

TM671

Creating efficient map View HMI applications



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. mapp View 1.2 Automation Studio 4.2.7 |
| Hardware | ARsim |

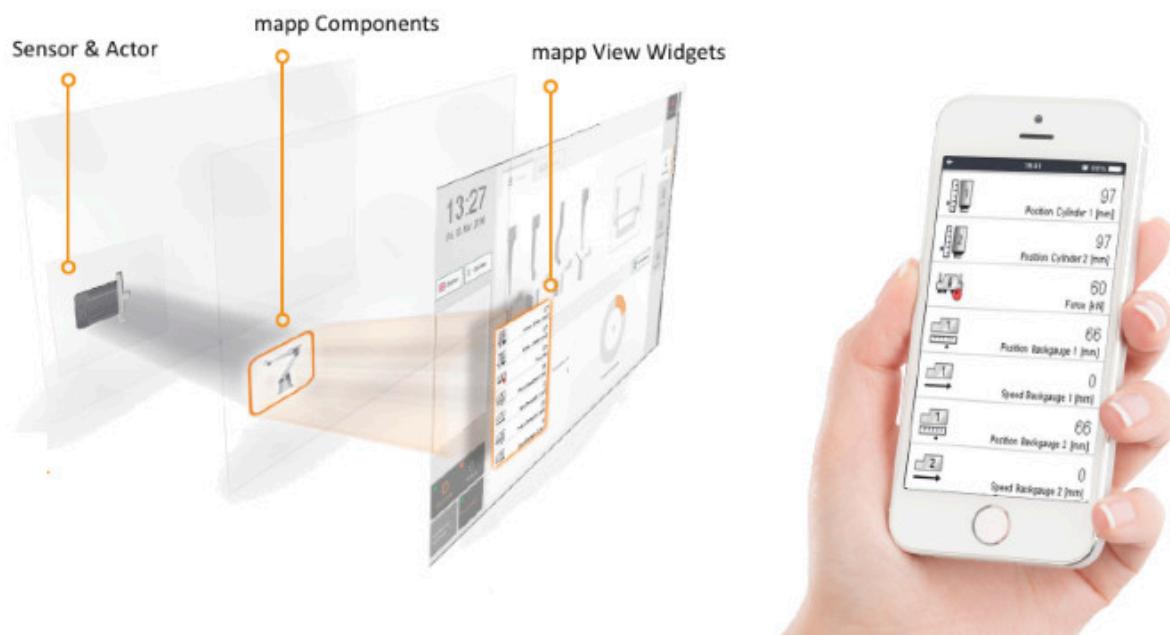
Table of contents

| | | |
|-----|--|----|
| 1 | Introduction..... | 4 |
| 1.1 | Learning objectives..... | 4 |
| 1.2 | Symbols and safety notices..... | 4 |
| 2 | Dialog and message boxes..... | 5 |
| 2.1 | Overview of dialog and message boxes..... | 5 |
| 2.2 | Creating and opening a dialog box..... | 5 |
| 2.3 | MessageBox overview..... | 11 |
| 2.4 | Open message box and evaluate confirmation..... | 12 |
| 3 | Using roles and rights on widgets..... | 17 |
| 3.1 | Overview of roles and rights on widgets..... | 17 |
| 3.2 | Role-based visibility and usability of widgets..... | 17 |
| 4 | Animations in the HMI application..... | 21 |
| 4.1 | Simple visibility control for widgets..... | 21 |
| 4.2 | Visibility control calculated using multiple process values..... | 23 |
| 4.3 | Background image for a page..... | 27 |
| 4.4 | Changing the color of widgets..... | 30 |
| 4.5 | Relative and absolute positioning in Container widgets..... | 33 |
| 4.6 | Scaling the HMI application to the size of the browser window..... | 35 |
| 5 | Tasks with different variable types..... | 37 |
| 5.1 | Exchange information with different widgets..... | 38 |
| 5.2 | Changing users and switching the page..... | 44 |
| 5.3 | Using system variables..... | 45 |
| 6 | Exercises with the text system..... | 48 |
| 6.1 | Displaying indexed texts..... | 48 |
| 7 | Content engineering in XML..... | 53 |
| 7.1 | Adding widgets..... | 53 |
| 7.2 | Adding Container widgets..... | 54 |
| 8 | Multi-client – Multi-user..... | 55 |
| 8.1 | Client-specific OPC UA binding..... | 58 |
| 9 | Summary..... | 62 |

Introduction

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 get an overview of content engineering in XML files.
- 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.

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 shown. The dialog box is opened by clicking on the image in the header.

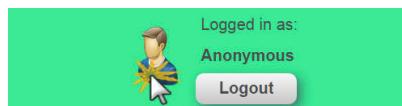


Figure 2: Open the dialog box by clicking on the image

The following steps are necessary for this:

- 1) Create a layout
- 2) Create content
- 3) Create a dialog box

Dialog and message boxes

- 4) Reference dialog boxes in the visualization object
- 5) Event binding for opening the dialog box
- 6) Event binding for closing the dialog box



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

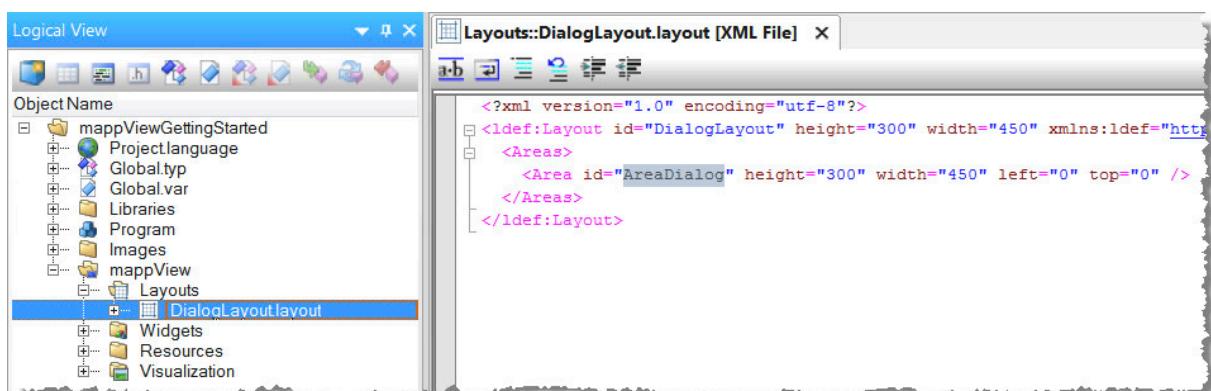


Figure 3: Add the layout file in Logical View and edit the XML file

Final result

```
<?xml version="1.0" encoding="utf-8"?>
<ldef:Layout id="DialogLayout" height="300" width="450"
xmlns:ldef="http://www.br-automation.com/iat2015/layoutDefinition/v2">
    <Areas>
        <Area id="AreaDialog" height="300" width="450" left="0" top="0" />
    </Areas>
</ldef:Layout>
```

2.2.2 Creating content

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

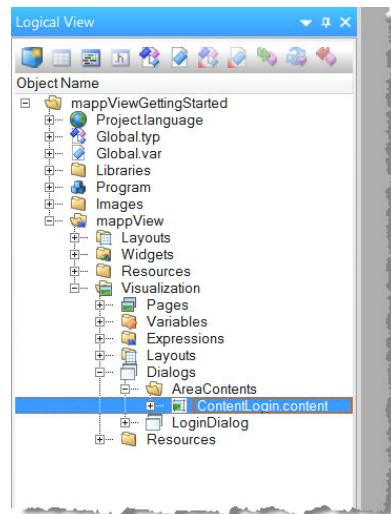


Figure 4: Content file added

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.

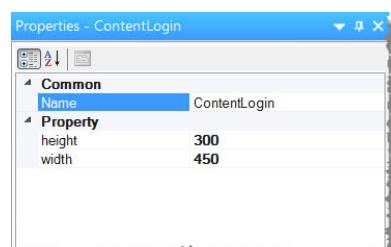


Figure 5: ContentLogin properties

Dialog and message boxes

A login widget and a button widget have to be placed in the content. The dialog box should be closed by selecting the button labeled "ButtonClose".

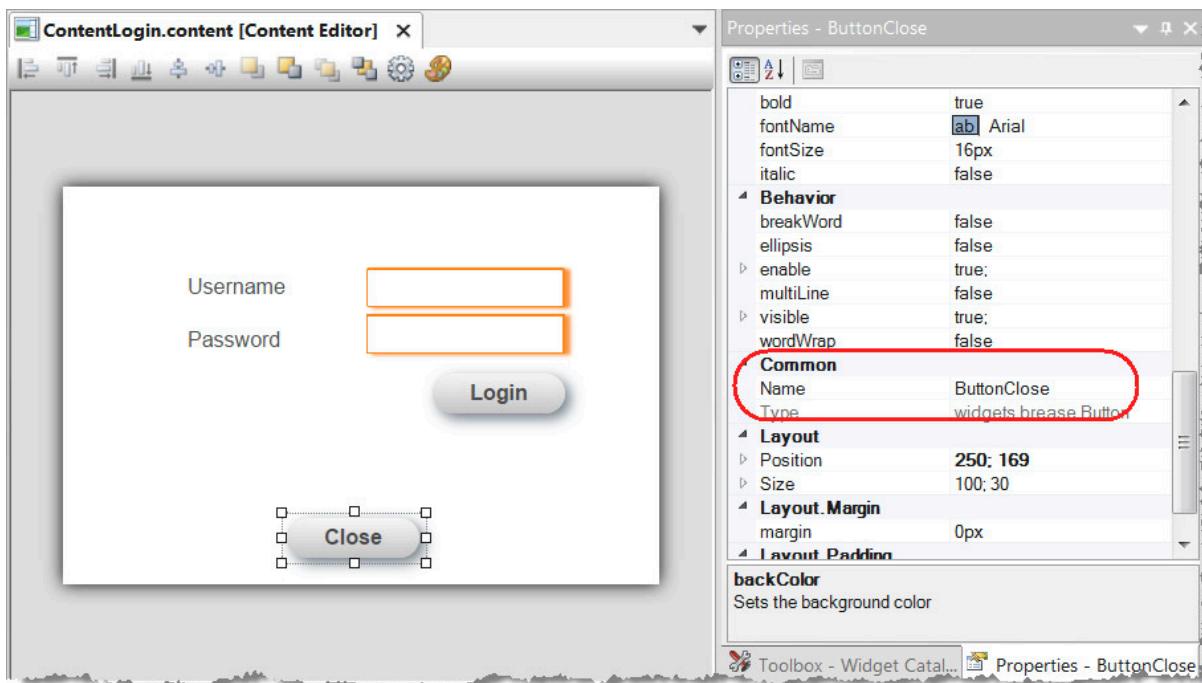
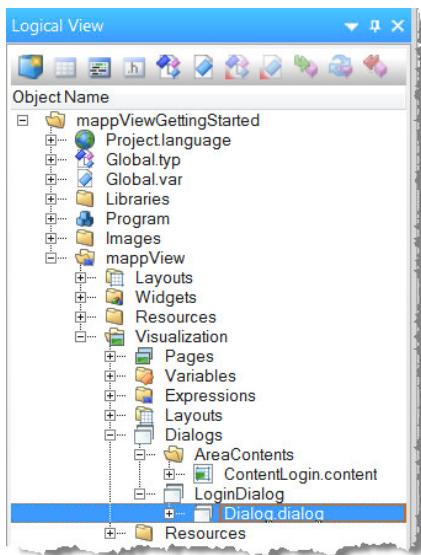


Figure 6: Widgets configured in the content

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.

