must be identical for indexed texts in a group. The value represented by the selector is added after a period.

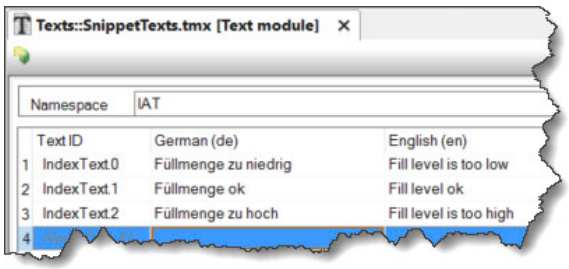


Figure 51: Text ID with continuous number for indexed texts

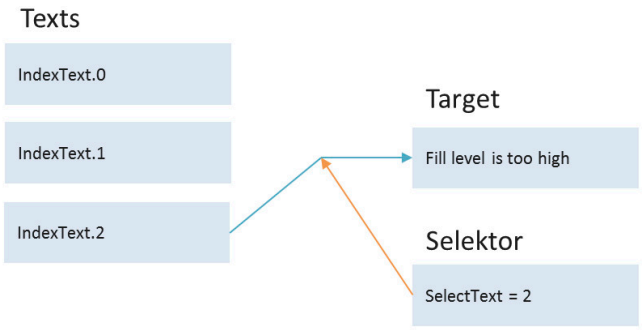
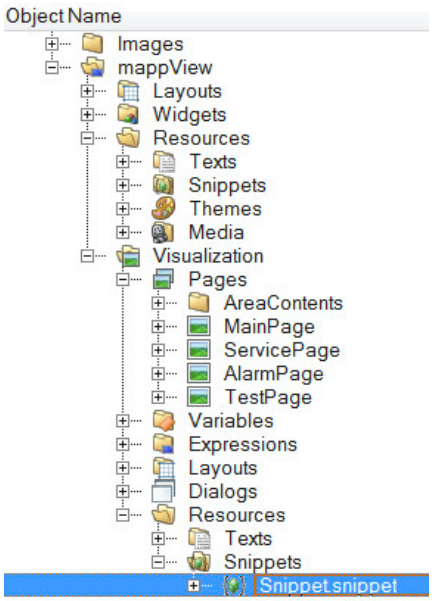


Figure 52: Displaying the text if SelectText has the value 2

6.1.2 Create a snippet



A snippet file from the Object Catalog is added to the Resources/Snippets node of the mapp View or visualization package.

Figure 53: Snippet file in the Logical View

After opening the file, a unique SnippetSet ID with the name "mySnippets" is added. This ID must be referenced in the element <SnippetsSets> of the visualization object (.vis).

```
<SnippetsSets>
     <SnippetsSet refId="mySnippets"/>
</SnippetsSets>
```

Then a snippet with the snippet ID="TextSnippet" of the type "IndexText" is added.

```
<Snippet id="TextSnippet" xsi:type="session" type="IndexText"
 formatItem="IndexText.{1}" />
```

In the attribute "formatItem", the Text ID from the previously created text file is transferred, whereby placeholder \{1\} is used for the value. When resolving the snippet during operation, the ID is calculated with the current value and the text is read from the text system.

A finished snippet file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<SnippetsSet id="mySnippets"
 xmlns="http://www.br-automation.com/iat2015/snippet/engineering/v3"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Snippets>
        <Snippet id="TextSnippet" xsi:type="session" type="IndexText"
            formatItem="IndexText.{1}" />
      </Snippets>
</SnippetsSet>
```

6.1.3 Binding of the session variable to the snippet – Snippet to the TextOutput widget

Session variable "SelectText" is connected to the snippet so that the snippet gets a value during operation. This occurs in an existing binding file.

The finished bindings look as follows:

Binding the RadioButtonGroup to the session variable

```
<Binding mode="twoWay">
    <Source xsi:type="session" refId="SelectText" attribute="value" />
    <Target xsi:type="brease" widgetRefId="RadioButtonGroup1"
        contentRefId="ContentTestPage" attribute="selectedIndex" />
</Binding>
```

Binding the session variable to the IndexText snippet

