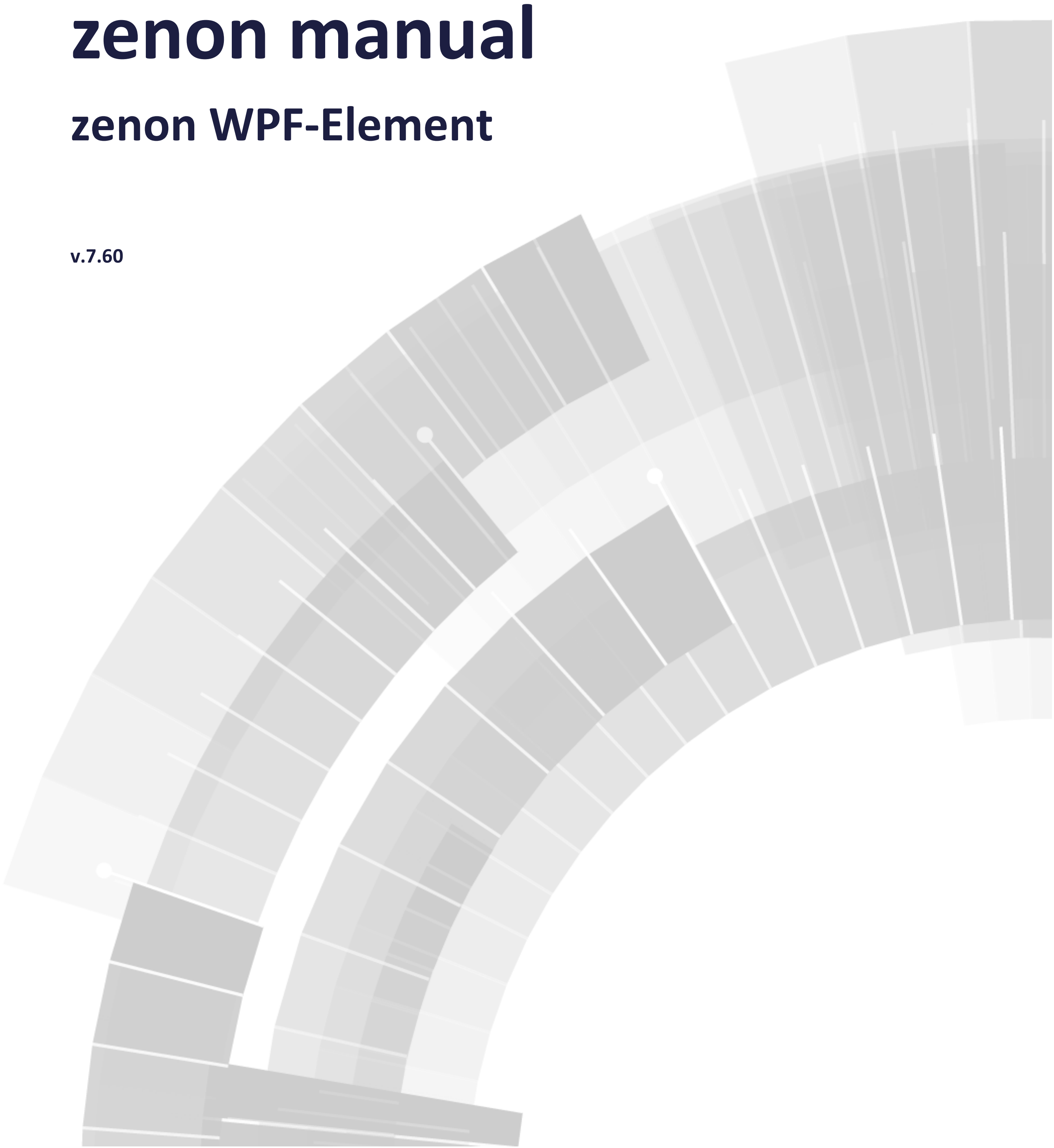
zenon manual

zenon WPF-Element

v.7.60



Contents

1. Welcome to COPA-DATA help

ZENON VIDEO-TUTORIALS

You can find practical examples for project configuration with zenon in our YouTube channel (https://www.copadata.com/tutorial_menu). The tutorials are grouped according to topics and give an initial insight into working with different zenon modules. All tutorials are available in English.

GENERAL HELP

If you cannot find any information you require in this help chapter or can think of anything that you would like added, please send an email to documentation@copadata.com (mailto:documentation@copadata.com).

PROJECT SUPPORT

You can receive support for any real project you may have from our Support Team, who you can contact via email at support@copadata.com (mailto:support@copadata.com).

LICENSES AND MODULES

If you find that you need other modules or licenses, our staff will be happy to help you. Email sales@copadata.com (mailto:sales@copadata.com).

2. WPF element

With the **WPF** dynamic element, valid WPF/XAML files in zenon can be integrated and displayed.

Note: In the zenon Editor, the standard tooltip for the WPF element is not displayed if a **.wpf** file is linked. Furthermore, in zenon Runtime, the zenon tooltip for WPF elements is not supported.

Information

All brand and product names in this documentation are trademarks or registered trademarks of the respective title holder.

3. Basics

XAML

XAML stands for **Extensible Application Markup Language**. The XML-based descriptive text developed by Microsoft defines graphic elements, animations, transformations, displays of color gradients etc. in Silverlight and WPF user interfaces. The use of XAML makes it possible to strictly separate design and programming. The designer prepares, for example, the graphical user interface and creates basic animations that are then used by the developers/project planners who create the application logic.

WPF

WPF stands for **Windows Presentation Foundation** and describes a graphics framework that is part of the Windows .NET framework:

- WPF provides a comprehensive model for the programmer.
- XAML describes, based on XML, the interface hierarchy as a markup language. Depending on the construction of the XAML file, there is the possibility to link properties, events and transformations of WPF elements with variables and functions of CD_PRODUCTNAME<.
- The framework unites the different areas of presentation such as user interface, drawing, graphics, audio, video, documents and typography.

For execution in zenon, Microsoft .NET framework version 3.5 or higher is required.

Information

Transparency

In order for WPF controls in which a transparent background has been defined to also be displayed as transparent, the following must be the case on the computer for both Editor and Runtime:

- The operating system must be at least Windows 8.1
- The .NET framework version 4.6 or higher must be installed

WPFs are not shown as transparent in Windows 7 or 8. Instead, the transparent areas are filled with the background color set on the zenon screen.

3.1 WPF in process visualization

XAML makes different design possibilities possible for zenon. Display elements and dynamic elements can be adapted graphically regardless of the project planning. For example, laborious illustrations are first created by designers and then imported into zenon as an XAML file and linked to the desired logic. There are many possibilities for using this, for example:

DYNAMIC ELEMENTS IN ANALOG-LOOK

Graphics no longer need to be drawn in zenon, but can be imported directly as an XAML file. This makes it possible to use complex, elaborately illustrated elements in process visualization. Reflections, shading, 3D effects etc. are supported as graphics. The elements that are adapted to the respective industry environment make intuitive operation possible, along the lines of the operating elements of the machine.

INTRICATE ILLUSTRATIONS FOR INTUITIVE OPERATION

The integration of XAML-based display elements improves the graphics of projects and makes it very easy to display processes clearly. Elements optimized for usability make operation easier. A clear display of data makes it easier to receive complex content. The flexible options for adapting individual elements makes it easier to use for the operator. It is therefore possible for the project planners to determine display values, scales and units on their own.

CLEAR PRESENTATION OF DATA AND SUMMARIES

Grouped display elements make it possible to clearly display the most important process data, so that the equipment operator is always informed of the current process workflow. Graphical evaluations, display values and sliders can be grouped into an element and make quick and uncomplicated control possible.

INDUSTRY-SPECIFIC DISPLAYS

Elements such as thermometers, scales or bar graphs are part of the basic elements of process visualization. It is possible, using XAML, to adapt these to the respective industry. Thus equipment operators can find the established and usual elements that they already know from the machines in process visualization at the terminal.

ADAPTATION TO CORPORATE DESIGN

Illustrations can be adapted to the respective style requirements of the company, in order to achieve a consistent appearance through to the individual process screen. For example, the standard operation elements from zenon can be used, which can then be adapted to color worlds, house fonts and illustration styles of the corporate design.

3.2 Referenced assemblies

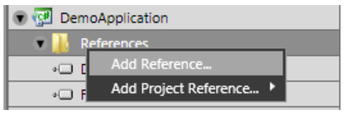
It is not just standard objects (rectangles, graphics, etc.) or effects (color gradients, animations, etc.) that can be displayed using the **WPF elements**, but also customized user controls (with logic in the code behind), which are referenced as assemblies.

For example, a user control that looks like a tacho and provides special properties and optical effects can be created, such as a "Value" property, which causes the pointer of the tacho to move and/or the corresponding value to be displayed in a label.

The workflow for this:

- The appearance of a user controls is labeled with standard objects, which are offered by WPF.
- The properties and interactions are programmed.
- The whole package is compiled and present in the form of a .NET assembly.

This assembly can also be used for WPF projects. To do this, it must be referenced (linked) in the WPF editor (for example: Microsoft Expression Blend). To do this, select the assembly in the zenon file selection dialog:



From this point in time, the WPF user controls of the assembly in the tool box can be selected under **Custom user controls** and used in the WPF project.

See also, in relation to this, the following chapter: Guidelines for developers (on page 24).

USED REFERENCED ASSEMBLIES IN ZENON

To use an assembly in zenon, this must be provided as a file.
Collective files in **.cdwpf** format administer these independently; no further configuration is necessary. Assemblies must be added to the `Files` folder for .xaml files:

- Click on `Files` on the project tree
- Select `Other`
- Select **Add file...** in the context menu
- The configuration dialog opens
- Insert the desired assembly

When displaying a WPF file in the **WPF element** (Editor and Runtime), the assemblies from this folder are loaded. It is thus also ensured that that when the Runtime files are transferred using **Remote Transport**, all referenced assemblies are present on the target computer.

A collective file (**.cdwpf**) can exist alongside an XAML file with the same name. All assemblies (*.dll) from all collective files and the `Other` folder are copied to the work folder. Only the highest file version is used if there are several assemblies with the same name.

Hint

DLLs that belong to a WPF element (referenced by the linked XAML file) can also be replaced in the Editor during ongoing operation.
To replace a DLL:

- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In Explorer, replace the DLL in the `\wpfache` folder of the Editor files.
  You can find this folder in the SQL directory under
  `...\PROJECT-GUID\FILES\zenon\custom\wpfcache`

As an alternative to replacement using Explorer, you can also replace the file in the zenon Editor directly; to do this:

- In the Visual Studio project settings, increase the file version of the DLL.
- Create the new DLL.
- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In the zenon Editor, delete the DLL from the `\Files\Other` folder and add the file with the higher version number.

MULTI-PROJECT ADMINISTRATION

With multi-project administration, the same assembly must be used in all projects. If an assembly is replaced by another version in a project, it must also be replaced in all other projects that are loaded in the Editor or in Runtime.

3.3 Workflows

The WPF/XAML technology makes new workflows in process visualization possible. The separation of design and functionality ensures a clear distinction of roles between the project engineer and designers; design tasks can be easily fulfilled by using pre-existing designs, which no longer need to be modified by the project engineer.

The following people are involved in the workflow to create WPF elements in zenon:

- Designer
  - illustrates elements
  - takes care of the graphics for MS Expression Design
- MS Expression Blend operator
  - Animates elements

  - Creates variables for the animation of WPF elements in zenon, which project engineer can access
- Project engineer
  - Integrates elements into zenon:
  - stores logic and functionality

We make a distinction:

- Workflow with Microsoft Expression Blend (on page 11)
- Workflow with Adobe Illustrator (on page 11)

3.3.1 Workflow with Microsoft Expression Blend

When using Microsoft Expression Blend, a WPF element is created in four stages:

1. Illustration of elements in **MS Expression Blend** (on page 12)
2. Open element in **MS Expression Design** and export as WPF
3. Animation in **MS Expression Blend** (on page 12)
4. Integration into zenon (on page 102)

You can find an example for creating a WPF elements with Microsoft Expression Blend in the Create button as XAML file with Microsoft Expression Blend (on page 12) chapter.