The following areas and pieces of content are used to design the "LoginDialog":

| type | baseContentRefId | areaRefId |
|-------------|-------------------------|------------------|
| Content | ContentLogin | AreaDialog |

Table 3: Properties for the dialog box

The following XML code has to be entered in the "LoginDialog.dialog" file:

```
<?xml version="1.0" encoding="utf-8"?>
<ddef:Dialog id="LoginDialog" layoutRefId="DialogLayout"
    xmlns:ddef="http://www.br-automation.com/iat2015/dialogDefinition/v2">
    <Assignments>
        <Assignment type="Content" baseContentRefId="ContentLogin"
            areaRefId="AreaDialog" />
    </Assignments>
</ddef:Dialog>
```

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 transferred 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

A new <EventBinding> is created in the available event binding file. As a source, the click event of the "Image" widget placed in the ContentTop is used for displaying a user.



If the widgetRefId of the "Image" widget placed in the content is unknown, it has to be checked by opening the corresponding piece of content.

The finished entry in the example for clicking on an "Image" widget looks as follows:

```
<Source xsi:type="widgets.brease.Image.Event"
contentRefId="ContentTop" widgetRefId="Image1" event="Click" />
```

After the event has been defined, the subsequent action must be defined. The possible attributes are described in Automation Help.

The concrete action for opening a dialog box looks as follows:

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

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

Dialog and message boxes

```
<EventBinding>
    <Source xsi:type="widgets.brease.Image.Event" contentRefId="ContentTop"
        widgetRefId="Image1" event="Click"/>
    <EventHandler>
        <Action>
            <Target xsi:type="clientSystem.Action">
                <Method xsi:type="clientSystem.Action.OpenDialog"
                    dialogId="LoginDialog" />
            </Target>
        </Action>
    </EventHandler>
</EventBinding>
```

Expected result

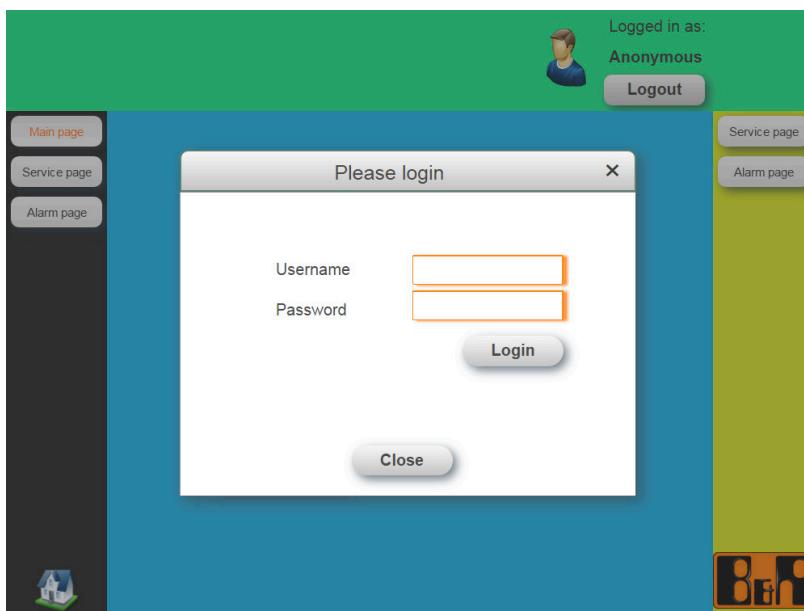


Figure 8: HMI application in the browser with an opened dialog box after clicking on the user image

2.2.6 Event binding for closing the dialog box

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

As a source, the click event of the Button widget placed in the ContentLogin with the name "ButtonClose" is used to close the dialog box. The finished entry in the example for clicking on a "Button" widget looks as follows:

```
<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.



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

The action for closing the dialog box looks as follows:

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

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

```
<EventBinding>
    <Source xsi:type="widgets.breeze.Button.Event"
        contentRefId="ContentLogin" widgetRefId="ButtonClose" event="Click"/>
    <EventHandler>
        <Action>
            <Target xsi:type="clientSystem.Action">
                <Method xsi:type="clientSystem.Action.CloseDialog"
                    dialogId="LoginDialog"/>
            </Target>
        </Action>
    </EventHandler>
</EventBinding>
```

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 9: 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.

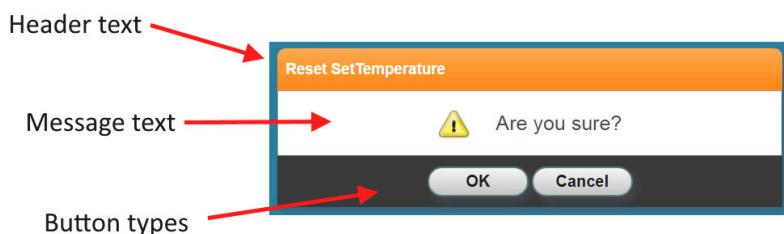


Figure 10: Areas of a message box

Dialog and message boxes

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 4: 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 |
| Retry | 32 |
| Ignore | 64 |

Table 5: 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 Content MainPage to open the message box
- 2) Create an event binding
- 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 / Client system actions / ShowMessageBox

2.4.1 Button widget for opening the message box

The "Button" widget placed in the Content MainPage 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.

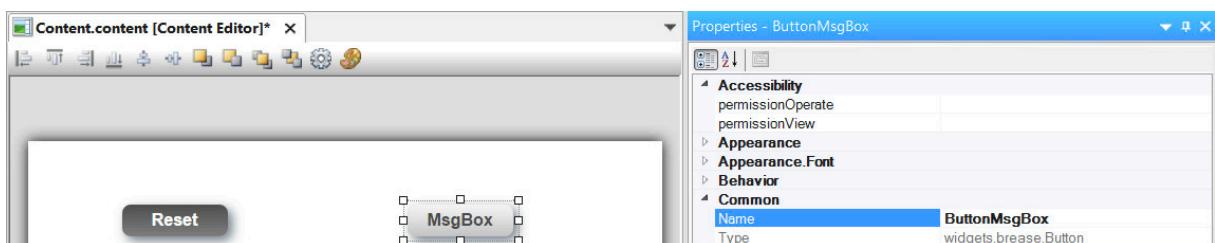


Figure 11: Button for opening a message box

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 Content MainPage with the name "ButtonMsgBox" is used for opening the message box.

The finished entry for clicking on a "Button" widget looks as follows:

```
<Source xsi:type="widgets.brease.Button.Event"
contentRefId="Content MainPage" widgetRefId="ButtonMsgBox" 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.