```
<Binding mode="oneWay">
    <Source xsi:type="session" refId="SelectText" attribute="value" />
    <Target xsi:type="snippet" refId="TextSnippet" attribute="value" />
</Binding>
```

Binding the resolved snippet text to the TextOutput widget

```
<Binding mode="oneWay">
    <Source xsi:type="snippet" refId="TextSnippet" attribute="value" />
    <Target xsi:type="brease" widgetRefId="TextOutput1"
    contentRefId="ContentTestPage" attribute="value" />
</Binding>
```

Expected result

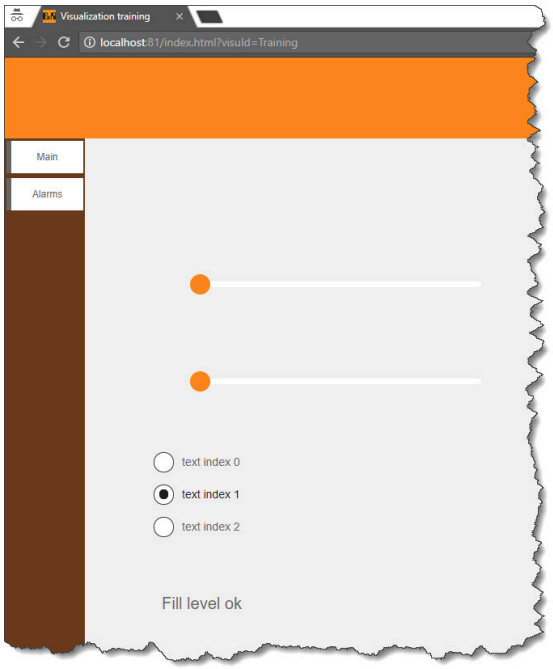


Figure 54: A change to the value leads to a change of the text from the text system.

Additional exercise: Output value within a localized text for a message box

HMI application / mapp View / Guides / FAQ / Event and action applications / MessageBox: Displaying a numeric value in the message

7 Multi-client – Multi-user

Different users require personalized content. mapp View allows you to implement role-based views without having to program them into the machine application.

These customized pages can even be viewed on different devices simultaneously. As previously shown in the exercise for session variables and in the system variables, the same HMI application can be carried over to more than one client.

It's not recommended to carry over the machine HMI application to mobile end devices because operation with a tablet, for example, usually corresponds to a specific role e.g. a service technician.

mapp View provides the option of creating standalone HMI applications (.vis) with the pages specific to the case of application, which makes it possible to reuse existing contents in the pages.