3.3.2 Workflow with Adobe Illustrator

Based on traditional design processes with **Adobe Illustrator** the following workflow is available:

1. Illustration of elements in **Adobe Illustrator** (on page 17)
2. Import of **.ai** files and preparation in **MS Expression Design** (on page 19)
3. WPF export from **MS Expression Design** (on page 19)
4. Animation in **MS Expression Blend** (on page 20)
5. Integration into zenon (on page 97)

You can find an example for creation in the Workflow with Adobe Illustrator (on page 16) chapter.

4. Guidelines for designers

This section informs you how to correctly create WPF files in Microsoft Expression Blend and Adobe Illustrator. The tutorials on Creating a button element (on page 12) and a bar graph element (on page 16) show you how fully functional WPF files for zenon can be created from pre-existing graphics in a few steps.

The following tools were used for this:

- Adobe Illustrator CS3 (AI)
- Microsoft Expression Design 4 (ED)
- Microsoft Expression Blend 4 (EB)
- zenon

> **Information**
>
> *If referenced objects (assemblies) are used in WPF, note the instructions in the Referenced objects (on page 8) chapter.*

4.1 Workflow with Microsoft Expression Blend

With Microsoft Expression Blend, a WPF element:

- is illustrated
- is converted into WPF format using **MS Expression Design**
- animated

The following example shows the illustration and conversion of a button element into an XAML file.

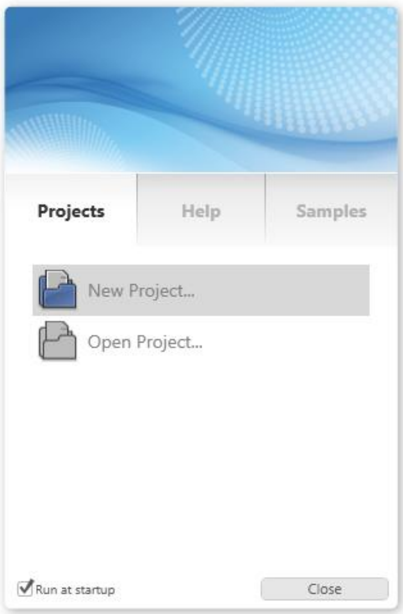
Note: A test version of "Microsoft Expression Blend" can be downloaded from the Microsoft website.

4.1.1 Create button as an XAML file with Microsoft Expression Blend

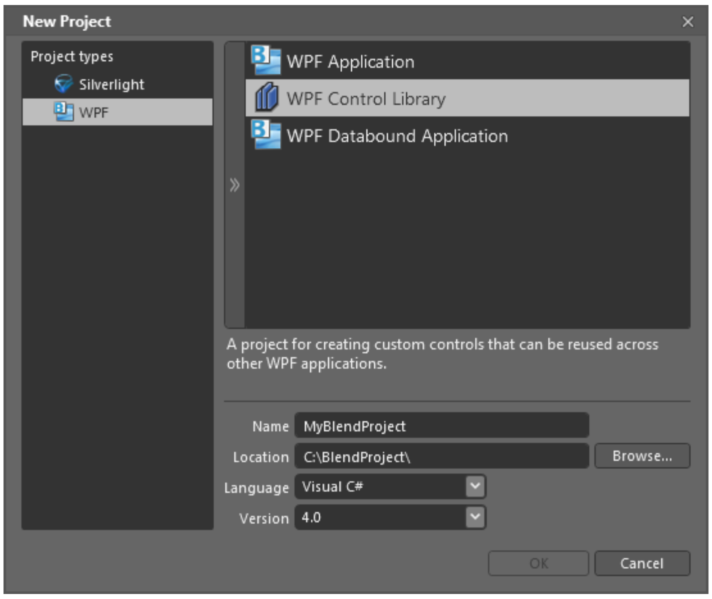
CREATE BUTTON

1. Start Expression Blend

2. select the **New Project** option



3. Select `WPF` as project type
4. give it a path and name of your choice (MyBlendProject, for example)



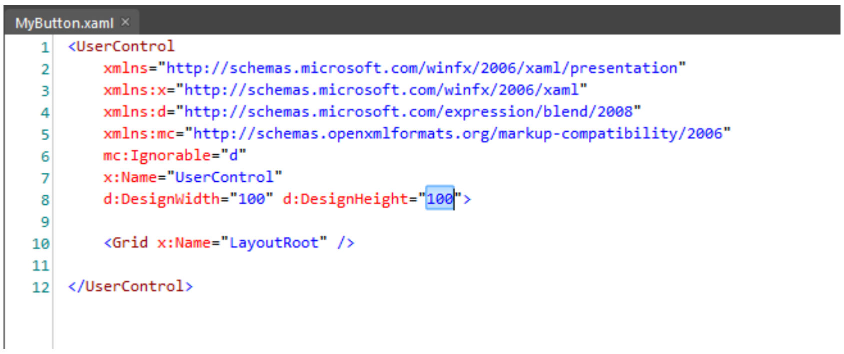
A project for creating custom controls that can be reused across other WPF applications.

Name: MyBlendProject
Location: C:\BlendProject\
Language: Visual C#
Version: 4.0

The **Language** and **Version** settings can be ignored, because no functionality is to be programmed.

5. After the dialog has been confirmed with **OK**, Microsoft Blend creates a new project with the chosen settings. Expression Blend adds an empty XAML file which already contains a class reference.
6. Delete the CS file that belongs to the XAML file using the context menu.

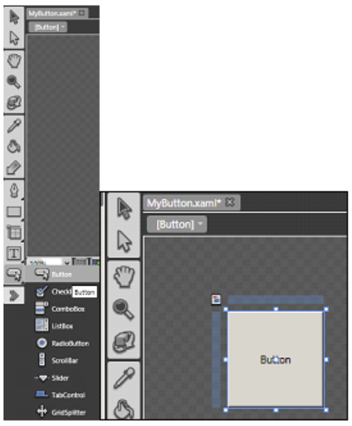
7. Rename the XAML file **MainControl.xaml** to **MyButton.xaml**.
8. The development size of the file is set at 640 x 480 pixels as standard and must still be changed:
   a) switch to **XAML** view
   b) correct the size to 100 x 100 pixels
   c) Delete the class reference x:Class="MyBlendProject.MyButton"



MyButton.xaml

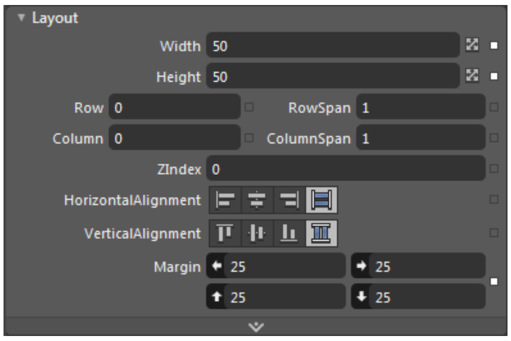
```
<UserControl
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    x:Name="UserControl"
    d:DesignWidth="100" d:DesignHeight="100">

    <Grid x:Name="LayoutRoot" />

</UserControl>
```

9. switch to **Design** view

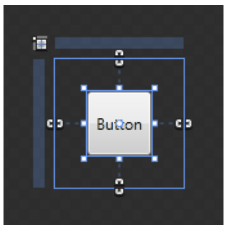


10. add a button via the toolbar
11. define the properties
    - Width: 50
    - Height: 50

- Margins: 25


Layout
Width 50
Height 50
Row 0 RowSpan 1
Column 0 ColumnSpan 1
ZIndex 0
HorizontalAlignment
VerticalAlignment
Margin 25 25 25 25

The button is therefore at the center of the control.



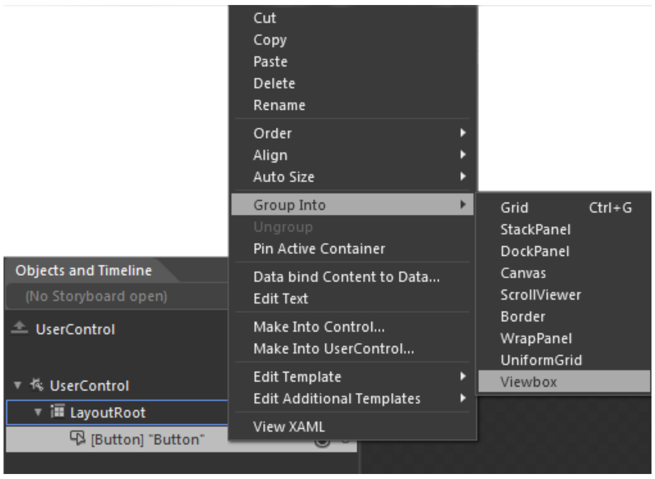
12. Save the changes and open the file in Internet Explorer to check it. You will see that the button is displayed in a size of 50 x 50 pixels.

MAKE BUTTON SCALABLE

If you integrate this status into zenon, the button will always have the exact size of 50 x 50 pixels. Because the button can be implemented as a scalable button, switch to Expression Blend again:

1. Select the button in the tree view.
2. select the *Group Into->Viewbox* button in the context menu
3. the button is inserted into a **Viewbox**
4. Define the properties of the viewbox
   - Width: Auto
   - Height: Auto

5. save the file



6. If you now open the file in Internet Explorer, the button is automatically scaled when the IE window size is changed. This file will now also automatically adapt to changes in the size of the **WPF element** in zenon.

CHANGE NAME

Before you can integrate the file into zenon, you must give the **WPF element** a name. The **WPF elements** are not named in Expression Blend as standard, and are labeled with square brackets and their type. zenon content is assigned to WPF content via the name of the **WPF elements**:

- in tree view, change the name
  - of the button on **MyButton**
  - of the ViewBox to **MyViewBox**

This button can now be integrated in zenon (on page 102) as an XAML file.

4.2 Workflow with Adobe Illustrator

When **Adobe Illustrator** is used, a WPF element:

- is illustrated in **Adobe Illustrator**
- is converted into a WPF in **MS Expression Design**
- is animated in **MS Expression Blend**

The following example shows the illustration and conversion of a bar graph element into an XAML file.

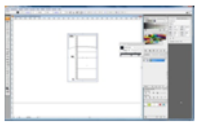
4.2.1 Bar graph illustration

A bar graph is created in Adobe Illustrator.

1. AI: Starting element for bar graph

   Illustrated in Adobe Illustrator CS3.

2. AI: Path view of bar graph in Adobe Illustrator

   - All effects must be converted (**Object -> Convert appearance**)
   - All lines are transformed into paths (**Object -> Path -> Contour line**)
   - Do not use filters such as shading, blurring etc.