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

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

Dialog and message boxes

```
<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 ">::AsGlobalPV: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.

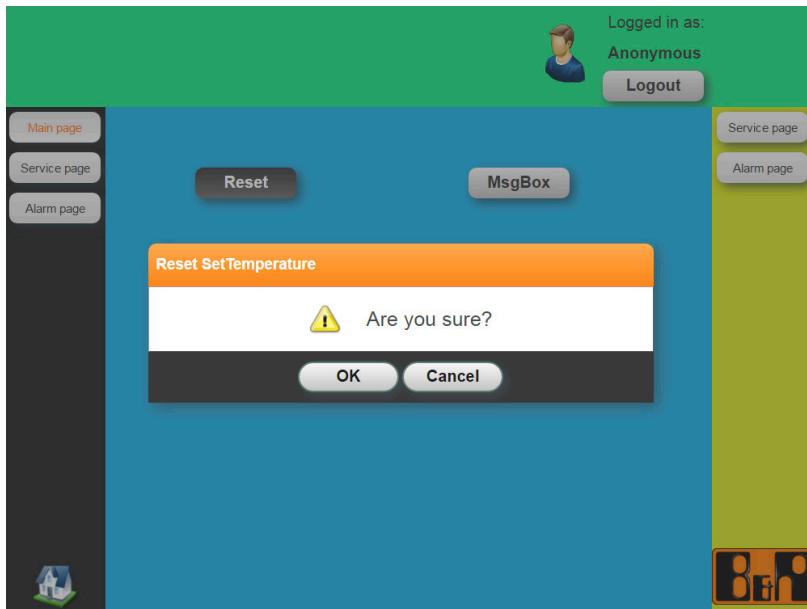


Figure 12: The message box is displayed in the foreground

Dialog and message boxes

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 13: 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.

| Name | Value | Value | Value | Value |
|---------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------|
| SetTemperature | | | | |
| Enable | True | | | |
| Audit Events | Off | | | |
| Authorization | | | | |
| Inherit From Parent | Inherit None | | | |
| Rights / Roles | Everyone | Operator | Service | |
| Visible | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | |
| Browse | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | |
| Subscribe | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | |
| Read | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | |
| Write | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | |
| EU Range Violation | | | | |

Figure 14: Restricted write access of an OPC UA variable for the role "Everyone"



[mapp View / Guidelines / 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 one 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.

Using roles and rights on widgets

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.

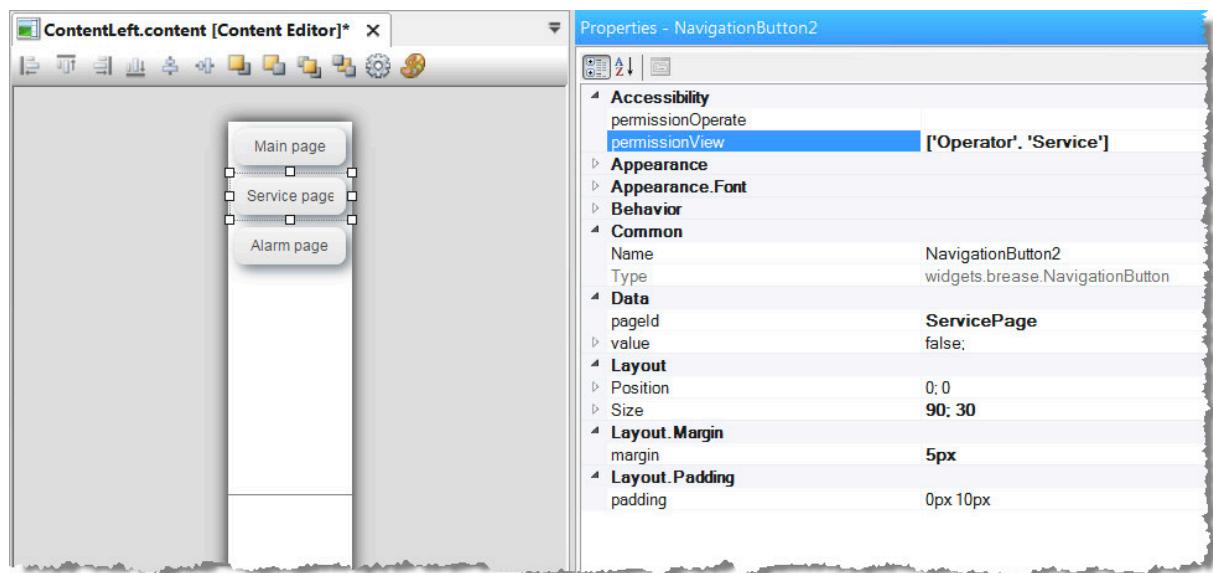


Figure 15: Configure the visibility controller on the property permissionView



Visibility cannot be restricted for NavigationButtons of the automatic navigation.

Expected result

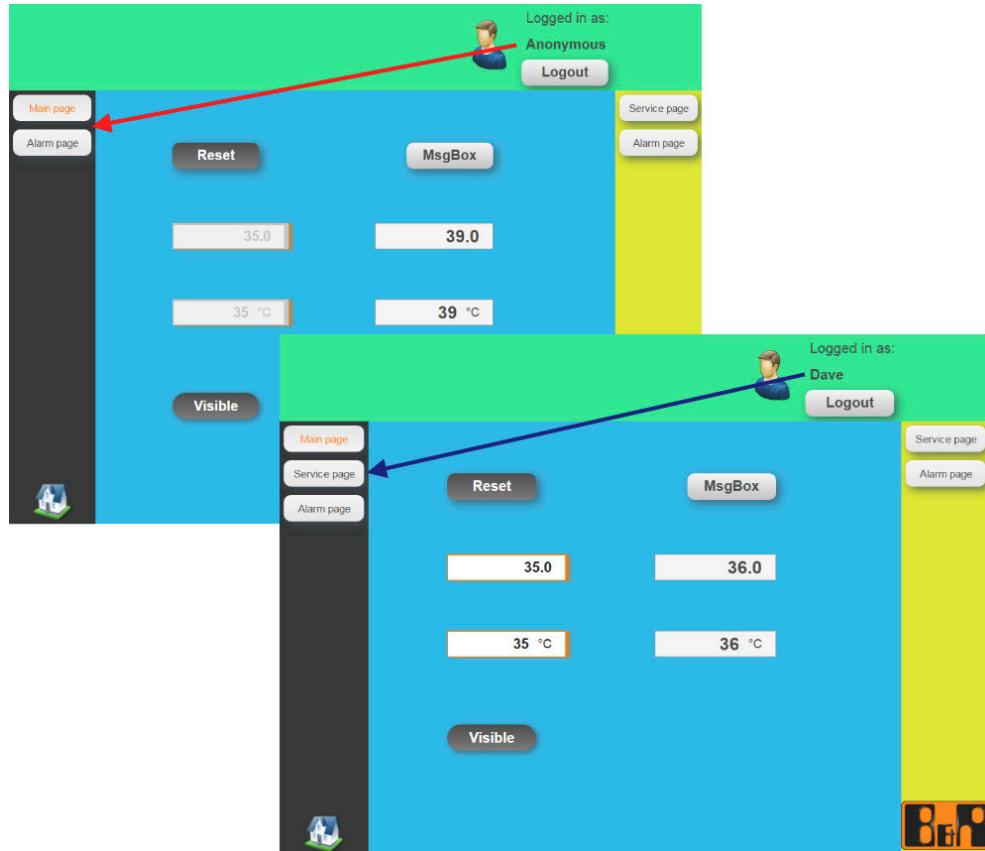


Figure 16: NavigationButton is only visible to users assigned the "Service" and "Operator" role

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".

Using roles and rights on widgets

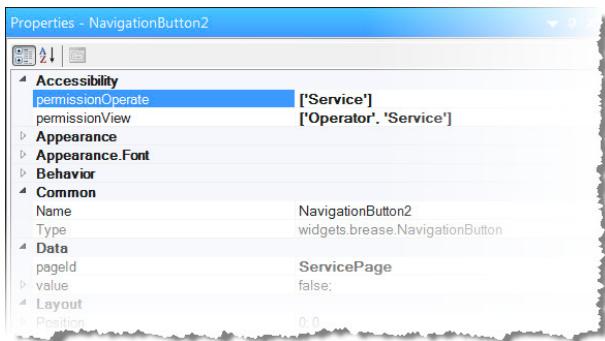


Figure 17: Access rights configuration on the property permissionOperate

Expected result:

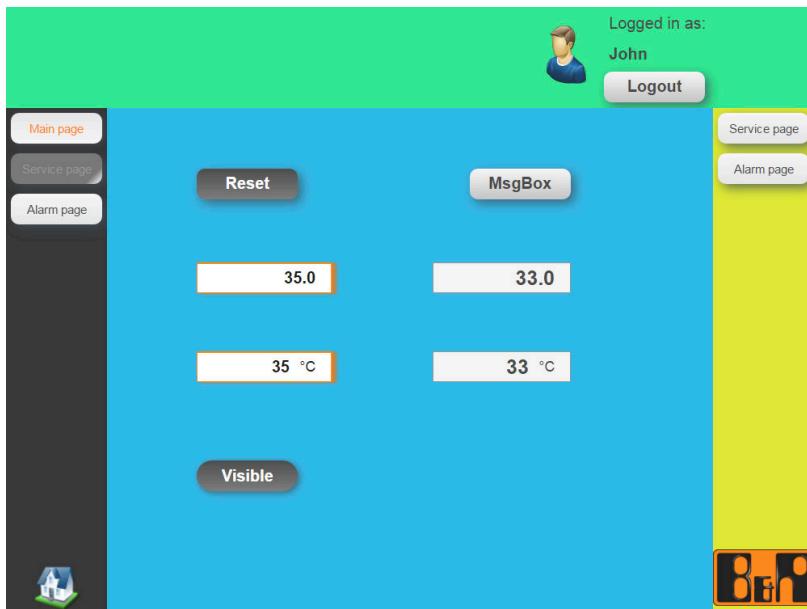


Figure 18: Users assigned the role "Operate" can see the NavigationButton but cannot use it

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.



mapp View / Guidelines / 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) In the Init subprogram "Program", add a variable "State1" of data type "BOOL"
- 2) Enable this variable as an OPC UA variable
- 3) Bind the OPC UA variable to a ToggleButton widget in the content "Content MainPage"
- 4) Bind the OPC UA variable to the visible property of a NumericOutput widget



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

Animations in the HMI application

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".

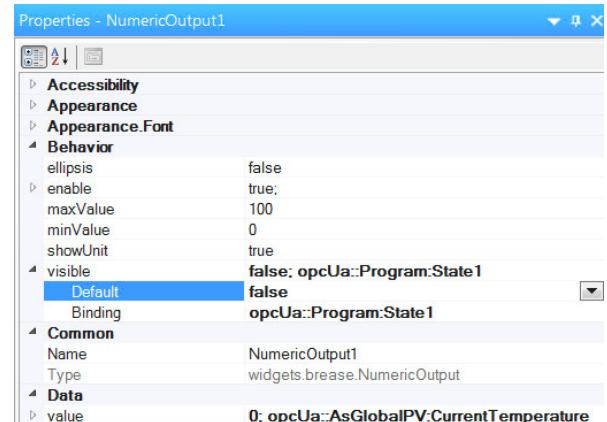


Figure 19: Visible binding of an OPC UA variable

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.

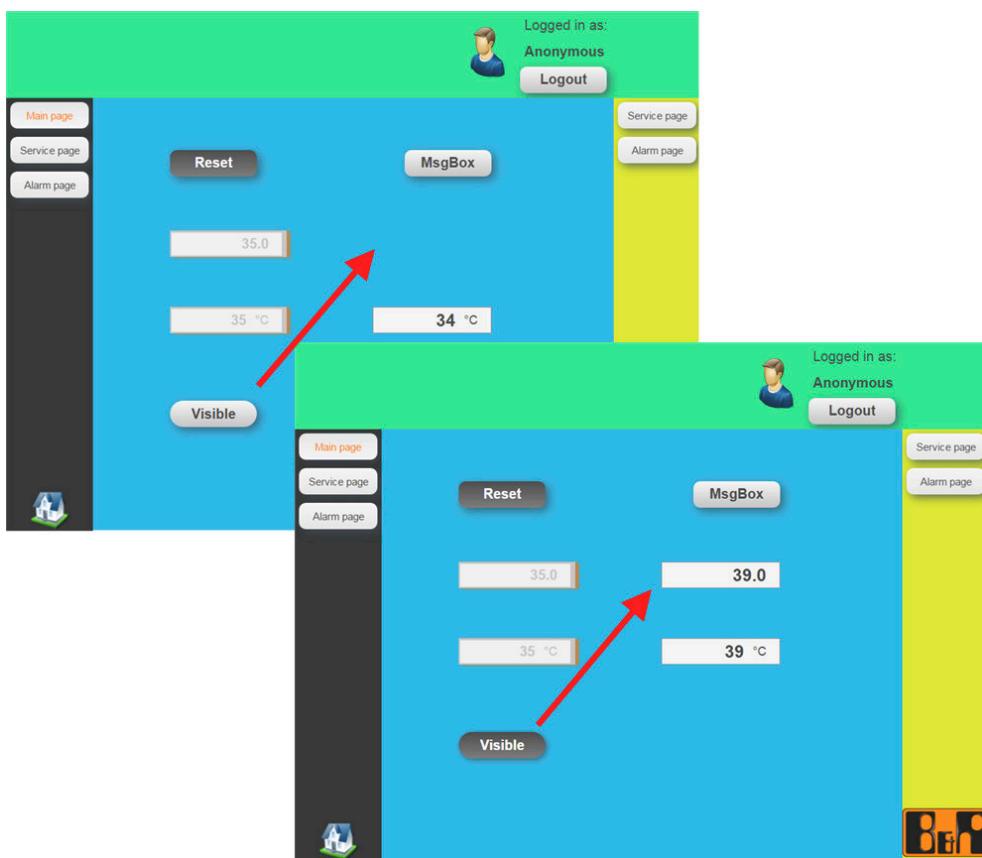


Figure 20: Visibility of the widget with a pressed or released ToggleButton

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)



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:

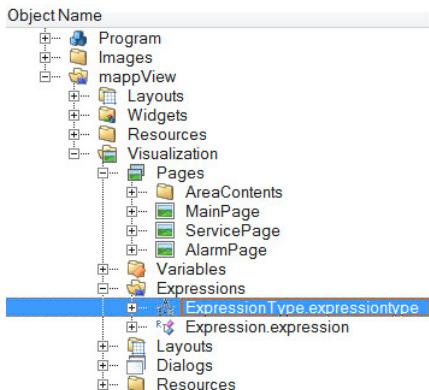
- 1) In the Init subprogram "Program", add variables "State1" and "State2" of data type "BOOL" and "State3" of data type "Integer".
- 2) Enable these variables as OPC UA variables
- 3) Bind the OPC UA variables "State1" and "State2" each to a ToggleButton widget in the content "ContentMainPage"
- 4) Bind the OPC UA variable "State3" to a BasicSlider widget in the content "ContentMainPage"
- 5) Insert 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.

Animations in the HMI application

4.2.1 Configure expression type



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.

Figure 21: ExpressionType and expression file in the Logical View

After opening the file, a unique ExpressionTypeSet ID with 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_INT" />
</Operands>
<Operation>
    (State3 > 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_INT" />
            </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 ExpressionsSet 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/XMLSchema-instance">
    <Expressions>
    </Expressions>
</ExpressionsSet>

```

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

```

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

```

To finish, the expression itself is also added under <Expressions>.

```

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

```

The finished configuration of the expression looks like this:

Animations in the HMI application

```
<?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. The finished 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. The binding occurs manually in a binding file. The source of the type "expression" and of the attribute "result" is connected to the corresponding widget property.