This exercise shows how several clients can be connected to an HMI application. The number of clients that are allowed to be connected simultaneously to the mapp View server is defined in the mapp View configuration. It's irrelevant if an HMI application is carried over to the client.

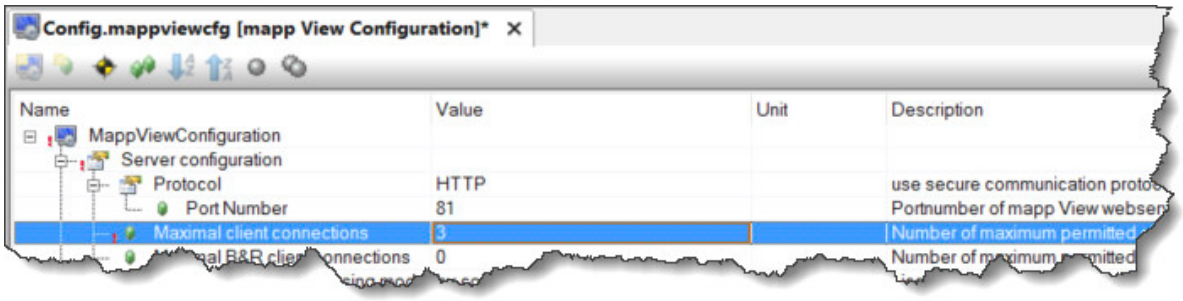


Figure 55: Allowing 3 client connections

The maximum number of client connections permitted must match the array length of the OPC UA structure with the client information.

In the previous exercises, the HMI application was shown in a browser window.

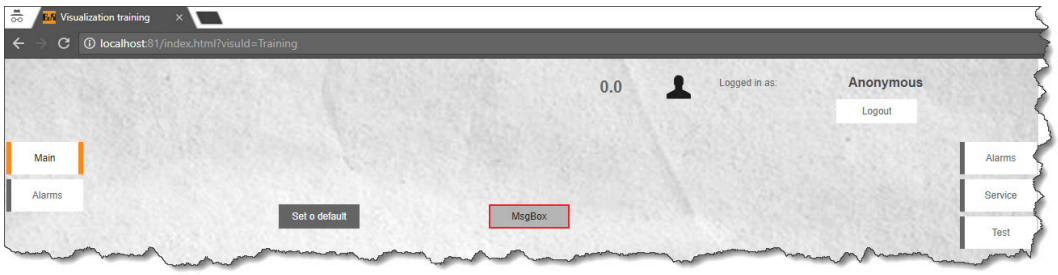


Figure 56: Displaying the HMI application in the browser

If the browser is opened for a second time with the same URL for testing a second client connection, then it is rejected by the mapp View server because the session is already "occupied" by this browser type. It doesn't matter if two windows (tabs) are used in a browser or if two browser instances are started.

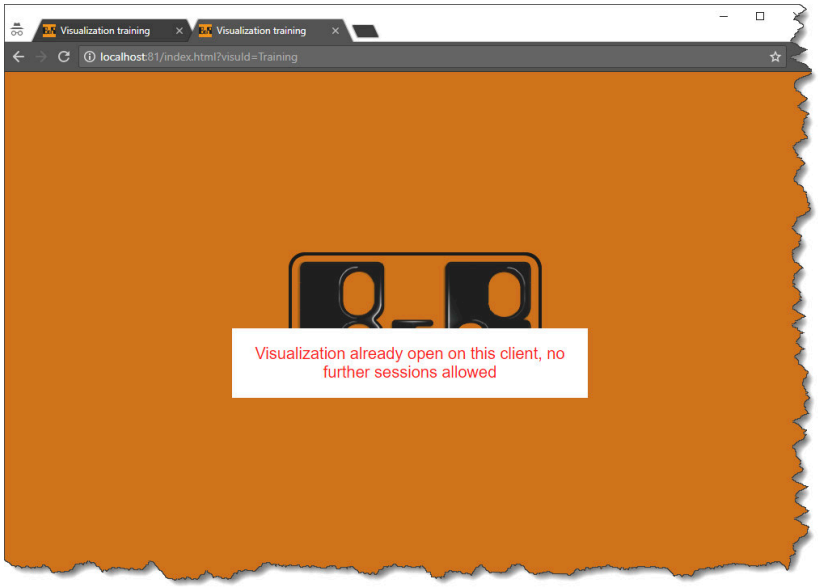


Figure 57: An error message if the browser connects to the same session

Exercise: Display HMI application in two client instances

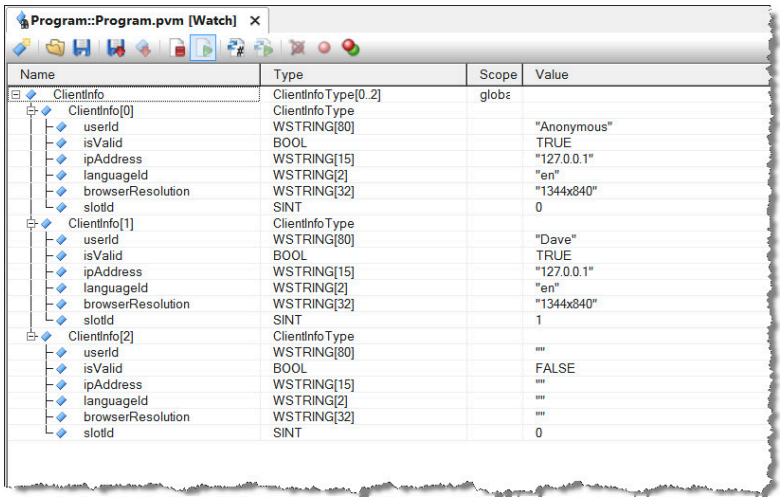
To test the HMI application with two clients, some browsers provide the option of opening a second instance in "incognito mode". In this case a new connection (session) is generated on the mapp View server.

Expected result

Two client instances can be displayed in each browser window and operated individually. A different user can be logged in on each client. Even the language and the system of measurement can be selected individually on each client.

Figure 58: Displaying the HMI application with two client instances

Values from two client connections are now contained in the OPC UA structure, which maps the client information.