NOTES ON COMPATIBILITY

Illustrations that were created with Adobe Illustrator are in principle suitable for WPF export. However, not all Illustrator effects can become corresponding effects in Expression Design/Blend. Note:

Effect	Description
Clipping masks	Clipping masks created in Adobe Illustrator are not correctly interpreted by Expression Design. These are usually shown in Blend as areas of black color. We recommend creating illustrations without clipping masks.
Filters and effects	Not all Adobe Illustrator filters are transferred into Expression Design accordingly: Thus blurring filters, shading filters and corner effects from Illustrator do not work in Expression Design. Solution: - Most effects can be converted so that they can be read correctly by Expression Design using the **Object -> Convert appearance** command in Adobe Illustrator. - Corner effects from Adobe Illustrator are correctly interpreted by MS Design if they are converted to AI in paths.
Text fields	To be able to link text fields with code, these must be created separately in Expression Blend. "**Labels**" are required for dynamic texts; simple "**text fields**" are sufficient for static information. There is no possibility to create text labels in MS Design. These must be directly created in MS Blend.
Transparencies and group transparencies	There can be difficulties in Adobe Illustrator with the correct interpretation of transparency settings, in particular from group transparency settings. However MS Expression Blend and MS Expression Design do offer the possibility to create new transparency settings.
Multiply levels	These level settings in Adobe Illustrator are not always correctly displayed by MS Expression Blend. However, there is the possibility to "**Multiply levels**" directly in Expression Design.
Indicating instruments and standard positions	To prepare the graphics optimally for animation, the indicator and slider must always be set to the starting position, usually `0` or `12:00 o'clock`. Thus the position parameters for rotations etc. are also correct in Blend and an animation can be implemented without conversion of position data.

4.2.2 WPF export

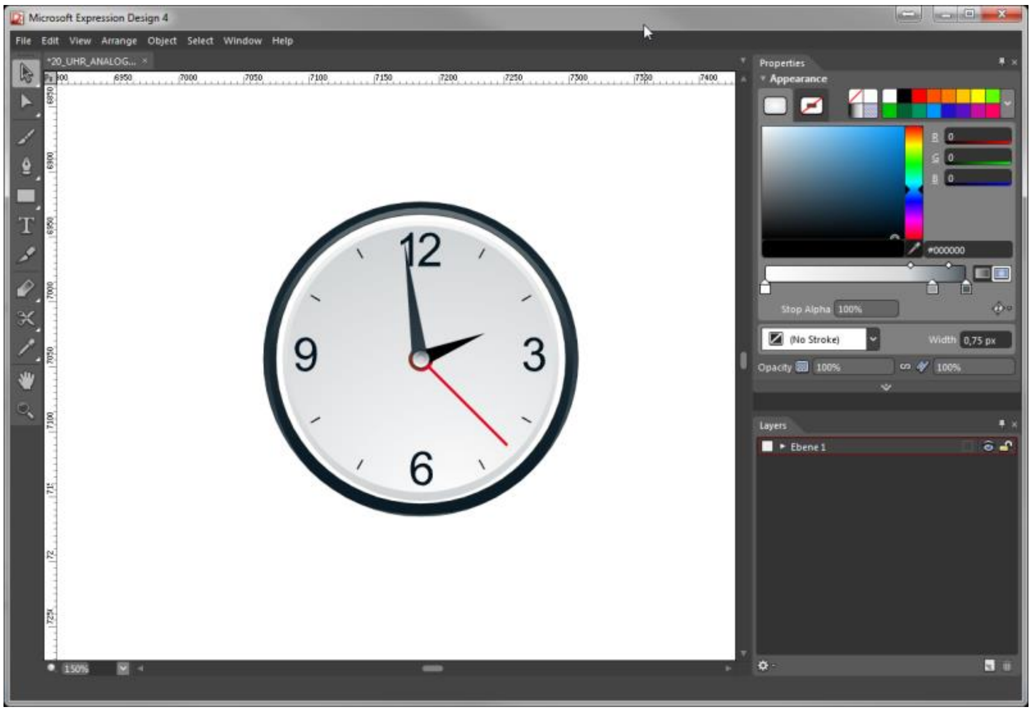
WPF files are required for animation in Microsoft Expression Blend. We recommend Microsoft Expression Design for this export, because it provides good results and most Illustrator effects are correctly interpreted.

Note: There is a free plug-in for the direct export of WPF files from Adobe Illustrator available on the internet. This plug-in provides a quick, uncomplicated way of exporting from Illustrator, however it is less suited to the current application because it lead to graphical losses. Even color deviations from the original document are possible.

Files in **.ai** format can regularly be imported into Expression Design; the paths are retained in the process.
Attention: Some common Illustrator effects cannot be displayed by Expression Design correctly however (see Illustration (on page 17) chapter).

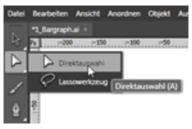
We export the pre-created bar graph element in 5 stages:

1. ED: Import
   - Import the prepared Illustrator file (on page 17) in **Microsoft Expression Design** via *File -> Import*
2. ED: Optimization



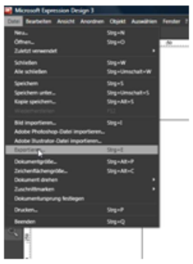
   - If the starting file is not correctly displayed in MS Expression Design, it can still be subsequently edited and optimized here

3. ED: Select



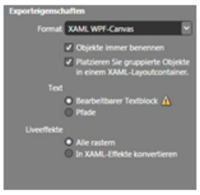
- Highlight the element for WPF export with the **direct selection** arrow in MS Expression Design; in this case it is the whole clock

4. ED: Start export


Microsoft Expression Design 3
Datei Bearbeiten Ansicht Anordnen Objekt Auswählen Fenster

- Start the export via *File -> Export*
- the dialog for configuring the export settings opens

5. ED: Export settings



Exporteigenschaften
Format: XAML WPF-Canvas
✓ Objekte immer benennen
✓ Platzieren Sie gruppierte Objekte in einem XAML-Layoutcontainer.
Text
● Bearbeitbarer Textblock
○ Pfade
Liveeffekte
● Alle rastern
○ In XAML-Effekte konvertieren

- Enter the following export settings:

a) **Format**: `XAML Silverlight 4 / WPF Canvas`

Always name objects: Activate with tick

Place the grouped object in an XAML layout container: Activate with tick

b) **Text:** `Editable text block`

c) **Line effects:** `Rasterize all`

The exported file has **.xaml** file suffix. It is prepared and animated (on page 20) in MS Expression Blend in the next stage.

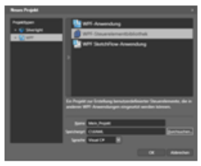
4.2.3 Animation in Blend

With MS Expression Blend:

- static XAML files from MS Expression Design are animated
- Variables for controlling effects that can be addressed by zenon are created

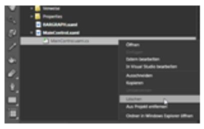
In thirteen steps, we go from a static XAML to an animated element, that can be embedded in zenon:

1. EB:create project

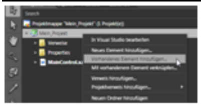
   a) Open Microsoft Expression Blend
   b) Create a new project
   c) Select the **Project type** of *WPF- >WPF Control Library*
   d) Give it a name (in our tutorial: **My_Project**)
   e) Select a location where it is to be saved
   f) Select a language (in our tutorial: C#)
   g) Select Framework Version 3.5

2. EB: delete MainControl.xaml.cs

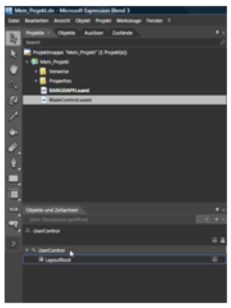
   a) Navigate to **MainControl.xaml.cs**
   b) Delete this file using the **Delete** command in the context menu

3. EB: Open exported XAML file

   a) Open the context menu for **My_Project** (right mouse button)
   b) Select **Add existing element...**
   c) Select the XAML file exported from Microsoft Expression Design, in order to open this in Microsoft Expression Blend

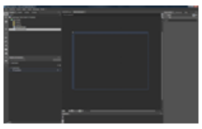
4. EB: Open MainControl.xaml

a) Open the automatically created **MainControl.xaml**

b) In the **Objects and Time axes** area, navigate to the **UserControl** entry

5. EB: Adapt XAML code



a) Click on **UserControl** with the right mouse button

b) Select **Display XAML** in the contextual menu.

c) Delete lines 7 and 9 in the XAML code:

```
x:Class="My_Project.MainControl"

d:DesignWidth="640" d:DesignHeight="480"
```

6. EB: check XAML code

- The XAML code should now look like this:

```
<UserControl
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    xmlns:d=http://schemas.microsoft.com/expression/blend/2008
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"


    mc:Ignorable="d"
    x:Name="UserControl">


    <Grid x:Name="LayoutRoot"/>
</UserControl>
```

7. EB: Copy elements

a) Open the XAML file imported from Expression Design

b) Mark all elements

c) Select **Delete** in the context menu

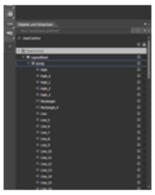
d) Change back to the automatically created XAML file

8. EB: Insert element

a) Click on **Layout Root** with the right mouse button

b) Select **Insert**

9. EB: Adapt layout type

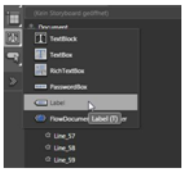


a) Click on *Layout root* -> *Change layout type* -> *Viewbox* with the right mouse button

b) The structure should now look like this: **UserControl -> LayoutRoot -> Grid -> Elements**

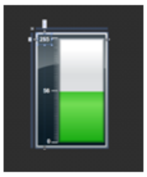
c) Give a name for **LayoutRoot** and **Grid** by double-clicking on the names

10. EB: Texts and values



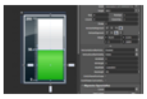
- Dynamic and static texts are labeled with text fields
- Values (numbers) are issued with **Labels**

11. EB: Insert labels



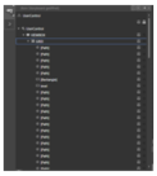
- Labels replace numbers that are to be subsequently linked using INT variables (must be carried out for all number elements)

12. EB: Set property



- To display 100%, set the bar graph element's **MaxHeight** property to `341` (the maximum height of the indicator element is `340`)

13. EB: prepare for use in zenon



a) Delete all name labels (names may only be given for elements that are to be addressed via zenon)

b) Save the XAML file with any desired name

c) Integrate the XAML file into zenon (on page 97)

A tip for checking: If the XAML file is displayed with no problems in Microsoft Internet Explorer and the window size of Internet Explorer adapts to it, it will also be correctly used in zenon.

5. Guidelines for developers

This section handles the creation of simple WPF user controls with code-behind functionality using Microsoft Visual Studio and debugging this user control in Runtime.

The following tools were used for this:

- Microsoft Visual Studio 2015
- zenon

Information

A Microsoft Visual Studio version from 2012 is recommended, due to the better-integrated XAML designer.

5.1 Creation of a simple WPF user control with code behind function

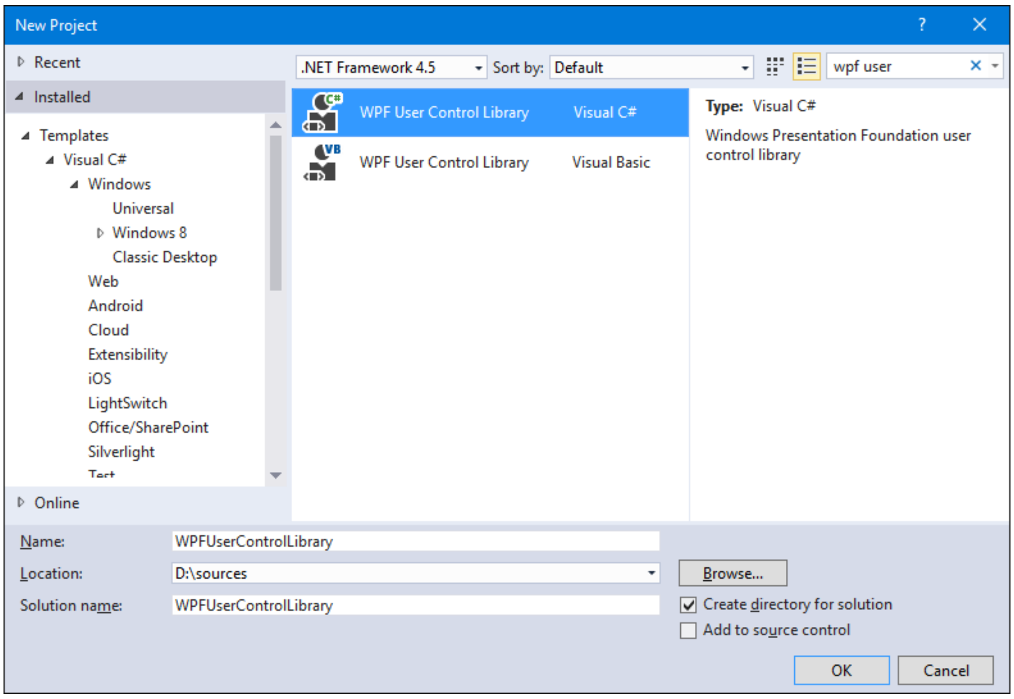
The creation and incorporation of a simple user control is described in this chapter. Because only the fundamental mechanisms/process for integration into zenon is described, the functionality of the user control is limited to the addition of two values. There is intentionally no enhanced error handling or explicit completion, in order to retain the simplicity of this example.