```

<Binding mode="oneWay">
    <Source xsi:type="expression" refId="myExprResult" attribute="result" />
    <Target xsi:type="brease" contentRefId="ContentMainPage"
        widgetRefId="NumericOutput2" attribute="visible" />
</Binding>

```

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.

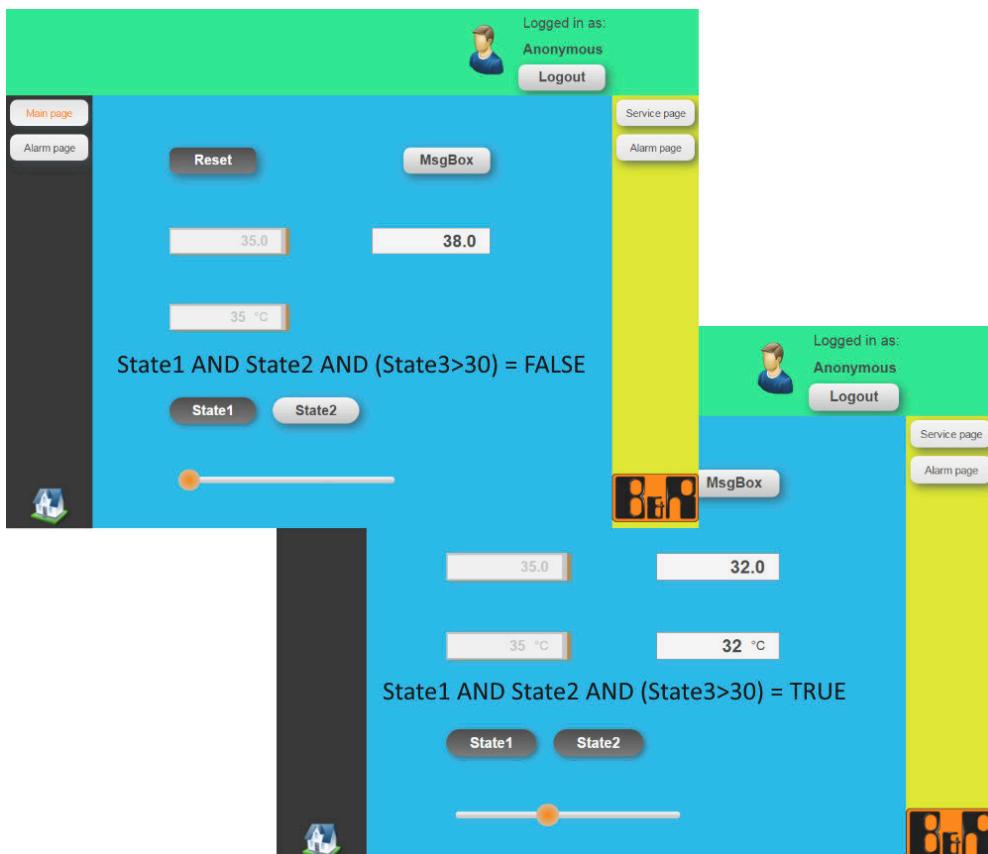


Figure 22: Visibility of the NumericOutput widget based on the expression result

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.

```

<Assignment type="Content" baseContentRefId="Content MainPage"
areaRefId="AreaMain" backColor="#2CB9E8" />

```

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:

Animations in the HMI application

- 1) Select an image with a page ratio of 4:3
- 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 by the attribute backGround="Media/imageName".

```
<pdef:Page id="MainPage" layoutRefId="MyLayout"  
    xmlns:pdef="http://www.br-automation.com/iat2015/pageDefinition/v2"  
    displayName="$IAT/MainPage"  
    backGround="Media/BackGround.png">
```

If the project is transferred, no effect is shown yet since the color of the area appears on the image. Only by setting a transparent background color with backColor="transparent" is the image also shown during operation.

```
<Assignments>  
    <Assignment type="Content" baseContentRefId="ContentMainPage"  
        areaRefId="AreaMain" backColor="transparent"/>  
    <Assignment type="Content" baseContentRefId="ContentTop"  
        areaRefId="AreaTop" backColor="transparent"/>  
    <Assignment type="Content" baseContentRefId="ContentLeft"  
        areaRefId="AreaLeft" backColor="transparent"/>  
    <Assignment type="Content" baseContentRefId="ContentRight"  
        areaRefId="AreaRight" backColor="transparent"/>  
</Assignments>
```

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

Expected result



Figure 23: Display a background image on a page

4.3.2 Background image and area colors

In this exercise, the combination of using a background image and background color for an area is shown.

Exercise: Background image and area colors

The goal of this exercise is to define a half-transparent color in the header so that the image still shows through the page. An RGBA value on the property backColor must be used in order to use a color with a transparency (alpha channel).



mapp View / Engineering / Themes and styles / Color, gradient and shadow

The background color for the area is defined in the assignment for the AreaTop on the MainPage as follows:

```
<Assignment type="Content" baseContentRefId="ContentTop"  
areaRefId="AreaTop" backColor="rgba(51, 153, 255, 0.5)"/>
```

The last value for the color specification defines the transparency.

Animations in the HMI application

Expected result

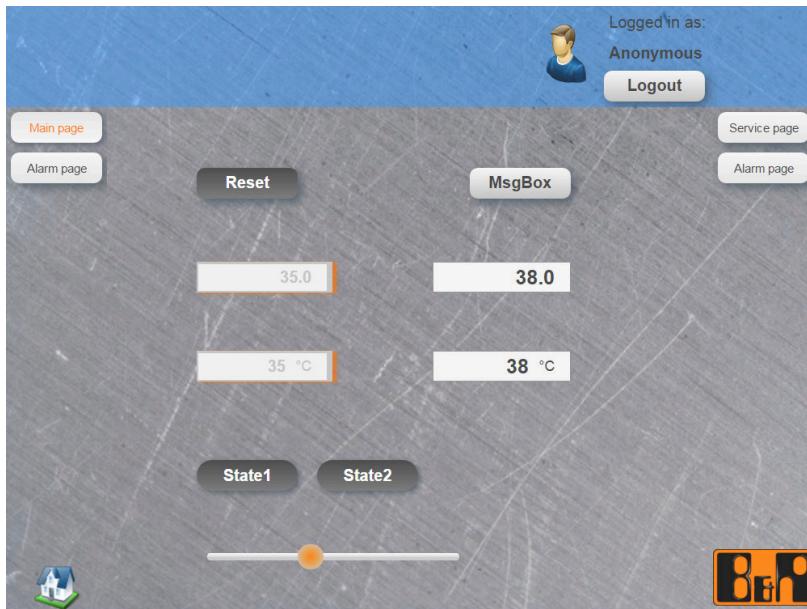


Figure 24: Transparent color of an area

4.4 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 25: 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.

4.4.1 Define the style for the "TRUE" state

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

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 3px 3px 2px"
- 4) Add the attribute borderColor="#FF0000" for a red frame

Finished result for the button style "Error":

```
<Style id="Error" xsi:type="widgets.breeze.Button"
fontSize="16px" bold="true" textColor="#474747"
mouseDownTextColor="#FFFFFF" cornerRadius="8px"
borderWidth="2px 3px 3px 2px" mouseDownBorderWidth="2px 3px 3px 2px"
borderColor="#FF0000"
backGroundGradient="linear-gradient(to bottom, #F7F7F7 0%, #C6C6C6 100%)"
mouseDownBackGroundGradient="linear-gradient(to bottom, #66CCFF 0%, #52A3CC 100%)"
disabledBackGroundGradient="linear-gradient(to bottom, #e2e2e2 0%, #dbdbdb
50%, #d1d1d1 51%, #fefefe 100%)"
shadow="4px 4px 14px 0px rgba(62, 79, 92, 0.75)"
mouseDownShadow="2px 2px 8px 0px rgba(62, 79, 92, 0.75)"/>
```

4.4.2 Event binding for changing the style

In this step the event binding establishes which state the OPC UA variable "::Program:State2" of the style "Error" or the style "Operate1" is shown.

```
<Source xsi:type="opcUa.Event" refId="::Program:State2" event="ValueChanged"/>
```

That style should be shown in the "FALSE" state of the variable that was also configured on the widget (Operate1) and in the "TRUE" state, which represents an error in the process sequence of the style "Error". The "TRUE" state is evaluated in the <EventHandler> by an additional condition condition="newValue" and that of the "FALSE" state with condition="NOT newValue".

A style is changed via the widget action "SetStyle".

```
<EventHandler condition="newValue">
    <Action>
        <Target xsi:type="widgets.breeze.Button.Action"
            contentRefId="ContentMainPage" widgetRefId="ButtonMsgBox" >
            <Method xsi:type="widgets.breeze.Button.Action.setStyle"
                value="Error" />
        </Target>
    </Action>
</EventHandler>
```

The finished event binding for switching a widget style looks like this:

Animations in the HMI application