Name	Type	Scope	Value
ClientInfo	ClientInfoType[0..2]	globa	
ClientInfo[0]	ClientInfoType		
userId	WSTRING[80]		"Anonymous"
isValid	BOOL		TRUE
ipAddress	WSTRING[15]		"127.0.0.1"
languageId	WSTRING[2]		"en"
browserResolution	WSTRING[32]		"1344x840"
slotId	SINT		0
ClientInfo[1]	ClientInfoType		
userId	WSTRING[80]		"Dave"
isValid	BOOL		TRUE
ipAddress	WSTRING[15]		"127.0.0.1"
languageId	WSTRING[2]		"en"
browserResolution	WSTRING[32]		"1344x840"
slotId	SINT		1
ClientInfo[2]	ClientInfoType		
userId	WSTRING[80]		""
isValid	BOOL		FALSE
ipAddress	WSTRING[15]		""
languageId	WSTRING[2]		""
browserResolution	WSTRING[32]		""
slotId	SINT		0

Figure 59: Client information for a connected client in the Watch window

8 Dynamic graphics

Many HMI applications require process images and graphics to change depending on the condition and status of the machine application. mapp View offers the "Paper" widget for this purpose. It provides a graphical area for displaying and modifying elements of SVG[1] graphics.

When using the "Paper" widget, it is checked at runtime to ensure that the required license is available.

Exercise: Using the "Paper" widget

The goal of this exercise is to use the "Paper" widget to animate a graphic in a mapp View HMI application. An SVG graphic representing a rotating disc is used. There is an element in the SVG file that is set to rotate using a transformation.

1) Copy SVG graphic "MotorShaft.svg" to the media package
2) Position the "Paper" widget
3) Link the SVG graphic with the "Paper" widget
4) Activate process variable "transformation" on the OPC UA server

   The element should be able to rotate around its center. The following transformation can be used for this purpose.