CREATE WPF USER CONTROL

1. Create a new **Solution** and a **WPF User Control Library** in this in Visual Studio.

   The .NET framework version 4 was selected for this example. A different version can also be selected, which must be installed on the target system on which Runtime will subsequently be started.

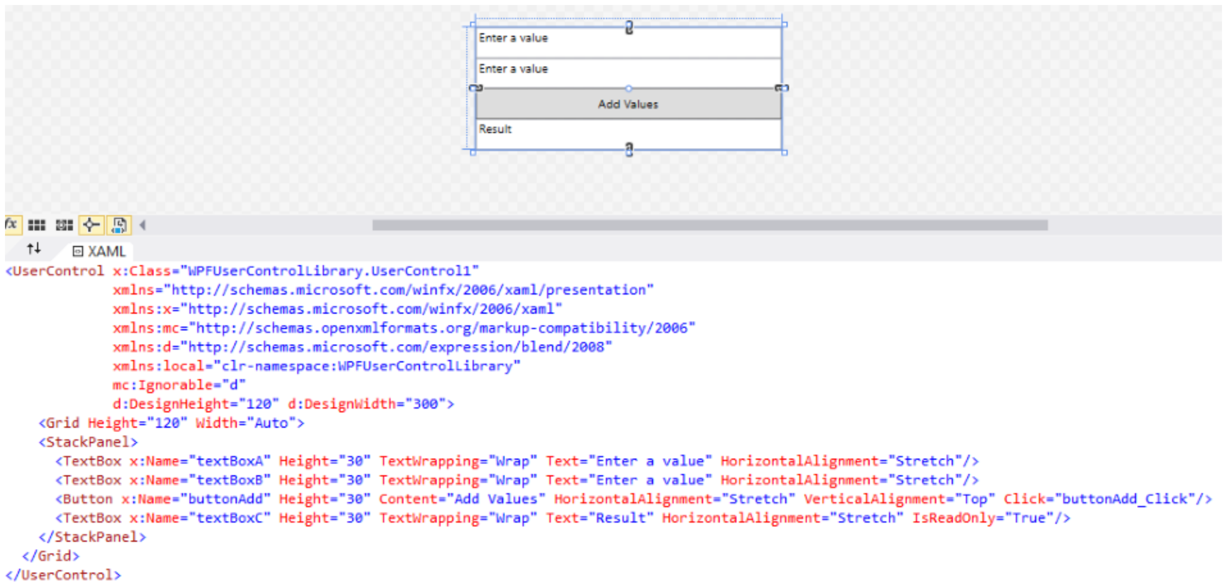
   Info: If the corresponding project template does not appear in the list of available templates, this can be added by means of the search (field at the top right of the dialog).



.NET Framework 4.5
Sort by: Default
wpf user
WPF User Control Library — Visual C#
Type: Visual C#
Windows Presentation Foundation user control library
Name: WPFUserControlLibrary
Location: D:\sources
Solution name: WPFUserControlLibrary
[✓] Create directory for solution
[ ] Add to source control

In our example, the project is given the name **WPFUserControlLibrary**.

2. Create 3 text boxes and a button in the UserControl1.xaml file:



```
<UserControl x:Class="WPFUserControlLibrary.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:WPFUserControlLibrary"
             mc:Ignorable="d"
             d:DesignHeight="120" d:DesignWidth="300">
    <Grid Height="120" Width="Auto">
    <StackPanel>
      <TextBox x:Name="textBoxA" Height="30" TextWrapping="Wrap" Text="Enter a value" HorizontalAlignment="Stretch"/>
      <TextBox x:Name="textBoxB" Height="30" TextWrapping="Wrap" Text="Enter a value" HorizontalAlignment="Stretch"/>
      <Button x:Name="buttonAdd" Height="30" Content="Add Values" HorizontalAlignment="Stretch" VerticalAlignment="Top" Click="buttonAdd_Click"/>
      <TextBox x:Name="textBoxC" Height="30" TextWrapping="Wrap" Text="Result" HorizontalAlignment="Stretch" IsReadOnly="True"/>
    </StackPanel>
  </Grid>
</UserControl>
```

3. Add the following code in the click event of the button:

```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  try
  {
    textBoxC.Text = (Convert.ToInt32(textBoxA.Text) + Convert.ToInt32(textBoxB.Text)).ToString();
  }
  catch (Exception ex)
  {
    textBoxC.Text =  "Error adding values: " + ex.Message;
  }
}
```

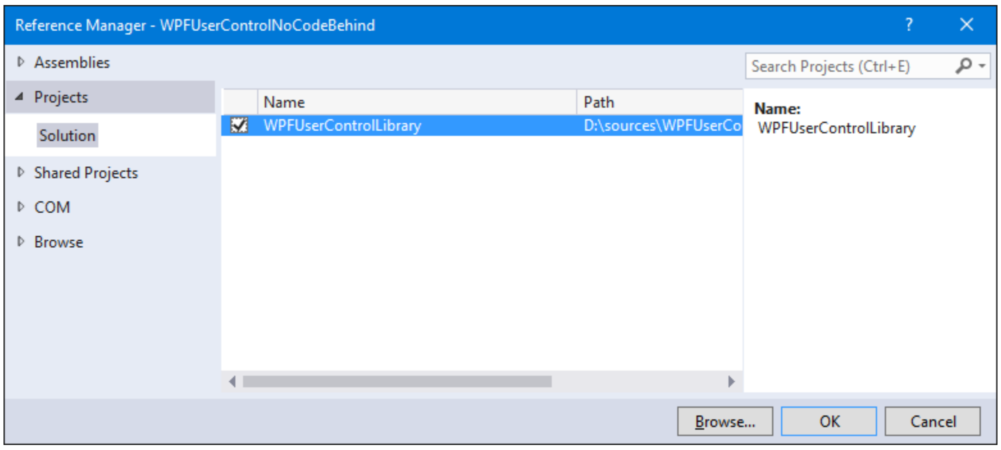
Now you have the user control with the required functionality available. However, because zenon can only display XAML files that do not link to a code-behind file, an additional XAML file is needed that references the library (assembly) that has just been built.

CREATION OF THE XAML FILE (WITHOUT CODE BEHIND) FOR ZENON

Proceed as follows to create the XAML file required in zenon.

1. Create a further project, again as a **WPF User Control Library**
2. It was called **WPFUserControlNoCodeBehind** in our example.

3. Insert a reference to the project that has just been built into this new project.



Name	Path
✓ WPFUserControlLibrary	D:\sources\WPFUserCo

Name: WPFUserControlLibrary

4. The XAML files (**UserControl1.xaml**) looks as follows:

```
<UserControl x:Class="WPFUserControlNoCodeBehind.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:WPFUserControlNoCodeBehind"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
  <Grid>

  </Grid>
</UserControl>
```

5. Because all necessary content is contained in the DLL that has been created and no code-behind file can be used, delete the following lines:

   x:Class=`"WPFUserControlNoCodeBehind.UserControl1"`

   xmlns:local=`"clr-namespace:WPFUserControlNoCodeBehind"`

6. Also delete (for the designer's size setting) the following lines:

   mc:Ignorable=`"d"`

   d:DesignHeight=`"300"` d:DesignWidth=`"300"`

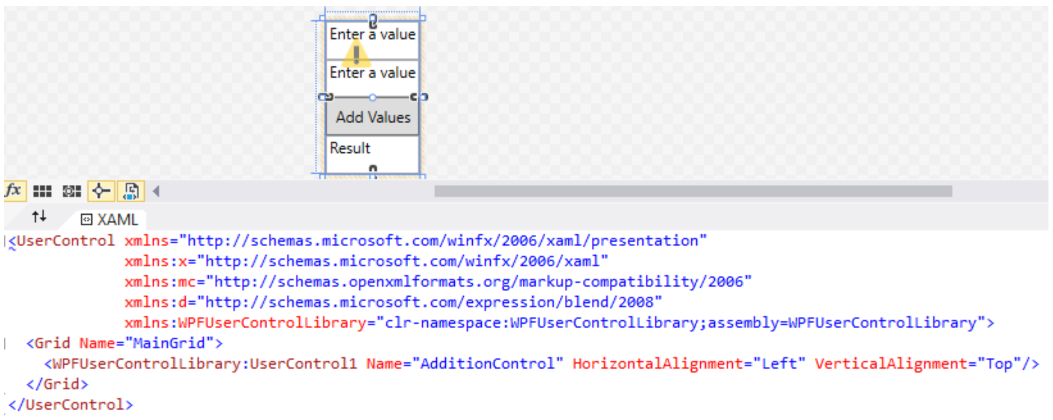
7. Delete the code-behind file (**UserControl1.xaml.cs**) in this project.

8. Drag the user control that has been created beforehand (for the project **WPFUserControlLibrary**) over the toolbox in the XAML designer.

9. Assign a name for the grid and the user control.

   Attention: If no name is given here, these elements do not appear in the linking dialog in the zenon Editor and thus cannot be made dynamic.

10. The XAML file should now look as follows:

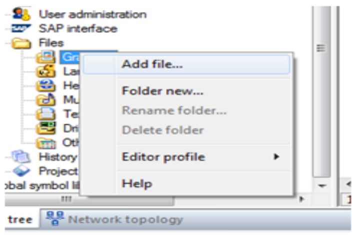


```
<UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:WPFUserControlLibrary="clr-namespace:WPFUserControlLibrary;assembly=WPFUserControlLibrary">
  <Grid Name="MainGrid">
    <WPFUserControlLibrary:UserControl1 Name="AdditionControl" HorizontalAlignment="Left" VerticalAlignment="Top"/>
  </Grid>
</UserControl>
```

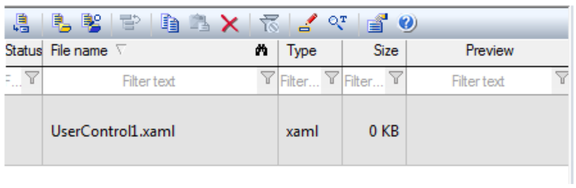
In the next step, how the DLL and XAML file are incorporated into zenon is explained.

STEPS IN ZENON

1. Open the zenon Editor
2. Go to `File -> Graphics`.
3. Select **Add file...** in the context menu



4. Select the XAML file at the save location (**UserControl1.xaml** from the **WPFUserControlNoCodeBehind** project) and add this:



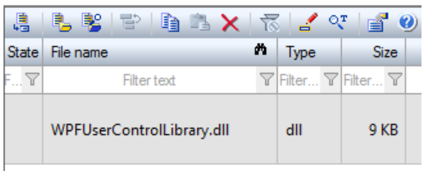
Status	File name	Type	Size	Preview
	Filter text	Filter...	Filter...	Filter text
	UserControl1.xaml	xaml	0 KB	

5. Insert the DLL with the functionality for the XAML file.

   To do this:

   a) Select, in the context menu, File -> Other**Add file...**.