```
<EventBinding>
    <Source xsi:type="opcUa:Event" refId="::Program:State2" event="ValueChanged"/>
    <EventHandler condition="newValue">
        <Action>
            <Target xsi:type="widgets.brease.Button.Action"
                contentRefId="ContentMainPage" widgetRefId="ButtonMsgBox" >
                <Method xsi:type="widgets.brease.Button.Action.setStyle"
                    value="Error" />
            </Target>
        </Action>
    </EventHandler>
    <EventHandler condition="NOT newValue">
        <Action>
            <Target xsi:type="widgets.brease.Button.Action"
                contentRefId="ContentMainPage" widgetRefId="ButtonMsgBox" >
                <Method xsi:type="widgets.brease.Button.Action.setStyle"
                    value="Operate1" />
            </Target>
        </Action>
    </EventHandler>
</EventBinding>
```

Expected result



Figure 26: Changing a style



When using OPC UA variables in the event binding, take note that a ValueChanged event that was executed once no longer exists when the client is restarted. This leads to the style change of the widget in this case not being updated.

As a solution for this, the OPC UA variable (source) can be connected to a session variable (target). There is always a reaction to the ValueChanged event for the session variable in the event binding because it is always updated by the binding.

see: ["Session variables" on page 41](#)

4.5 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 select 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 GroupBox with relative positioning to the content ContentServicePage
- 2) Add the transparent GroupBoxes as child boxes for the first GroupBox (= line)
- 3) Add widgets to the respective GroupBoxes
- 4) Bind the OPC UA variable "::Program:State2" to the visible property of the first child GroupBox

4.5.1 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 | 10;10 |
| Size | 580;270 |

Table 6: Properties of the GroupBox

Animations in the HMI application

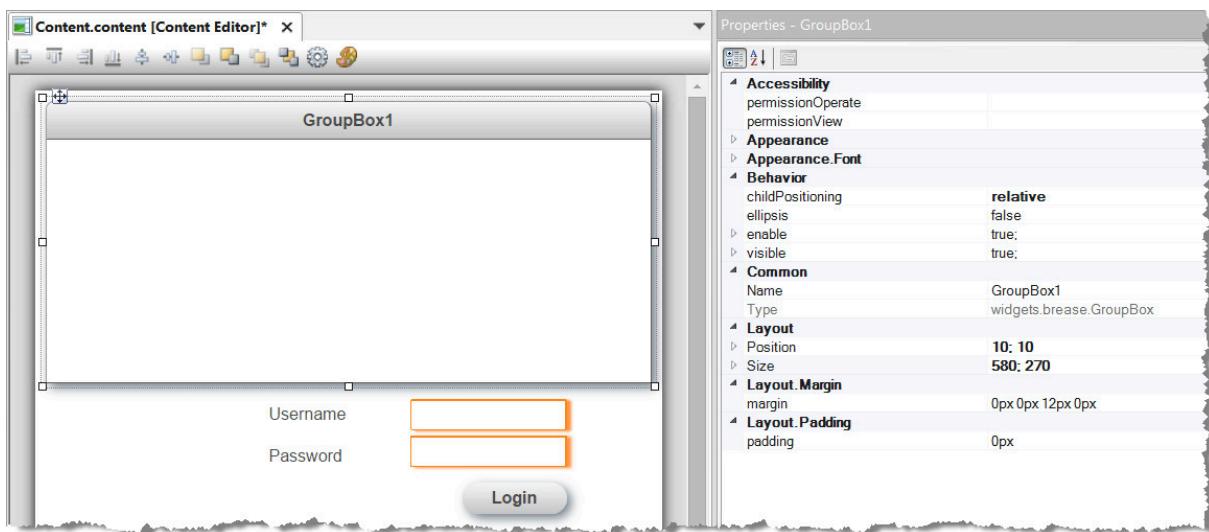


Figure 27: Parent GroupBox with relative positioning

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 7: 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.

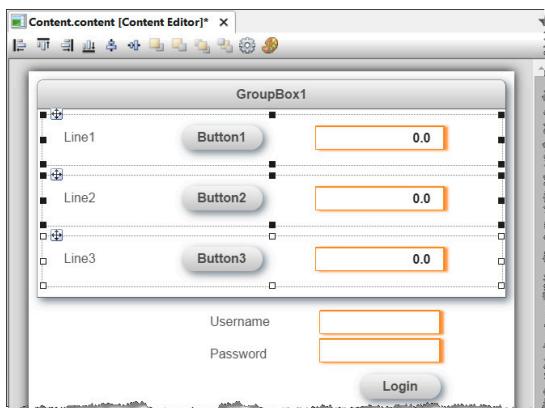


Figure 28: Completely configured GroupBoxes – Display with multi-select

Subsequently, the first child GroupBox is configured as a value binding to the OPC UA variable ">::Program:State2" via the visible property.

Expected result

If the ToggleButton for changing the OPC UA variable ">::Program:State2" 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.

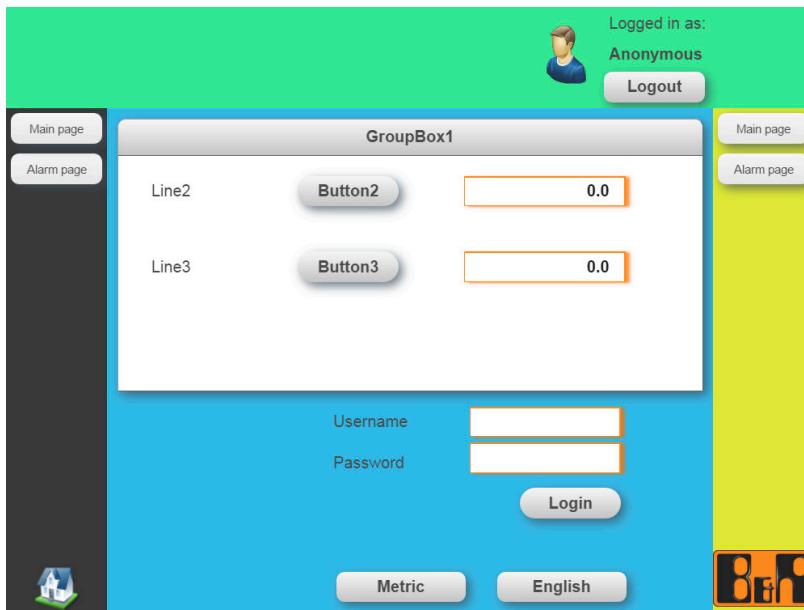


Figure 29: Hidden first line in a GroupBox

4.6 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.

Animations in the HMI application

Expected result

Changing the size if the browser window and restarting the HMI application scales the HMI application correspondingly. The piece of content is resized so that it is completely 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 8: 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 9: Additional variable type



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.

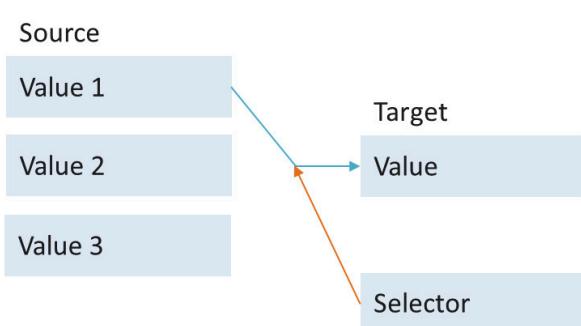


Figure 30: Binding to an element of a list

Tasks with different variable types

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.

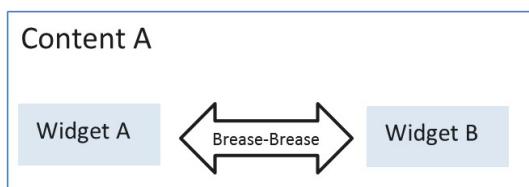


Figure 31: Widget information exchange within the same content

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.

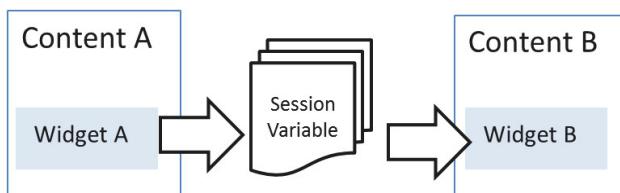


Figure 32: Widget information exchange via content borders

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 page "TestPage" and expand the automatic navigation
- 2) A BasicSlider widget and a RadialGauge widget are added in the ContentTestPage of the new page.
- 3) The value property of the RadialGauge widget is linked with the value property of the BasicSlider widget.



Steps 1 and 2 can now be carried out independently with the following steps.

Ways to create a new page:

- 1) Copy an existing page in the Logical View
Error messages in the output window show that the IDs of the page or the content have already been used.
- 2) Rename the page package "TestPage"
- 3) Open the .content file under the newly added page with "Open as text"
- 4) Change the ID to "ContentTestPage"
- 5) Open the .page file and change the page ID= "TestPage"
- 6) Change the attribute "displayName" to a text ID to be newly created (e.g. \$IAT/TestPage")
- 7) Enter in the <Assignment> of the AreaMain baseContentRefId="ContentTestPage"
- 8) Reference the new page ID in the HMI application (.vis) under the element <Pages>
- 9) Complete an existing text file (.tmx) with the text ID and the localizable texts
- 10) Add a new page to the automatic navigation (.nav)
Ensure that this new page is also reachable as a destination of e.g. the MainPage

When these steps have been performed, you can then navigate to the new page. Now a BasicSlider widget and a RadialGauge widget can be added to the graphic content editor.



If a piece of content is still open in a text editor (XML), it must first be closed before being opened with the visual editor.

5.1.2 Brease-Brease binding

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

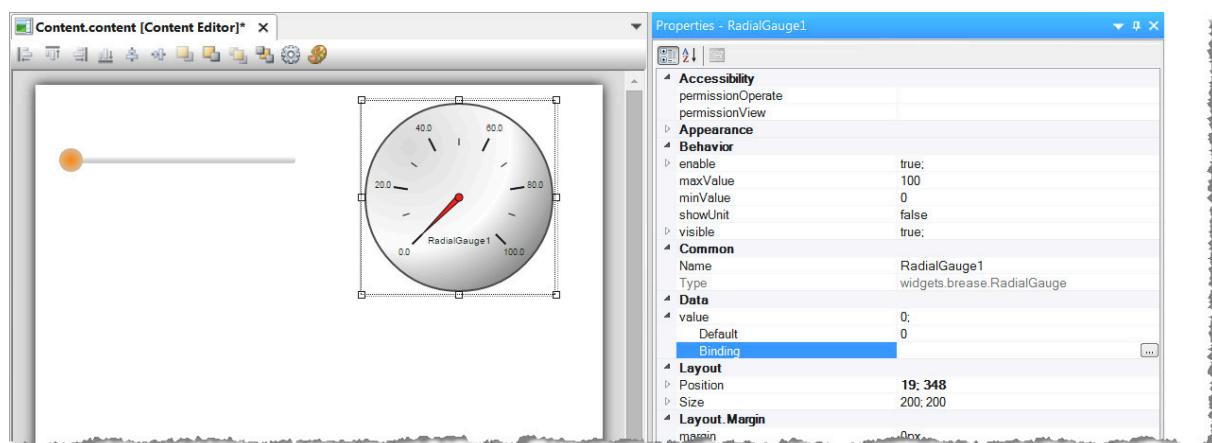


Figure 33: Setting up the binding on the value property of the RadialGauge widget

Tasks with different variable types

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

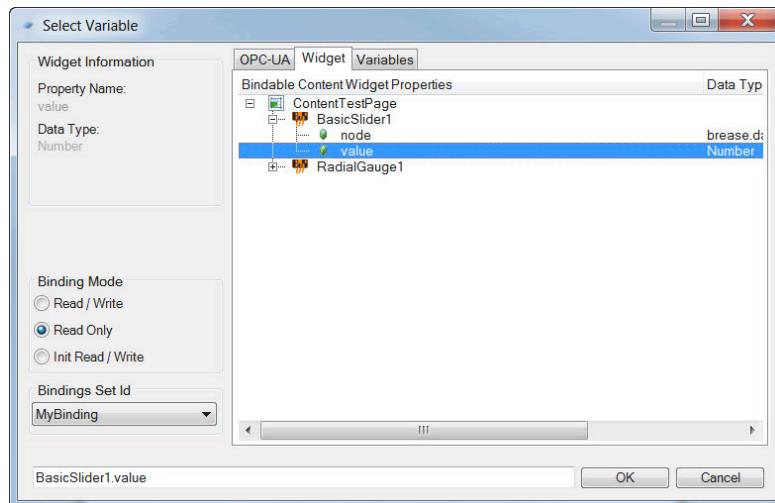


Figure 34: Value binding on a widget located in the same content

Expected result

The binding between the widgets is entered in the binding file.

```
<Binding mode="oneWay">
    <Source xsi:type="brease" contentRefId="ContentTestPage"
        widgetRefId="BasicSlider1" attribute="value" />
    <Target xsi:type="brease" contentRefId="ContentTestPage"
        widgetRefId="RadialGauge1" attribute="value" />
</Binding>
```

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

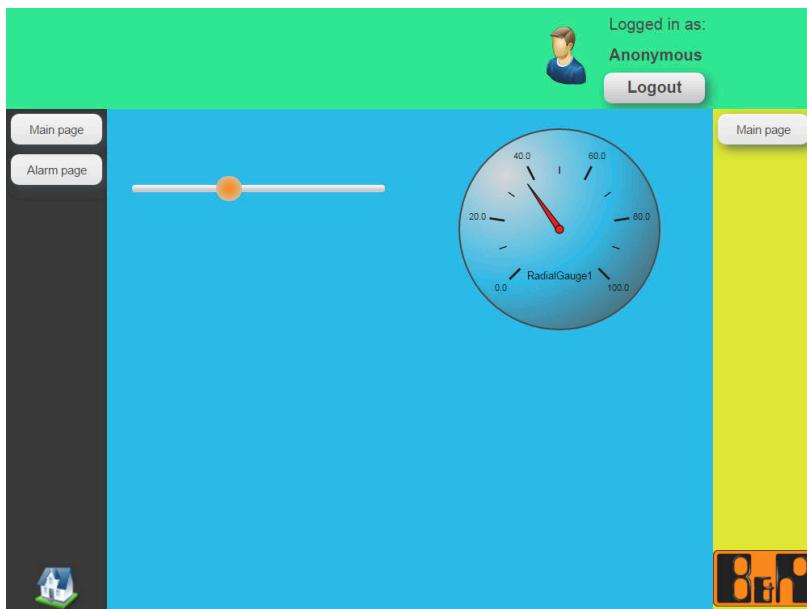


Figure 35: Changing the value of the slider moves the arrow pointer of the RadialGauge

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_INT"
- 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

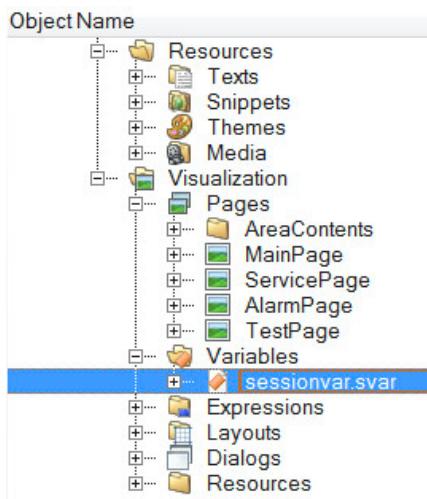
5.1.4 Session variables

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



mapp View / Engineering / Variables and data / Session variables

Tasks with different variable types



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

Figure 36: Session variable file in the Logical View

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_INT" 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_INT" 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".

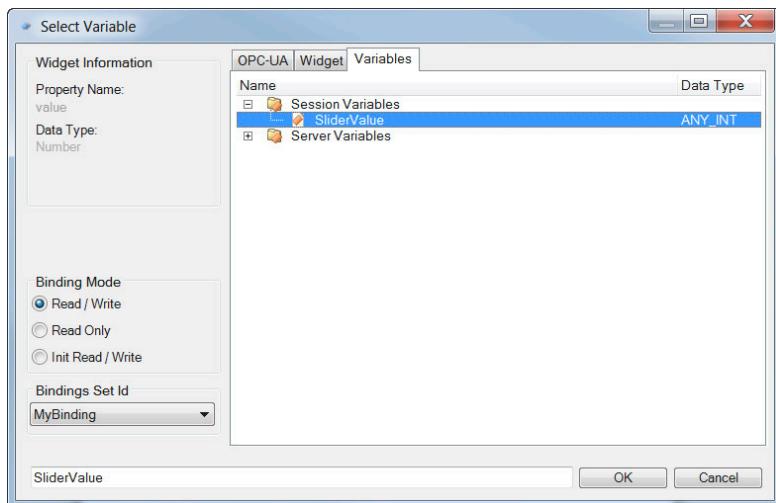


Figure 37: Selecting the session variable

To finish, a NumericOutput widget is added in the ContentTop and the same session variable connected to its value property. The bindings look like this after saving the content in the binding file.