```
transformation:=
'[{"select":"#Shaft","duration":3000,"spin":[720,15,30]}]';
```

5) Bind the OPC UA values to the "transformation" property using the "Paper" widget

Create an SVG graphic

More detailed information about creating SVG graphics can be found on w3schools.com.

(see **www.w3schools.com/graphics/svg_intro.asp**)

Visualization \ mapp View \ Widgets \ Drawing \ Paper

Figure 60: Example of a rotating disc

[1] Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. The SVG specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999 (Source: en.wikipedia.org).

The SVG file "MotorShaft.svg" can be checked by opening it in an external text editor.

Possible transformations

Supported transformations are briefly explained in Automation Help.

mapp View / Widgets / Drawing / Paper / Concept

Configure the "Paper" widget

Now the "Paper" widget is added to a "ContentTestPage" content.

Then the "Paper" widget is configured. During this process, the path of the SVG file and the transformation string are linked to the "Paper" widget.

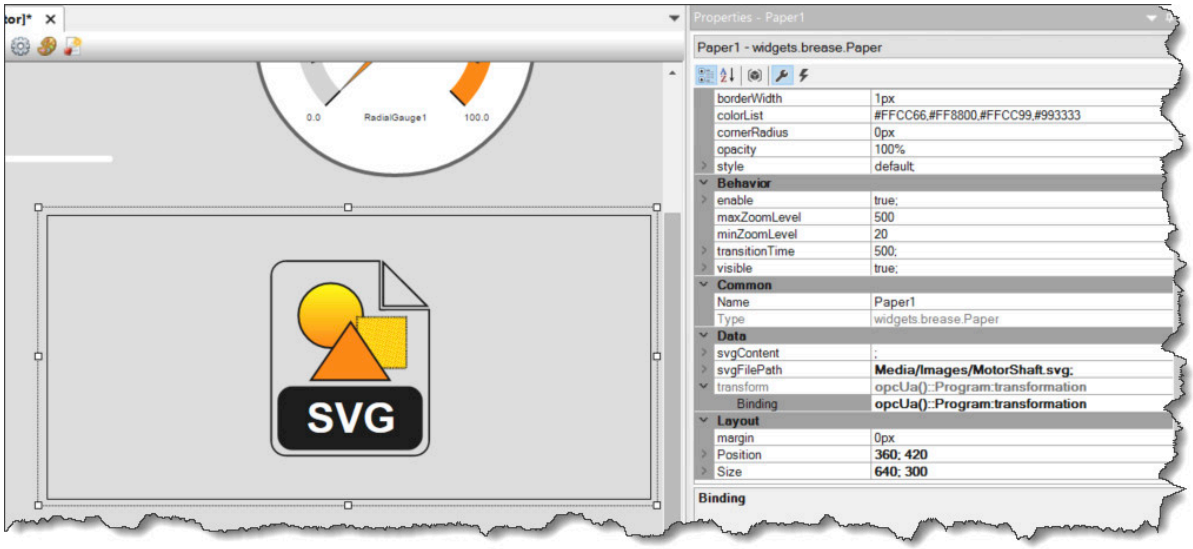


Figure 61: Configuration of the "Paper" widget

A transformation is enabled each time process variable "transformationIndex" changes. The "Paper" widget executes it, thereby animating the graphic.

9 Summary

The event and action system and HMI variables can also be used to implement complex HMI concepts. With the help of the tasks and exercises in this training module, you're provided the opportunity to significantly expand your basic knowledge of mapp View. Additional information, configurations and the combination of various widget applications allow many new concepts to be utilized when designing an HMI application.

The instructions in this training module serve to provide deeper understanding and offer an overview of possible expansions to a mapp View HMI application. The references to the help documentation included in this training module show where additional information related to configurations can be found.

Offered by the Automation Academy

The Automation Academy provides targeted training courses for our customers as well as our own employees.
At the Automation Academy, you'll develop the skills you need in no time!
Our seminars make it possible for you to improve your knowledge in the field of automation engineering. Once completed, you will be in a position to implement efficient automation solutions using B&R technology. This will make it possible for you to secure a decisive competitive edge by allowing you and your company to react faster to constantly changing market demands.

Seminars

Quality and relevance are essential components of our seminars. The pace of a specific seminar is based strictly on the experience that course participants bring with them and tailored to the requirements they face. A combination of group work and self-study provides the high level of flexibility needed to maximize the learning experience.
Each seminar is taught by one of our highly skilled and experienced trainers.

Training modules

Our training modules provide the basis for learning both at seminars as well as for self-study. These compact modules rely on a consistent didactic concept. Their bottom-up structure allows complex, interrelated topics to be learned efficiently and effectively. They serve as the best possible companion to our extensive help system. The training modules are available as downloads and can be ordered as printed versions.

Topic categories:

- ⇨ Control technology
- ⇨ Motion control
- ⇨ Safety technology
- ⇨ HMI
- ⇨ Process control
- ⇨ Diagnostics and service
- ⇨ POWERLINK and openSAFETY

ETA system

The ETA system provides realistic constructions for training, education and laboratory use. Two different basic mechanical constructions can be selected. The ETA light system offers a high degree of mobility, saves space and is well-suited for lab work. The ETA standard system has a sturdy mechanical structure and includes pre-wired sensors and actuators.

Find out more!

Would you like additional training? Are you interested in finding out what the B&R Automation Academy has to offer? You've come to the right place.
Detailed information can be found under the following link:
www.br-automation.com/academy