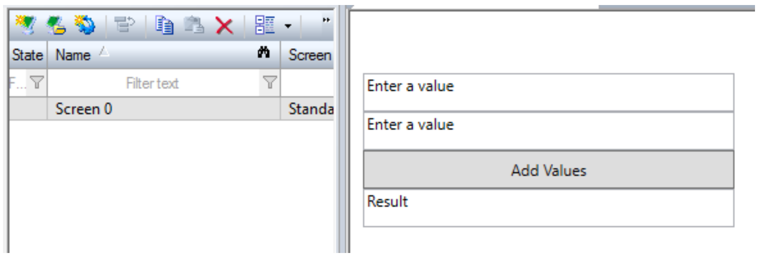
b) Select the file **WPFUserControlLibrary.dll** (from the output path) of the first project (**WPFUserControlLibrary**).



State	File name	Type	Size
	WPFUserControlLibrary.dll	dll	9 KB

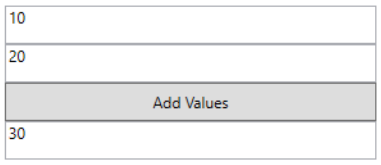
6. Create a zenon screen.
7. Add a WPF element and select the previously-incorporated XAML file.

   You should now see the following in the zenon Editor:



State	Name	Screen
	Screen 0	Standa

8. Start zenon Runtime in order to also test the control there.



Information

The XAML file and referenced assemblies can also be saved in complied form as a *.cdwpf file. Only one file thus need to be imported in the Editor (under `Files -> Graphics`). Further information on this can be found in the CDWPF files (collective files) (on page 49) chapter.

Hint: When developing a WPF user control, it is usually more practical to insert the XAML file and the referenced DLL(s) separately. This makes the replacement of the DLL and debugging easier. Further information on the topic of debugging in the Debugging the WPF user control in Runtime (on page 30).

Hint

DLLs that belong to a WPF element (referenced by the linked XAML file) can also be replaced in the Editor during ongoing operation.
To replace a DLL:

- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In Explorer, replace the DLL in the `\wpfache` folder of the Editor files.
  You can find this folder in the SQL directory under
  `...\PROJECT-GUID\FILES\zenon\custom\wpfcache`

As an alternative to replacement using Explorer, you can also replace the file in the zenon Editor directly; to do this:

- In the Visual Studio project settings, increase the file version of the DLL.
- Create the new DLL.
- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In the zenon Editor, delete the DLL from the `\Files\Other` folder and add the file with the higher version number.

Further examples can be found in the Examples: Integration of WPF into zenon (on page 97) chapter.

5.2 Debugging the WPF user control in Runtime

To debug the WPF user control in Runtime, proceed as follows.

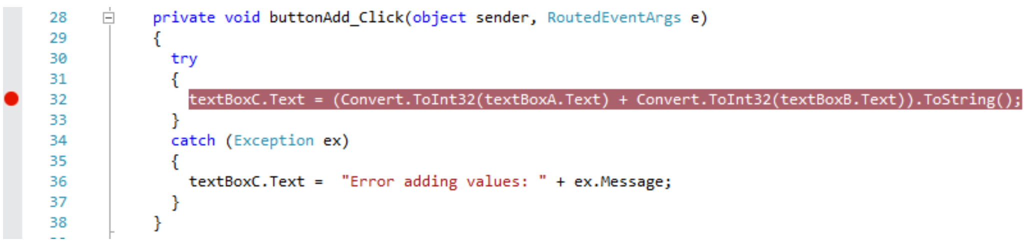
In this example, the control described in the Creation of a simple WPF user controls with code behind function (on page 24) is used.

DEBUGGING BY MEANS OF ATTACH TO PROCESS

1. Ensure that zenon Runtime has been started and a screen with the WPF user control is open.

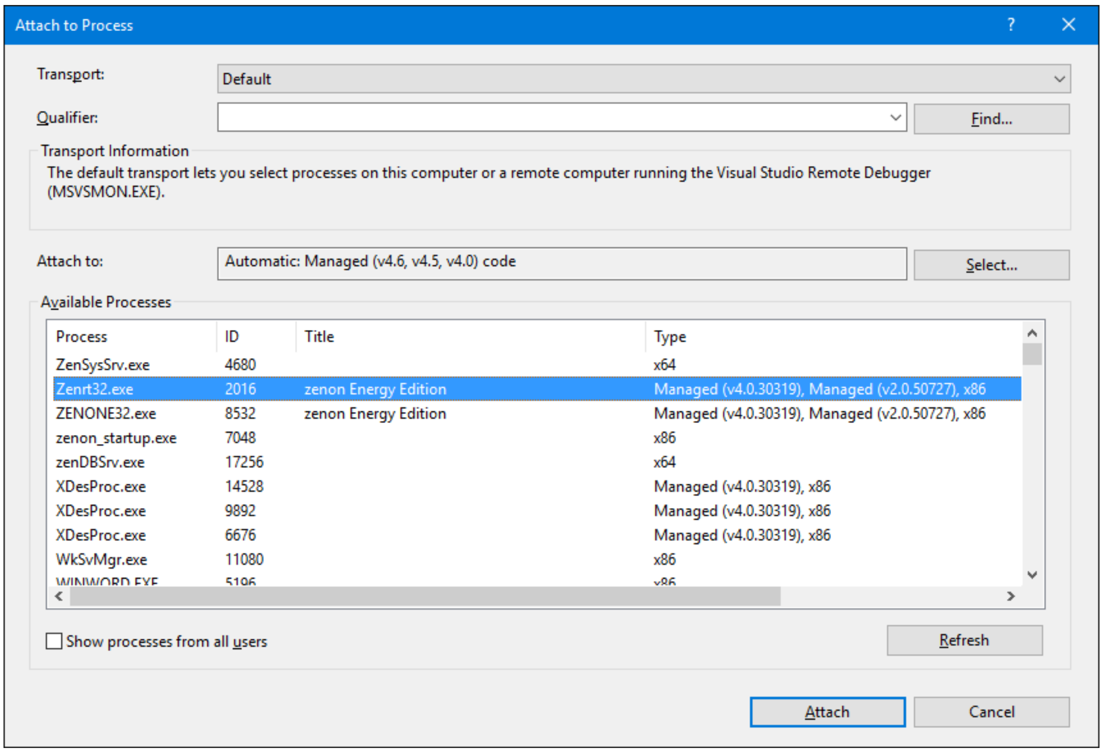
   Furthermore, ensure that the DLL that is currently being used corresponds to the build (Version) of the user control project (**WPFUserControlLibrary**).

2. Set a breakpoint in the click event of the button in the Visual Studio project



```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  try
  {
    textBoxC.Text = (Convert.ToInt32(textBoxA.Text) + Convert.ToInt32(textBoxB.Text)).ToString();
  }
  catch (Exception ex)
  {
    textBoxC.Text =  "Error adding values: " + ex.Message;
  }
}
```

3. In Visual Studio, under **Debug** , select the **Attach to Process** menu item.
4. Select the zenon Runtime process



Attach to Process

Transport: Default

Qualifier:

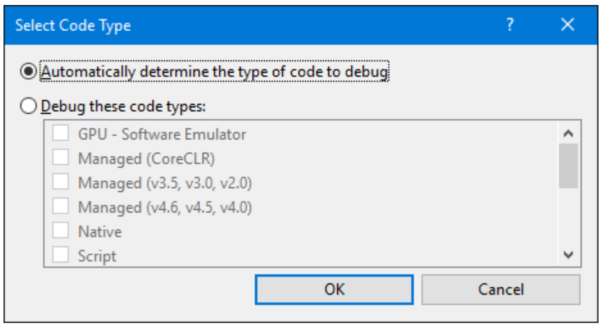
Transport Information
The default transport lets you select processes on this computer or a remote computer running the Visual Studio Remote Debugger (MSVSMON.EXE).

Attach to: Automatic: Managed (v4.6, v4.5, v4.0) code

Available Processes

Process	ID	Title	Type
ZenSysSrv.exe	4680		x64
Zenrt32.exe	2016	zenon Energy Edition	Managed (v4.0.30319), Managed (v2.0.50727), x86
ZENONE32.exe	8532	zenon Energy Edition	Managed (v4.0.30319), Managed (v2.0.50727), x86
zenon_startup.exe	7048		x86
zenDBSrv.exe	17256		x64
XDesProc.exe	14528		Managed (v4.0.30319), x86
XDesProc.exe	9892		Managed (v4.0.30319), x86
XDesProc.exe	6676		Managed (v4.0.30319), x86
WkSvMgr.exe	11080		x86
WINWORD.EXE	5196		x86

5. Under **Attach to**, select either **Automatic** or the corresponding .NET framework version (**v4.x** in this case)



Select Code Type

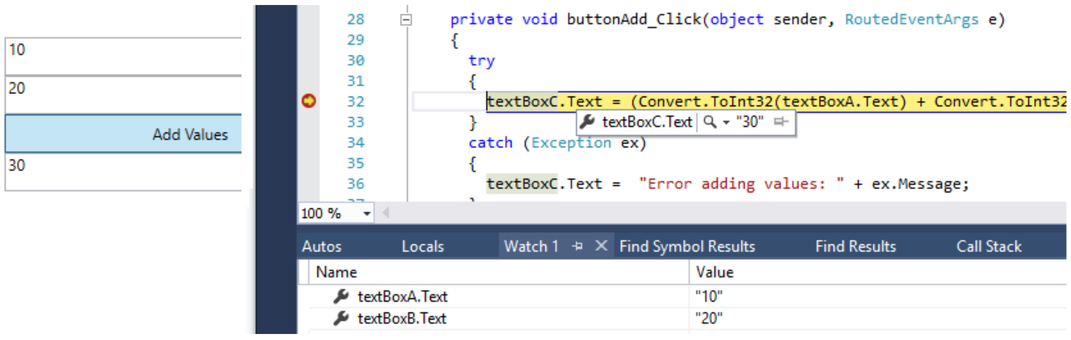
(●) Automatically determine the type of code to debug

( ) Debug these code types:

- [ ] GPU - Software Emulator
- [ ] Managed (CoreCLR)
- [ ] Managed (v3.5, v3.0, v2.0)
- [ ] Managed (v4.6, v4.5, v4.0)
- [ ] Native
- [ ] Script

6. Click on **Attach**.

7. Now trigger the breakpoint in which you enter values into the WPF control in zenon Runtime and click on the button

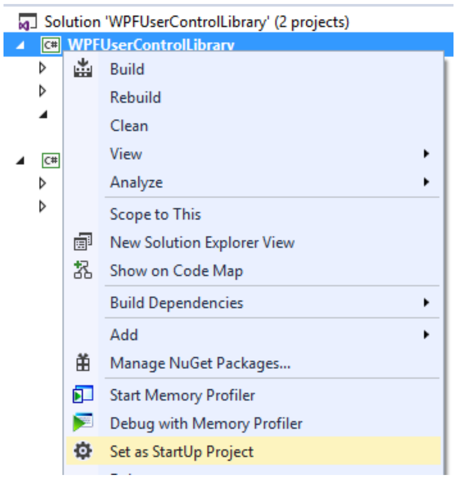


```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  try
  {
    textBoxC.Text = (Convert.ToInt32(textBoxA.Text) + Convert.ToInt32
  }   textBoxC.Text  "30"
  catch (Exception ex)
  {
    textBoxC.Text = "Error adding values: " + ex.Message;
```

Name	Value
textBoxA.Text	"10"
textBoxB.Text	"20"

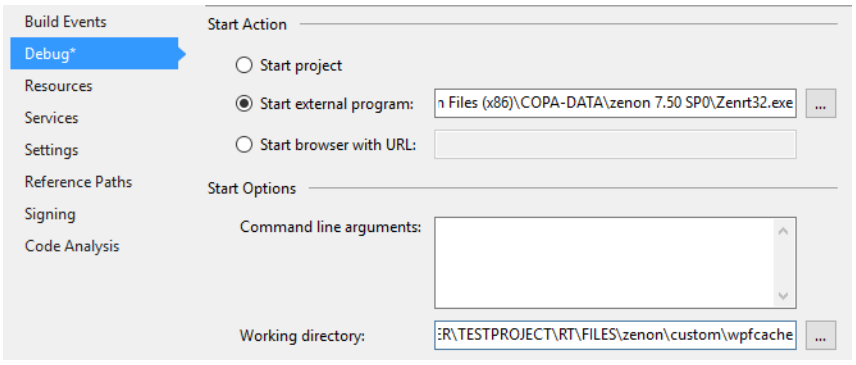
DEBUG USING START EXTERNAL PROGRAM

1. Ensure that zenon Runtime has been closed.
2. Ensure that, in the zenon Editor, the project that contains the WPF user control has been set as the start project.
3. Ensure that the user control project (**WPFUserControlLibrary**) is set as the start project in Visual Studio.