```
<Binding mode="twoWay">
    <Source xsi:type="session" refId="SliderValue" attribute="value" />
    <Target xsi:type="brease" contentRefId="ContentTestPage"
        widgetRefId="BasicSlider2" attribute="value" />
</Binding>
<Binding mode="oneWay">
    <Source xsi:type="session" refId="SliderValue" attribute="value" />
    <Target xsi:type="brease" contentRefId="ContentTop"
        widgetRefId="NumericOutput1" attribute="value" />
</Binding>
```

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.

Tasks with different variable types

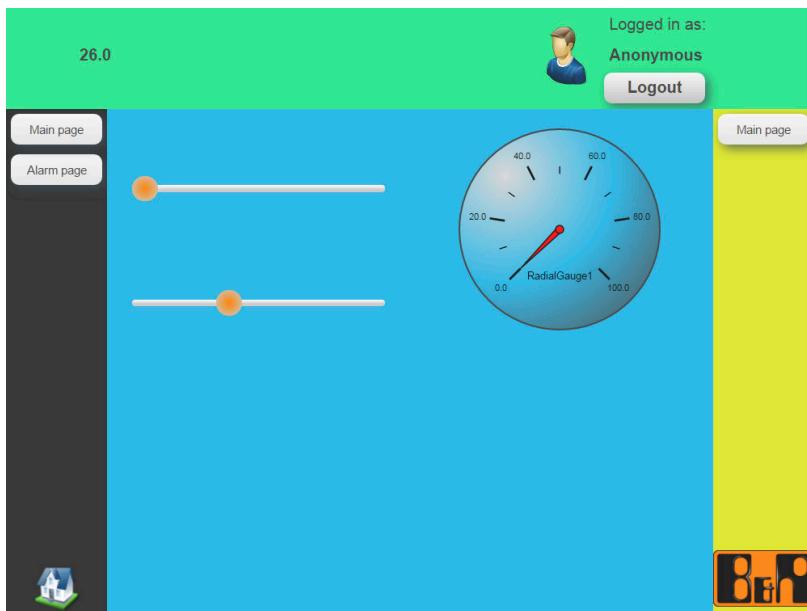


Figure 38: Binding spanning different pieces of content via a session variable

5.2 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.



[mapp View / Guidelines / FAQ / Event and action applications / Changing page with logout](#)

Expected result

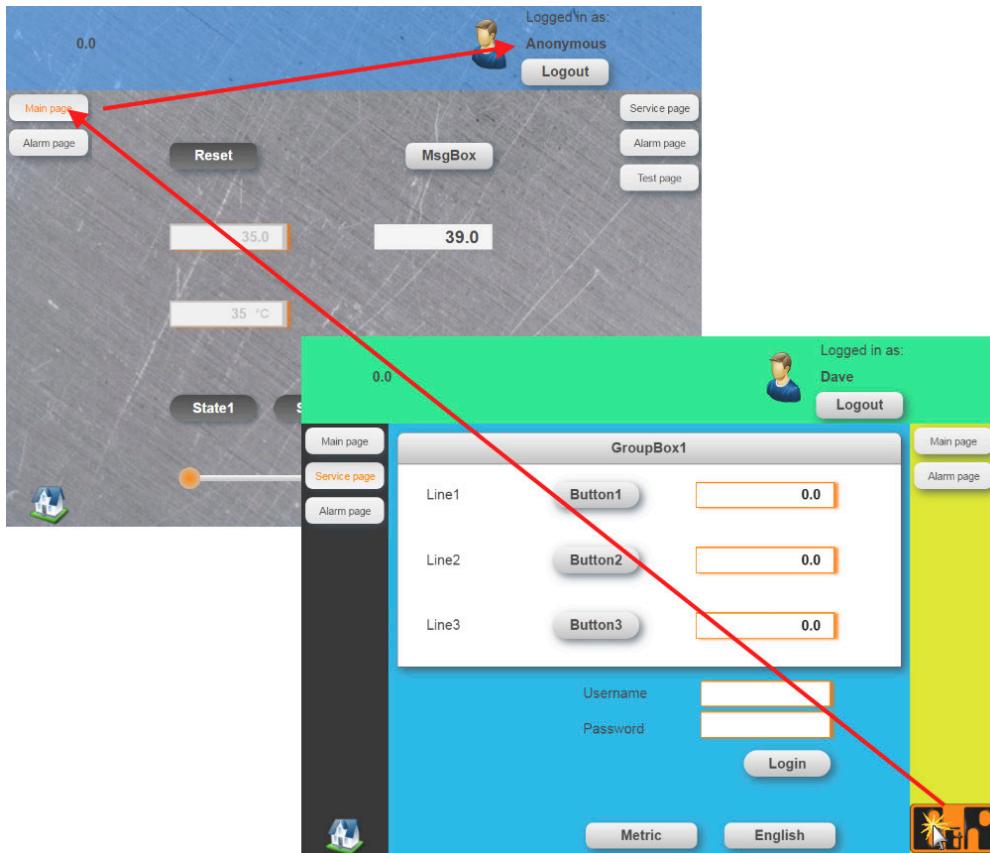


Figure 39: Logout with navigation to MainPage

5.3 Using system variables

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



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 connected to an OPC structure array.

The following steps are necessary for this:

Tasks with different variable types

- 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.

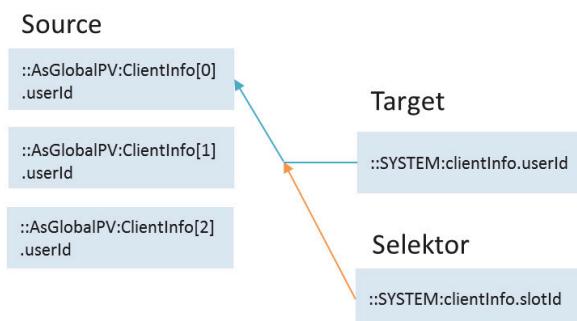


Figure 40: Schema of the list binding for the user ID

Expected result

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

Tasks with different variable types

| Name | Type | Scope | Force | Value |
|-------------------|----------------------|--------|-------|-------------|
| ClientInfo | ClientInfoType[0..2] | global | | |
| ClientInfo[0] | 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 | | | 0 |
| ClientInfo[1] | ClientInfoType | | | |
| userId | WSTRING[80] | | | "" |
| isValid | BOOL | | | FALSE |
| ipAddress | WSTRING[15] | | | "" |
| languageId | WSTRING[2] | | | "" |
| browserResolution | WSTRING[32] | | | "" |
| slotId | SINT | | | 0 |
| ClientInfo[2] | ClientInfoType | | | |
| userId | WSTRING[80] | | | "" |
| isValid | BOOL | | | FALSE |
| ipAddress | WSTRING[15] | | | "" |
| languageId | WSTRING[2] | | | "" |
| browserResolution | WSTRING[32] | | | "" |
| slotId | SINT | | | 0 |

Figure 41: Client information for a connected client in the Watch window

Exercises with the text system

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.



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).

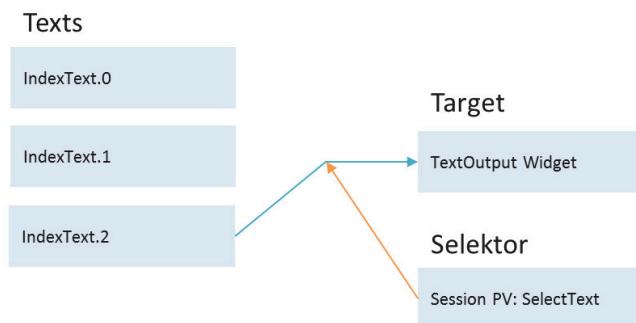


Figure 42: Schema for displaying indexed texts

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.

| Text ID | English (en) |
|-------------|------------------------|
| IndexText.0 | Fill level is too low |
| IndexText.1 | Fill level ok |
| IndexText.2 | Fill level is too high |
| TODO | |

Figure 43: Text ID with continuous number for indexed texts

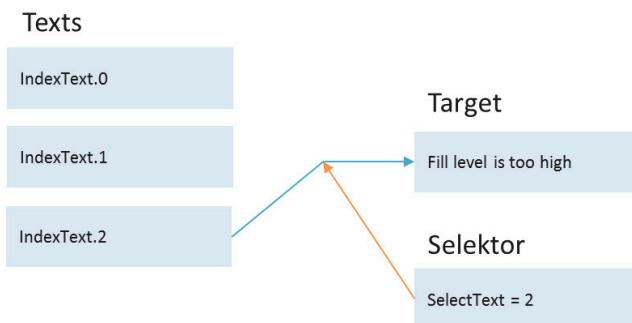
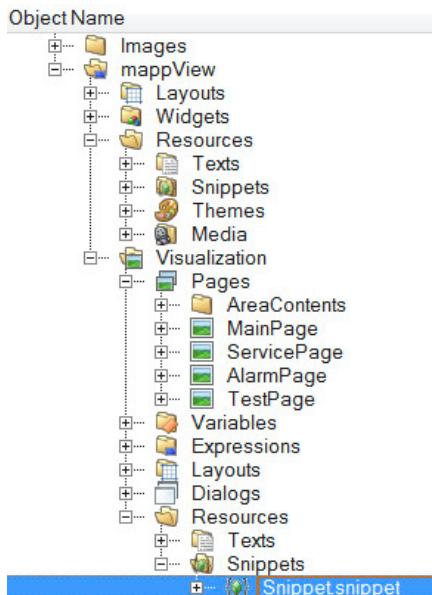


Figure 44: Displaying the text if SelectText has the value 2

Exercises with the text system

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 45: 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

Exercises with the text system

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

Expected result

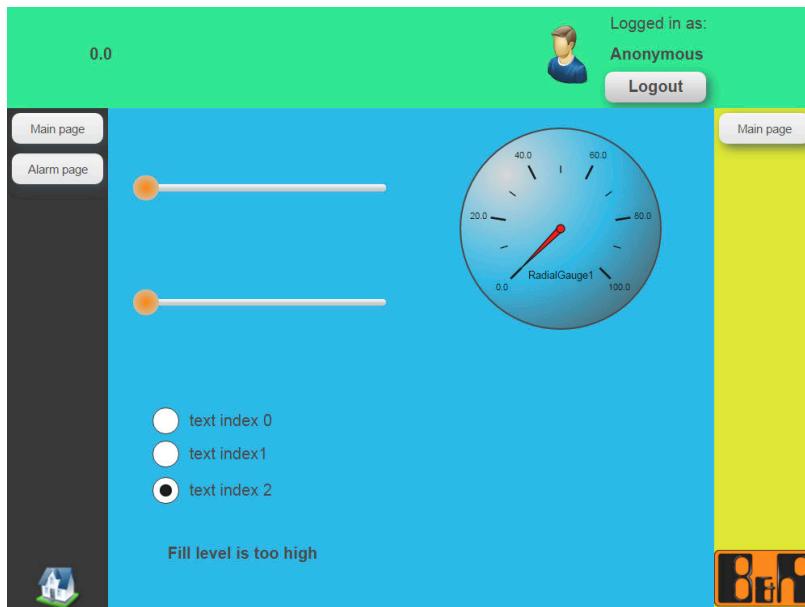


Figure 46: A change to the value leads to a change of the text from the text system.

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



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

7 Content engineering in XML

The content was previously configured in the visual editor. For anyone who prefers editing in XML, the following exercises show what has to be considered when configuring a piece of content in XML.



It's not possible to have a piece of content open simultaneously in the visual editor and the XML editor. Furthermore, what must be taken into account is that formatting carried out in XML (indentation, etc.) is lost after editing the content in the visual editor.

The XML editor is opened in the shortcut menu of the content file "Open / Open as text". In the following exercise, the content "ContentAlarmPage" is opened with the text editor. For better readability, the entries of the XML schemas are not shown:

```
<?xml version="1.0" encoding="utf-8"?>
<Content id="ContentAlarmPage" height="500" width="600" >
    <Widgets>

    </Widgets>
</Content>
```

| Property name | Description |
|---------------|--|
| Content ID | Unique ID over all content files of the Logical View |
| Height | Height of the contents in pixels |
| Width | Width of the content in pixels |

Table 10: Overview of the basic properties of a content file

If the content is opened, all widgets that can be added to the <Widgets> element are shown in the Widget Catalog.

7.1 Adding widgets

A widget is added with drag and drop within a <Widgets> element. When dropped from the Widget Catalog, the minimum required attributes of the widget are shown.

```
<Widget xsi:type="widgets.brease.Label" id="Label1"
height="30" width="80" left="" top="" text="Label1" zIndex="0" />
```

As seen in the Label widget example, the widget is given a unique ID. The size is defined by the widget with "width" and "height"; the position must be added by the user with "top" and "left".

Furthermore, every widget within a <Widgets> element must have a unique zIndex – beginning with "0".



When using copy and paste, ensure that every widget has a unique ID and a unique zIndex. These two attributes are checked to determine if they are unique when saving the file and an error message is shown in the output window if there is double usage.

Content engineering in XML

For every widget, further attributes that are permitted for the respective widget can be added or changed. The file is validated when it is saved (capitalization, etc.). Possible attributes can be found in the widget documentation.



mapp View / Widgets

7.2 Adding Container widgets

Container widgets (e.g. `GroupBox`) allow child widgets to be added. In turn, a Container widget is labeled using a `<Widgets>` element. All the rules for a simple widget are also valid for the widget itself, however, the `zIndex` for widgets within the Container widget can again begin with "0".

```
<Widget xsi:type="widgets.brease.GroupBox" id="GroupBox1"
    height="150" width="200" left="" top="" zIndex="1" >
    <Widgets>
        ...
    </Widgets>
</Widget>
```

Additional widgets can now be added below the `<Widgets>` element. As practice, XML editor widgets can be added and configured. After specifying all of the necessary attributes (e.g. `top/left`), the content can be saved and closed.

Expected result



8 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.

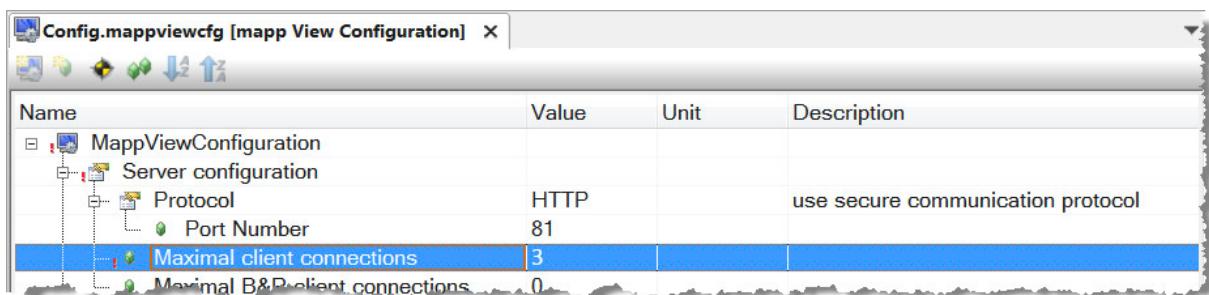


Figure 47: Allowing 3 client connections



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.

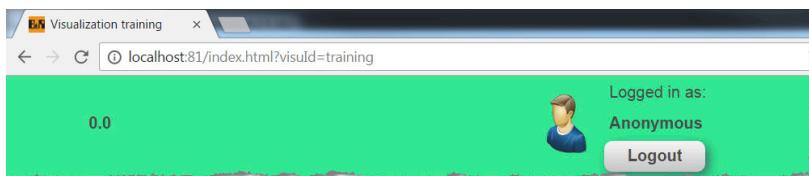


Figure 48: Displaying the HMI application in the browser

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.

Multi-client – Multi-user

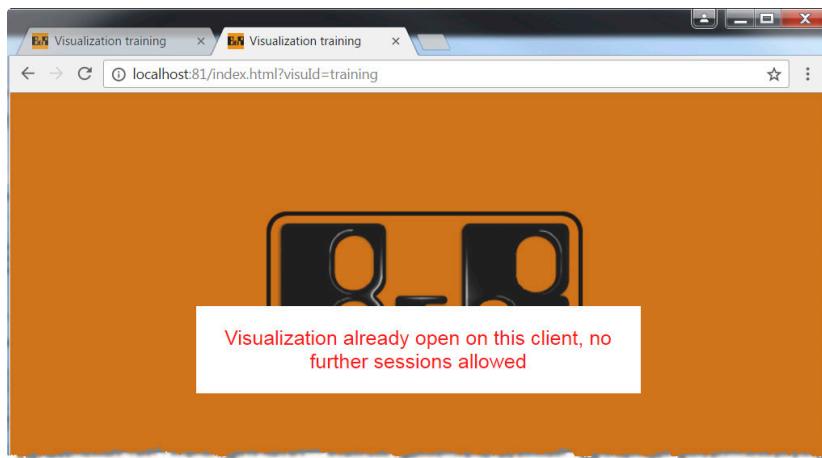


Figure 49: An error message if the browser connects to the same session

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. Alternatively, two different browsers from different manufacturers can be used on the same PC (e.g. Chrome and Explorer) or a different end device is used for access.

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.

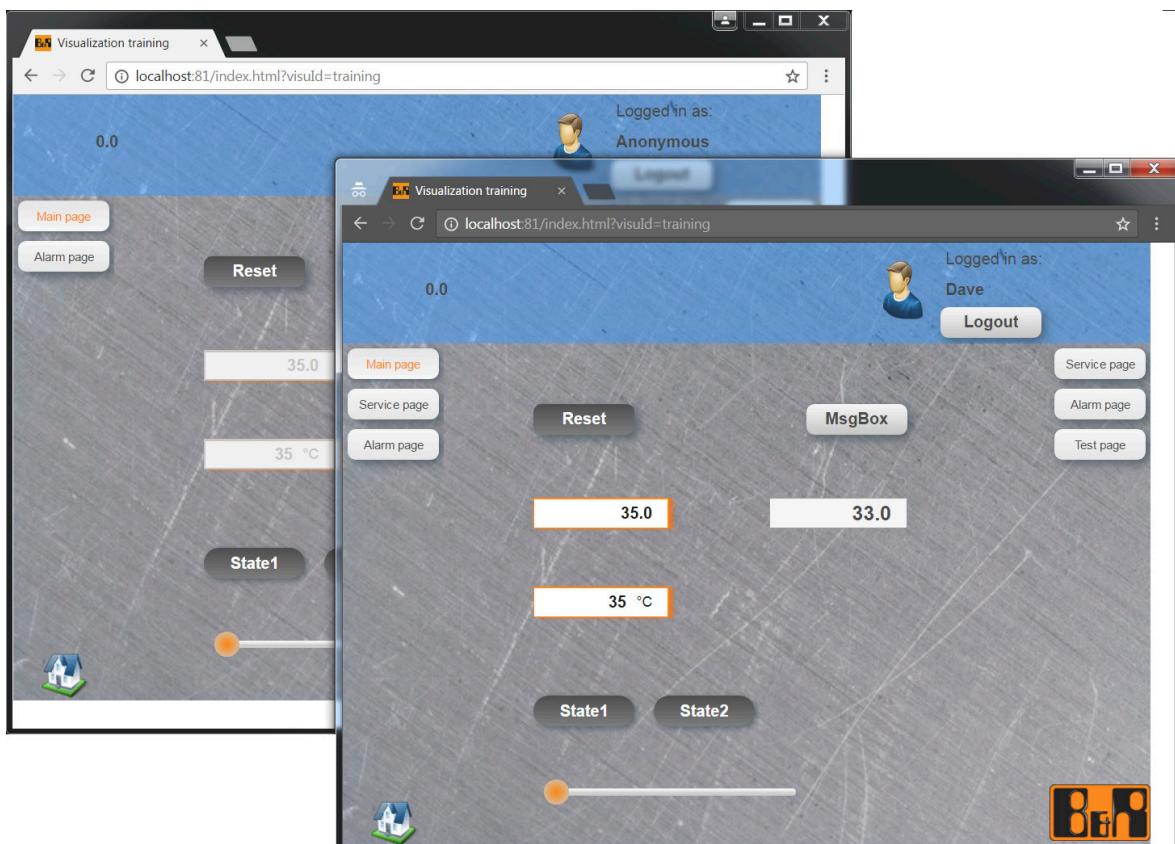


Figure 50: Displaying the HMI application with two client instances

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

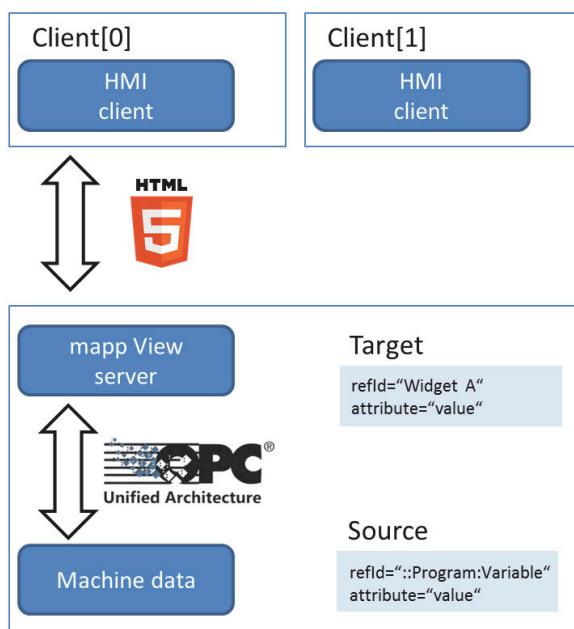
Multi-client – Multi-user

The screenshot shows a 'Watch' window with the title 'Program::Program.pvm [Watch]'. It displays a table with columns: Name, Type, Scope, and Value. The table contains data for three clients, indexed from 0 to 2. Each client entry includes fields for user ID, validity, IP address, language, browser resolution, and slot ID.

| Name | Type | Scope | Value |
|-------------------|----------------------|-------|-------------|
| ClientInfo[0] | ClientInfoType[0..2] | globz | |
| 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 |

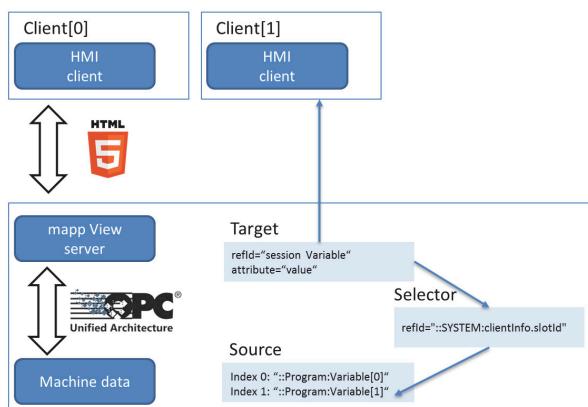
Figure 51: Client information for a connected client in the Watch window

8.1 Client-specific OPC UA binding



When binding an OPC UA variable to a widget or when using it as an event trigger in the event binding, this binding / event binding is valid for all clients on which an HMI application is delivered.

Figure 52: Diagram showing a value binding



The following exercise shows how a client-specific binding is correctly implemented on OPC UA variables for multi-client HMI applications using a list binding.

Figure 53: Diagram showing a list binding
TM671 - Creating efficient mapp View HMI applications

8.1.1 Client-dependent binding to a certain OPC UA variable

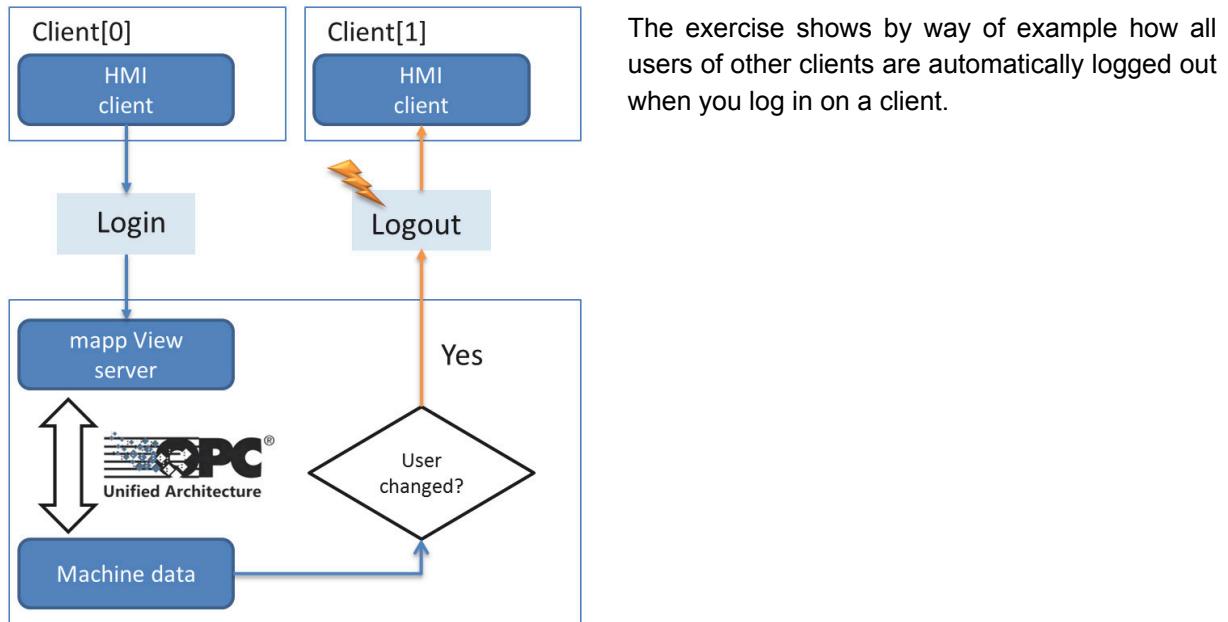


Figure 54: Diagram showing logout when a user changes

Exercise: Automatic logout when a user changes on another client

The following description shows how a logout command is sent to the other clients when the variable "clientSlot[0..2].userId" is changed.

The following steps are necessary for this:

- 1) Add the standard library "AsBrStr"
- 2) Create a program for recognizing a user change
- 3) Declare variables
- 4) Activate the variable "logoutCommand" in the OPC UA default view
- 5) Create a new session variable "LogoutCommand" with data type "BOOL"
- 6) Create a list binding in which the command is connected to the session variable as a function of the slotId (selector)
- 7) Create an event binding in which a logout for a ValueChanged of the session variable is recognized.



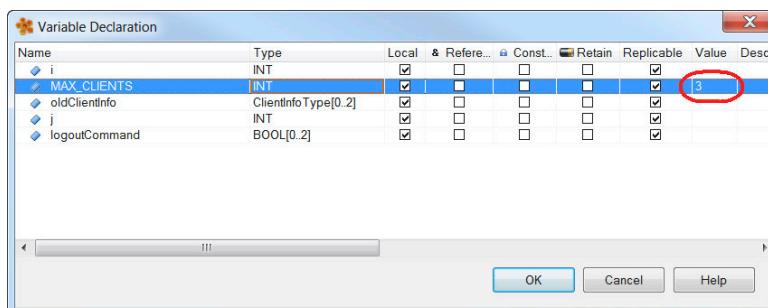
Step 1 can now be carried out independently.

8.1.2 Program for recognizing a user change on the client

The following program code for recognizing the login of a random client is used in the program "Program" of the Logical View.

Multi-client – Multi-user