4. In the project properties of the Visual Studio project, select under **Debug**, for **Start action**: **Start external program**
5. For **Start external program**, select the path of the zenon Runtime application.
6. Under Working Directory, select the **\wpfcache** folder of the Runtime files (**...\PROJECTNAME\RT\FILES\zenon\custom\wpfcache**)

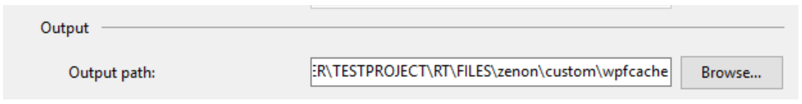
Hint: In the selected project in the zenon Editor, press the keyboard combination **CTRL+ALT+R** in order to jump directly to the root directory of the Runtime files.



[✓ Start external program:] n Files (x86)\COPA-DATA\zenon 7.50 SP0\Zenrt32.exe

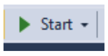
Working directory: :R\TESTPROJECT\RT\FILES\zenon\custom\wpfcache

7. In the project properties, enter **\wpfache** folder of the Runtime files as the **Output path** under **Build** .

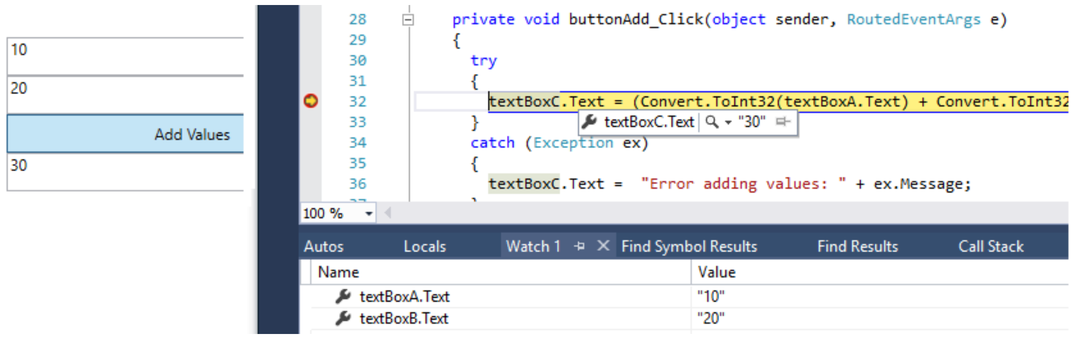


Output path: :R\TESTPROJECT\RT\FILES\zenon\custom\wpfcache

8. Create the project in Visual Studio
9. Start debugging in Visual Studio with **Start**



10. zenon Runtime is now started automatically.
11. Trigger the breakpoint by entering values in the WPF control in zenon Runtime and click on the button



```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  try
  {
    textBoxC.Text = (Convert.ToInt32(textBoxA.Text) + Convert.ToInt32
  }                       textBoxC.Text  "30"
  catch (Exception ex)
  {
    textBoxC.Text = "Error adding values: " + ex.Message;
```

Name	Value
textBoxA.Text	"10"
textBoxB.Text	"20"

Information

When starting zenon Runtime, the assemblies (DLLs) referenced in the WPF user controls from the **\FILES\zenon\custom\additional** folder, and/or the assemblies from **CDWPF** files in the **\FILES\zenon\custom\wpfcache** folder are copied. If the file version of the DLL in the **\wpfache** folder is one higher than the version of the "original file", it is not replaced!

For debugging, it is thus sufficient to only replace the file that is on the **\wpfache** folder directly.

For delivery, it must be ensured that the current version of the DLL is present in the **\additional** folder or the **CDWPF** file!

Attention: If only the DLL is updated in the **\additional** folder or in the **CDWPF**, but the version number is not increased, the DLL must be deleted manually in the **\wpcache** folder, because it is not updated otherwise (due to the above-described mechanism).

Hint

DLLs that belong to a WPF element (referenced by the linked XAML file) can also be replaced in the Editor during ongoing operation.
To replace a DLL:

- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In Explorer, replace the DLL in the `\wpfache` folder of the Editor files.
  You can find this folder in the SQL directory under
  `...\PROJECT-GUID\FILES\zenon\custom\wpfcache`

As an alternative to replacement using Explorer, you can also replace the file in the zenon Editor directly; to do this:

- In the Visual Studio project settings, increase the file version of the DLL.
- Create the new DLL.
- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In the zenon Editor, delete the DLL from the `\Files\Other` folder and add the file with the higher version number.

5.3 Data exchange between zenon and WPF user controls

There are different possibilities for exchanging data between zenon and WPF user controls.

- Data exchange using dependency properties (on page 35)
- Data replacement via VSTA (on page 39)

5.3.1 Data exchange using dependency properties

The most elegant and secure way to exchange data between zenon and self-created WPF user controls is by using `Dependency Properties`.

The WPF user control project created in the Creating a simple WPF user controls with code behind function (on page 24) serves as a basis (**WPFUserControlLibrary**).

In this chapter, the focus is purely on the core theme (`Dependency Properties` and data exchange between the user control and zenon in this case). Specific WPF features such as Databinding, etc., as well as explicit error handling, are not covered.

ADDITIONS TO THE CODE

1. Create the `TextChanged` Event for the `textBoxA` element in the **UserControl1.xaml** file

   `TextChanged="textBoxA_TextChanged"`

2. Add the following lines of code in the `UserControl1` class of the code behind file (**UserControl1.xaml.cs**)

```
/// <summary>
/// Gets or sets the ValueA.
/// </summary>
public double ValueA
{
  get
  {
    return (double)GetValue(ValueADependencyProperty);
  }
  set
  {
    SetValue(ValueADependencyProperty, value);
  }
}
```

```
    /// <summary>
    /// Dependency property for ValueA
    /// </summary>
    public static readonly DependencyProperty ValueADependencyProperty =
         DependencyProperty.Register("ValueA", typeof(double),
         typeof(UserControl1), new FrameworkPropertyMetadata(0.0, new
PropertyChangedCallback(OnValueADependencyPropertyChanged)));

    /// <summary>
    /// Called when [value a dependency property changed].
    /// </summary>
    /// <param name="source">The source.</param>
    /// <param name="e">The <see cref="DependencyPropertyChangedEventArgs"/> instance
containing the event data.</param>
    private static void OnValueADependencyPropertyChanged(DependencyObject source,
DependencyPropertyChangedEventArgs e)
    {
      UserControl1 control = source as UserControl1;
      if (control != null)
      {
        try
        {
          control.ValueA = (double)e.NewValue;
          control.textBoxA.Text = control.ValueA.ToString();
        }
        catch (Exception)
        {}
      }
    }

    /// <summary>
    /// Handles the TextChanged event of the textBoxA control.
    /// </summary>
    /// <param name="sender">The source of the event.</param>
    /// <param name="e">The <see cref="TextChangedEventArgs"/> instance containing the
event data.</param>
    private void textBoxA_TextChanged(object sender, TextChangedEventArgs e)
    {
      try
```

```
        {
          ValueA = Convert.ToDouble(textBoxA.Text);
        }
        catch (Exception)
        {}
      }
```

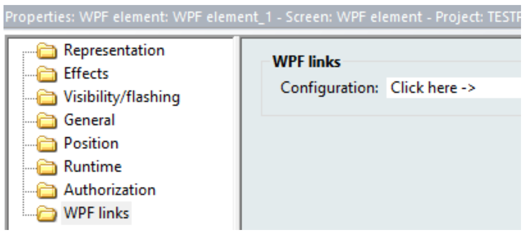
Then build the solution.

Information

A numerical property (`double`) is used in this example. Other simple data types (such as `bool`, `string`, `int`, etc.) can also be used.

LINKING IN ZENON

1. Update the WPF user control (DLL) in the zenon Editor.
2. Proceed as described in the creation of a simple WPF user controls with code behind function (on page 24) chapter.
3. Create a numeric variable in zenon. Link this variable to a dynamic text element. You place the dynamic text element in the screen next to the WPF element with your user control.
4. Open the screen that contains the WPF element and select, for the WPF element, under **WPF links**: **Configuration**

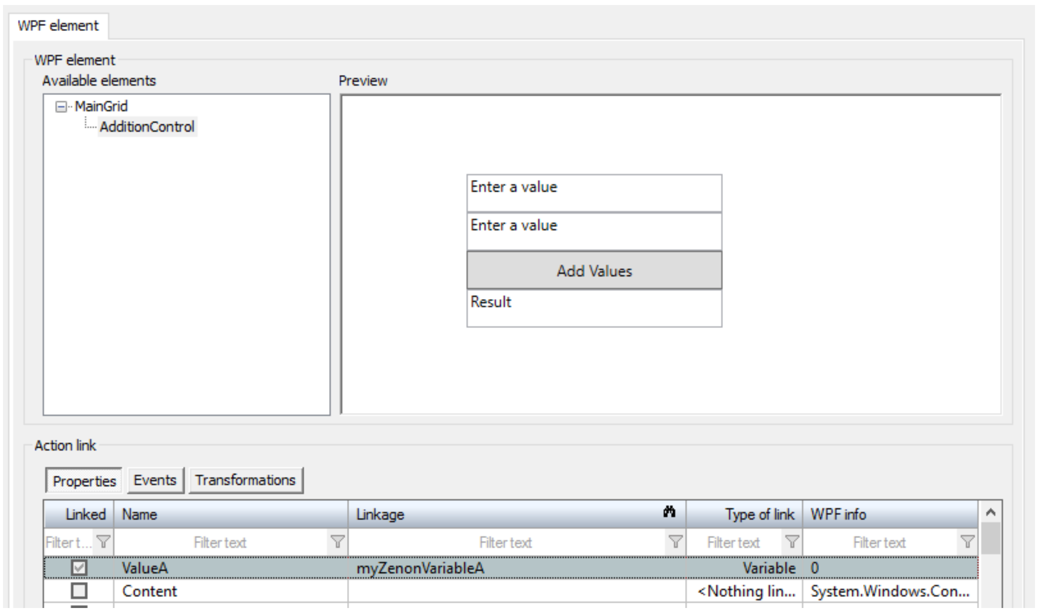


Properties: WPF element: WPF element_1 - Screen: WPF element - Project: TESTP

WPF links

Configuration: Click here ->

5. Expand the node in the tree at the top left and select **AdditionControl**

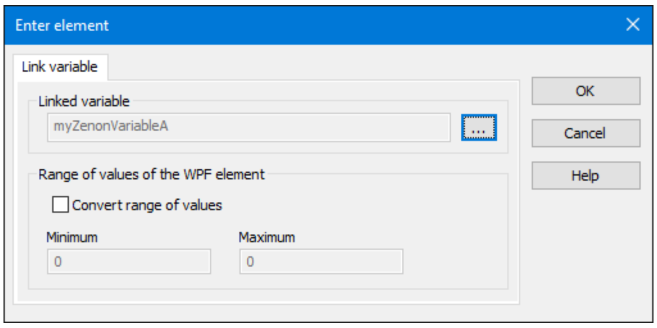


Linked	Name	Linkage	Type of link	WPF info
☑	ValueA	myZenonVariableA	Variable	0
☐	Content		<Nothing lin...	System.Windows.Con...