```
FOR i := 0 TO MAX_CLIENTS - 1 DO
    IF brsstrcmp(ADR(oldClientInfo[i].userId), ADR(ClientInfo[i].userId)) <> 0 THEN
        FOR j := 0 TO MAX_CLIENTS - 1 DO
            IF i <> j THEN
                logoutCommand[i] := FALSE;
                logoutCommand[j] := TRUE;
            END_IF;
        END_FOR;
    END_IF;
END_FOR;
oldClientInfo := ClientInfo;
```



The variables have to be declared in the variable declaration for the program according to the following image:

Figure 55: Variable declaration for the logout logic

Subsequently, the IEC variable "logoutCommand[0..2]" in the OPC UA default view is enabled and the array elements are also shown.

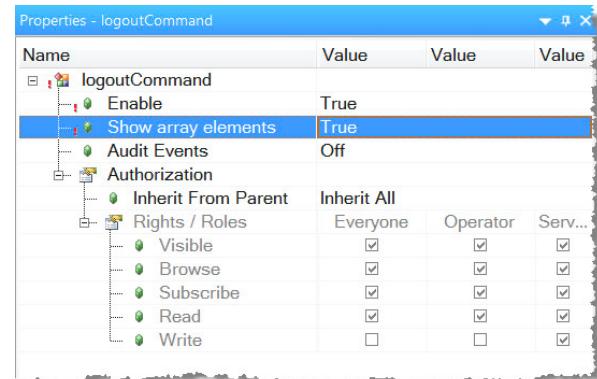


Figure 56: Declaration of OPC UA variable "logoutCommand"

8.1.3 Binding the session variable for receiving the logout command

A new session variable with name "LogoutCommand" and data type "BOOL" is created in the existing session variables file (.svar) for the Logical View. This is then used to save the logout command, which in turn is used in the event binding as an event for the logout action.

```
<Variable name="LogoutCommand" xsi:type="BOOL" value="false" />
```

This session variable is now connected in an existing binding file using a list binding. The system variable "::SYSTEM.clientInfo.slotId" is used as a selector, which receives a unique value (0-2) when connecting a client to the mapp View server.

```

<Binding mode="oneWay">
    <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="::Program:logoutCommand[0]" />
            <bt:Element index="1" refId="::Program:logoutCommand[1]" />
            <bt:Element index="2" refId="::Program:logoutCommand[2]" />
        </be>List>
    </Source>
    <Target xsi:type="session" refId="LogoutCommand" attribute="value" />
</Binding>

```

In the last step, a new <EventBinding> element is created in the existing event binding file. A value change to session variable "LogoutCommand" is defined as an event (source), and the method "Logout" for a clientSystem action is called in the <EventHandler> if condition="true".

```

<EventBinding>
    <Source xsi:type="session.Event" refId="LogoutCommand" event="ValueChanged" />
    <EventHandler condition="newValue">
        <Action>
            <Target xsi:type="clientSystem.Action">
                <Method xsi:type="clientSystem.Action.Logout" />
            </Target>
        </Action>
    </EventHandler>
</EventBinding>

```

Expected result

A logout is automatically performed on the other clients if there is a login on one of the connected clients.

Summary

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.

Seminars and training modules

At the Automation Academy, you will 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.



Automation Studio seminars and training modules

| Programming and configuration | Diagnostics and service |
|---|---|
| <p>SEM210 – Basics SEM246 – IEC 61131-3 programming language ST* SEM250 – Memory management and data storage</p> <p>SEM410 – Integrated motion control** SEM441 – Motion control: Electronic gears and cams** SEM480 – Hydraulics** SEM1110 – Axis groups and path-controlled movements**</p> <p>SEM510 – Integrated safety technology* SEM540 – Safe motion control***</p> <p>SEM610 – Integrated visualization*</p> | <p>SEM910 – Workshop: Control and I/O system design SEM920 – Diagnostics and service for end users SEM920 – Diagnostics and service with Automation Studio SEM950 – POWERLINK configuration and diagnostics*</p> <p>If you are unable to find a seminar on our website that suits your needs, we also offer customized seminars. Simply contact your sales representative to make the necessary arrangements: SEM099 – Individual training day</p> <p>Please visit our website for more information****.****: www.br-automation.com/academy</p> |

| Overview of training modules | |
|--|---|
| <p>TM210 – Working with Automation Studio TM213 – Automation Runtime TM223 – Automation Studio Diagnostics TM230 – Structured Software Development TM240 – Ladder Diagram (LD) TM241 – Function Block Diagram (FBD) TM242 – Sequential Function Chart (SFC) TM246 – Structured Text (ST) TM250 – Memory Management and Data Storage</p> <p>TM400 – Introduction to Motion Control TM410 – Working with Integrated Motion Control TM440 – Motion Control: Basic Functions TM441 – Motion control: Electronic gears and cams TM1110 – Integrated Motion Control (Axis Groups) TM1111 – Integrated Motion Control (Path Controlled Movements) TM450 – Motion Control Concept and Configuration TM460 – Initial Commissioning of Motors</p> <p>TM500 – Introduction to Integrated Safety TM510 – Working with SafeDESIGNER TM540 – Integrated Safe Motion Control</p> | <p>TM600 – Introduction to Visualization TM610 – Working with Integrated Visualization TM611 – Working with mapp View TM630 – Visualization Programming Guide TM640 – Alarm System, Trends and Diagnostics TM670 – Advanced Visual Components</p> <p>TN910 – Controller and I/O system design TM920 – Diagnostics and service TM923 – Diagnostics and Service with Automation Studio TM950 – POWERLINK Configuration and Diagnostics</p> <p>TM280 – Condition Monitoring for Vibration Measurement TM480 – The Basics of Hydraulics TM481 – Valve-based Hydraulic Drives TM482 – Hydraulic Servo Pump Drives TM490 – Printing Machine Technology</p> <p>In addition to the printed version, our training modules are also available on our website for download as electronic documents (login required):</p> |

Process control seminars and training modules

| Process control standard seminars | Process control training modules |
|--|--|
| <p>SEM841 – Process control training: Basic 1 SEM842 – Process control training: Basic 2 SEM890 – Advanced Process Control Solutions</p> | <p>TM800 – APROL System Concept TM810 – APROL Setup, Configuration and Recovery TM811 – APROL Runtime System TM812 – APROL Operator Management TM813 – APROL web portal TM820 – APROL solutions TM830 – APROL Project Engineering TM835 – APROL ST-SFC Configuration TM840 – APROL Parameter Management and Recipes TM850 – APROL Controller Configuration and INA TM860 – APROL Library Engineering TM865 – APROL Library Guide Book TM870 – APROL Python Programming TM880 – APROL reporting TM890 – The Basics of LINUX</p> |

* SEM210 - Basics is a prerequisite for this seminar.

** SEM410 - Integrated motion control is a prerequisite for this seminar.

*** SEM410 - Integrated motion control and SEM510 - Integrated safety technology are prerequisites for this seminar.

****Our seminars are listed in the Academy\Seminar area of the website.

*****Seminar titles may vary by country. Not all seminars are available in every country.

V1.0.0.3 ©2017/03/23 by B&R. All rights reserved.
All registered trademarks are the property of their respective owners.
We reserve the right to make technical changes.



TM671TRE.425-ENG