6. Select the line with `ValueA` (this is the name of the property that was created in the code beforehand) and select, for **Type of link:**, **Variable**.

   Hint: Give Properties a prefix so that this can be found more easily, for example: `_ValueA`

7. In the column under **Linkage**, print out the variable that was created in zenon beforehand



Linked variable: myZenonVariableA

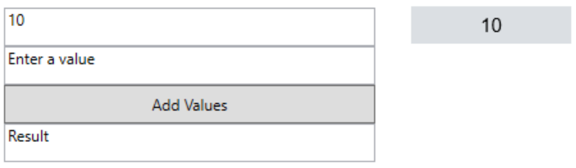
Range of values of the WPF element

☐ Convert range of values

Minimum: 0
Maximum: 0

8. Confirm the dialog with OK and build the Runtime files

9. Start Runtime in order to test the WPF user control



10. If the value is changed in user control, the value automatically changes in zenon and vice versa.

11. Of course you can debug the control as described in the Debugging the WPF user control in Runtime (on page 30) chapter, as well as create further dependency properties.

Information

The `UserControl_Loaded` event can be used in order to (automatically) access the values of the dependency property during initialization (when calling up the user control) for example.

5.3.2 Data replacement via VSTA

Data can also be exchanged between zenon and WPF user controls using VSTA.

The API element methods

- `get_WPFProperty` (reading of values)
- `set_WPFProperty` (writing of values)

are used for this.

The example used here is based on the example used in the Data exchange using dependency properties (on page 35) chapter.

CREATION OF A VSTA MACRO FOR DATA EXCHANGE BETWEEN ZENON AND THE WPF USER CONTROL

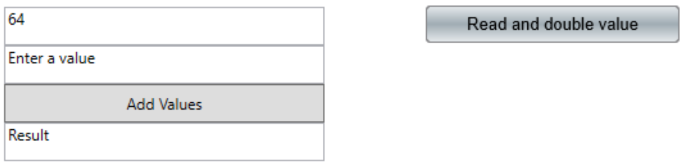
1. Create the following VSTA macro in the project add-in of the zenon project

```
/// <summary>
/// Sample Macro for data exchange between VSTA and a WPF User Control
/// </summary>
public void MacroWPFAccess()
{
    //Get the Screen and Element hosting the WPF User Control
    zenOn.IDynPicture myWPFScreen = this.DynPictures().Item("Screen");
    zenOn.IElement myWPFElement = myWPFScreen.Elements().Item("WPF_Element");

    //Read the current value from the WPF Element property
    double currentValue = Convert.ToDouble(myWPFElement.get_WPFProperty("AdditionControl", "ValueA"));

    //Double the value and write it back to the WPF Element property
    myWPFElement.set_WPFProperty("AdditionControl", "ValueA", currentValue * 2);
}
```

Whereby:

- `"Screen"` is the name of the zenon screen in which the WPF element is located
- `"WPF_Element"` is the name of the WPF element that contains the WPF user control
- `"AdditionControl"` is the name of the WPF user controls itself (defined in the (**UserControl1.xaml** file)
- `"ValueA"` is the name of the user control property

2. Create an execute VSTA macro function and link this to a button in the screen in which the WPF element is also located
3. Start Runtime to test changes



When executing the macro, the value is read by the control, doubled and written back.

Information

The user control properties used for this method of data exchange need not necessarily be dependency properties, as outlined in this example. "Standard" properties can also be used, see in relation to this the Access via VSTA "variable link" (on page 40) chapter.

5.4 Access to the zenon (Runtime) object model from a WPF user control

There are different possibilities for access to the zenon object model from a WPF user control. This is explained in more detail in the following chapters.

Attention

When using zenon COM objects with self-created user controls or external applications, they must be enabled using the ***Marshal.ReleaseComObject*** *method. Enabling by means of the* ***Marshal.FinalReleaseComObject*** *method must not be used, because this leads to a malfunction of zenon add-ins.*

5.4.1 Access via VSTA "variable link"

In order to get access to the zenon Runtime COM interface by means of "variable link", proceed as follows. The creation of a simple WPF user controls with code behind function (on page 24) serves as an initial example.

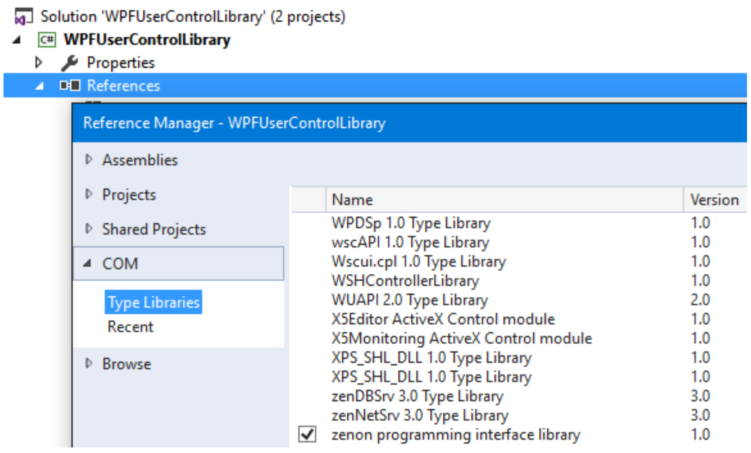
Information

The following code is intended to show an example of how the COM implements access to zenon Runtime and in doing so limits itself to the basic functionality. There is no explicit error handling, etc.

NECESSARY AMENDMENTS IN WPF USER CONTROL

The following steps are necessary in the WPF user control project (**WPFUserControlLibrary**).

Firstly, a reference to the zenon COM interface must be incorporated.


Solution 'WPFUserControlLibrary' (2 projects)
WPFUserControlLibrary
Properties
References

Reference Manager - WPFUserControlLibrary
Assemblies
Projects
Shared Projects
COM
Type Libraries
Recent
Browse

	Name	Version
	WPDSp 1.0 Type Library	1.0
	wscAPI 1.0 Type Library	1.0
	Wscui.cpl 1.0 Type Library	1.0
	WSHControllerLibrary	1.0
	WUAPI 2.0 Type Library	2.0
	X5Editor ActiveX Control module	1.0
	X5Monitoring ActiveX Control module	1.0
	XPS_SHL_DLL 1.0 Type Library	1.0
	XPS_SHL_DLL 1.0 Type Library	1.0
	zenDBSrv 3.0 Type Library	3.0
	zenNetSrv 3.0 Type Library	3.0
✓	zenon programming interface library	1.0

After this, the following code must be inserted in the `UserControl1` class:

```
//The zenon Project
zenOn.Project zenonProject = null;

/// <summary>
/// Property for the Variable link via VSTA
/// </summary>
public object zenonVariableLink
{
  get { return null; }
  set
  {
    if (value != null && zenonProject == null)
    {
      zenOn.Variable zenonVariable;
      try
      {
        zenonVariable = (zenOn.Variable)value;
      }
      catch (Exception)
      {
        return;
      }
      if ((zenonVariable!= null) && (!string.IsNullOrEmpty(zenonVariable.Name)))
      {
        zenonProject = zenonVariable.Parent.Parent;
      }
    }
  }
}

/// <summary>
/// Trigger used to notify the control from VSTA to release the COM resources
/// </summary>
public object zenonReleaseTrigger
{
  get { return null; }
  set
  {
    if ((bool)value && zenonProject != null)
    {
      try
      {
        Marshal.ReleaseComObject(zenonProject);
      }
      catch (Exception)
      {
        return;
      }
      zenonProject = null;
      GC.Collect();
      GC.WaitForPendingFinalizers();
      GC.Collect();
```

```
    }
  }
}
```

Whereby access to the properties `zenonVariableLink` (to initialize the COM object) and `zenonReleaseTrigger` (to unlock the COM object) are subsequently accessed from VSTA (write).

In order to test the COM access quickly very easily, it is possible to insert the following line of code in the existing button-click event of the user control.

```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  if (zenonProject != null)
  {
    MessageBox.Show(zenonProject.Name);
  }
  return;
  ...
```

Information

A `zenOn.Project` variable is used in this example. Of course other objects such as events, etc. of the zenon object model can also be used.

NECESSARY AMENDMENTS IN THE ZENON PROJECT/VSTA CODE

The following steps are necessary in the VSTA code:

Creation of a VSTA macro for the initialization

```
/// <summary>
/// Macro for API initialization in the WPF User Control
/// </summary>
public void MacroWPFInit()
{
  zenOn.IDynPicture myWPFScreen = this.DynPictures().Item("Screen");
  zenOn.IElement myWPFElement = myWPFScreen.Elements().Item("WPF_Element");
  myWPFElement.set_WPFProperty("AdditionControl", "zenonVariableLink",
this.Variables().Item(0));
}
```

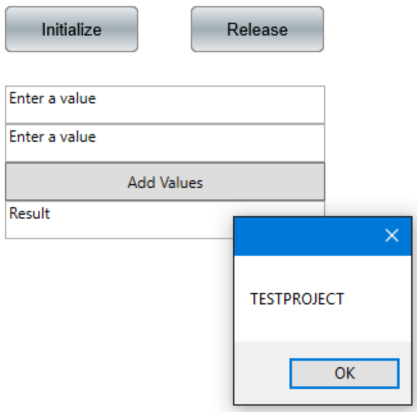
Creation of a VSTA macro for approval

```
/// <summary>
/// Macro for API release in the WPF User Control
/// </summary>
public void MacroWPFRelease()
{
  zenOn.IDynPicture myWPFScreen = this.DynPictures().Item("Screen");
  zenOn.IElement myWPFElement = myWPFScreen.Elements().Item("WPF_Element");
  myWPFElement.set_WPFProperty("AdditionControl", "zenonReleaseTrigger", true);
}
```

Create two execute VSTA macro functions that are linked with buttons, which are in the same screen as the WPF element.

Now start Runtime in order to test the functionality

- Execute the macro for initialization
- Click on the button in the WPF user control; a message box with the project name of the project appears



TESTPROJECT

- Execute the macro for release

In order to debug the user control, it is possible to proceed as described in the Debugging the WPF user control in Runtime (on page 30).

Hint

The initialization and release of the COM object in this example is only carried out for simple demonstration using VSTA macro functions. Depending on the application, and/or in practice, events in VSTA are better suited to this.

For example, the code for initialization in the `_Open` event of the screen can be executed with the WPF element and the code for release in the `_Close` event.

The mechanism described here is also used in the Display of WPF elements in the zenon Web Client (on page 93) chapter.

Attention

If COM objects are used in WPF user controls, these must always be explicitly approved before destroying the WPF user control (before closing the screen, before closing Runtime, before reloading).

5.4.2 Access via marshaling

In order to get access to the zenon Runtime COM interface by means of marshaling, proceed as follows. The creation of a simple WPF user controls with code behind function (on page 24) serves as an initial example.

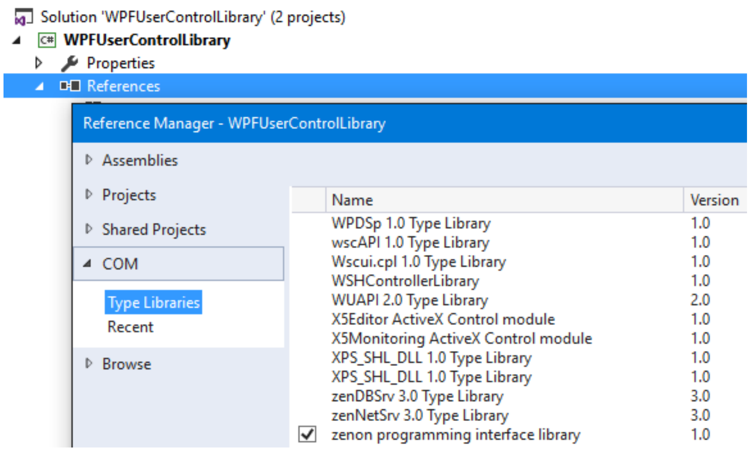
Information

The following code is intended to show an example of how the COM implements access to zenon Runtime and in doing so limits itself to the basic functionality. There is no explicit error handling, etc.

NECESSARY AMENDMENTS IN WPF USER CONTROL

The following steps are necessary in the WPF user control project (**WPFUserControlLibrary**).

Firstly, a reference to the zenon COM interface must be incorporated.



Solution 'WPFUserControlLibrary' (2 projects)
WPFUserControlLibrary
Properties
References

Reference Manager - WPFUserControlLibrary

	Name	Version
	WPDSp 1.0 Type Library	1.0
	wscAPI 1.0 Type Library	1.0
	Wscui.cpl 1.0 Type Library	1.0
	WSHControllerLibrary	1.0
	WUAPI 2.0 Type Library	2.0
	X5Editor ActiveX Control module	1.0
	X5Monitoring ActiveX Control module	1.0
	XPS_SHL_DLL 1.0 Type Library	1.0
	XPS_SHL_DLL 1.0 Type Library	1.0
	zenDBSrv 3.0 Type Library	3.0
	zenNetSrv 3.0 Type Library	3.0
✓	zenon programming interface library	1.0

After this, the following code must be inserted in the `UserControl1` class:

```
//The zenon Project
zenOn.Project zenonProject = null;
```

Furthermore, the constructor of the user controls must be supplemented with the lines below (to initialize the COM object):

```
/// <summary>
/// Constructor for UserControl1, initialize COM Object
/// </summary>
public UserControl1()
{
  InitializeComponent();

  try
  {
    zenonProject =
((zenOn.Application)Marshal.GetActiveObject("zenOn.Application")).Projects().Item("TES
TPROJECT");
  }
  catch (Exception)
  {
  }
}
```

The COM object must be approved in the `UserControl_Unloaded` event:

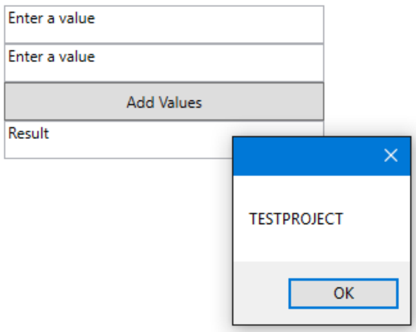
```
/// <summary>
/// Release COM Object
/// </summary>
private void UserControl_Unloaded(object sender, RoutedEventArgs e)
{
  try
  {
    if (zenonProject != null)
    {
      Marshal.ReleaseComObject(zenonProject);
      zenonProject = null;
    }
  }
  catch (Exception)
  {
  }
}
```

In order to test the COM access quickly very easily, it is possible to insert the following line of code in the existing button click event of the user control.

```
private void buttonAdd_Click(object sender, RoutedEventArgs e)
{
  if (zenonProject != null)
  {
    MessageBox.Show(zenonProject.Name);
  }
  return;

  ...
```

Now build the solution and update the WPF user control in the zenon project.

Start Runtime to test the user control.



In order to debug the user control, it is possible to proceed as described in the Debugging the WPF user control in Runtime (on page 30).

Information

A `zenOn.Project` variable is used in this example. Of course other objects such as events, etc. of the zenon object model can also be used.

Attention

If COM objects are used in WPF user controls, these must always be explicitly approved before destroying the WPF user control (before closing the screen, before closing Runtime, before reloading).

Information

No access by means of marshaling is possible in the zenon web client. If access to the COM interface is required there, the method described in the Access via VSTA "variable link" (on page 40) must be used.

6. Engineering in zenon

In order to be able to use WPF user controls in zenon, version 3.5 (or higher, depending on the .NET framework version used in the user control) of the Microsoft framework must be used on both the Editor computer and the Runtime computer.

CONDITIONS FOR THE WPF DISPLAY IN ZENON

The dynamization is currently available for simple variable types (numerical data types as well as string). Arrays and structures cannot be dynamized.

Therefore the following WPF functions can be implemented in zenon:

- Element properties that correspond to simple data types, such as S`String, Int, Double, Bool` etc.
- Element properties of the "Object" type, which can be set with simple data types
- Element events can be used with functions; the parameters of the events are not however available in and cannot be evaluated in zenon
- Element transformation, for which a **RenderTransform** is present for the element in the XAML file

  Attention: if the content is outside of the area of the WPF element during transformation, this is not labeled

Notes on dBase: No shade can be displayed in zenon for WPF elements.

Attention

*If the Runtime files were created for a project for a version before 6.50, existing **WPF elements** are not included into Runtime screens.*

6.1 CDWPF files (collective files)

A CDWPF file (with the suffix ***.cdwpf**) is an renamed ZIP file that contains the following components:

- XAML file (to reference the user control assembly)
- DLL file (the actual WPF user control, optional)
- Preview graphics (for preview, optional)

Rules for the use of collective files:

- The files (XAML, DLL, preview graphics) can be in the CDWPF file directly or in a joint folder.
- The name of the collective file should correspond to the names of the XAML file.
- Only one XAML file may be contained.
- The preview graphic should be small and no more than 64 pixels high.
  Name of the preview file: **preview.png** or the name of the XAML file with the suffix **png**.

- Any number of assemblies can be used. The distinction is made on the basis of the file version.
- Collective files do not need to contain an assembly.
- All subfolders are examined and only taken into account with ***.dll**, ***.xaml** or ***.png** files.
- If a collective file (***.cdwpf**) is replaced by a file with a different version, all corresponding CDWPF files in all symbols and images in all projects must be adapted.

6.2 create WPF element

To create a WPF element

1. In the elements toolbar, select the symbol for **WPF element** or the **Elements** entry in the menu
2. Select the start point in the main window.
3. Pull open the element with the mouse.
4. In properties, select **Representation** the property **XAML file** in the group.
5. The file selection dialog opens.
6. Select the desired file
   Files of the following formats are valid:
   - *.xaml: Extensible Application Markup Language
   - *.cdwpf: WPF collective file, also shows preview image

   (The file must already be present in the Project Manager under **Files/graphics** or created in the dialog.)
7. Configure the links (on page 50).

Information

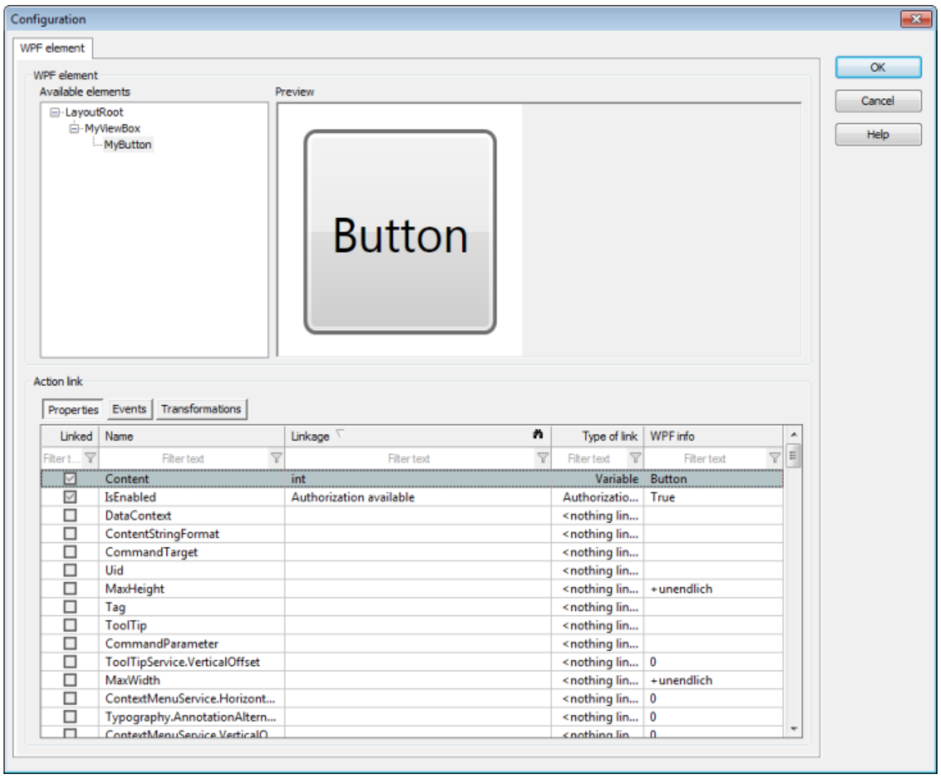
If referenced assemblies are used, note the instructions in the Referenced assemblies (on page 8) chapter.

6.3 Configuration of the linking

To configure a WPF element

1. In properties, select **WPF links** the property **Configuration** in the group.
2. The dialog with three tabs opens with a preview of the XAML file and the elements present in the file

DIALOG CONFIGURATION



Linked	Name	Linkage	Type of link	WPF info
Filter t...	Filter text	Filter text	Filter text	Filter text
✓	Content	int	Variable	Button
✓	IsEnabled	Authorization available	Authorizatio...	True
☐	DataContext		<nothing lin...	
☐	ContentStringFormat		<nothing lin...	
☐	CommandTarget		<nothing lin...	
☐	Uid		<nothing lin...	
☐	MaxHeight		<nothing lin...	+unendlich
☐	Tag		<nothing lin...	
☐	ToolTip		<nothing lin...	
☐	CommandParameter		<nothing lin...	
☐	ToolTipService.VerticalOffset		<nothing lin...	0
☐	MaxWidth		<nothing lin...	+unendlich
☐	ContextMenuService.Horizont...		<nothing lin...	0
☐	Typography.AnnotationAltern...		<nothing lin...	0
☐	ContextMenuService.VerticalO		<nothing lin	0

Parameter	Description
Available elements	Shows the named file elements in a tree structure. The selected element can be linked with process data. **WPF** is assigned to process data based on the element name. Therefore elements are only shown if they and the attendant elements have a name. Allocations are configured and shown in the **Properties**, **Events**, **Transformations** tabs. Hint: If the corresponding elements are not displayed, check in the XAML file to see if this has a name (for example: `<Grid  Name="GridName">`).
Preview	The selected element is shown flashing in the preview.
Properties (on page 53)	Configuration and display of properties (variables, authorizations, interlockings, linked values).
Events (on page 59)	Configuration and display of events (functions).
Transformations (on page 61)	Configuration and display of transformations.
Name	Name of the property.
Connection	Selection of link.
Link type	Type of link (variable, authorization, function)
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	Shows if a property is currently being used. Not contained by default in the view, but can be selected using *Context menu->Column selection*.

Information

Only logical objects can be displayed in the configuration dialog. Visual objects are not displayed. You can read about backgrounds and how visual objects can be animated in the Allocation of zenon object to WPF content.

EDIT HYPERLINKS

All configured hyperlinks can be edited from the properties of the element. Click on the element and open the property group **WPF links**. Hyperlinks can be further configured here, without having to open the dialog.

Limitations:

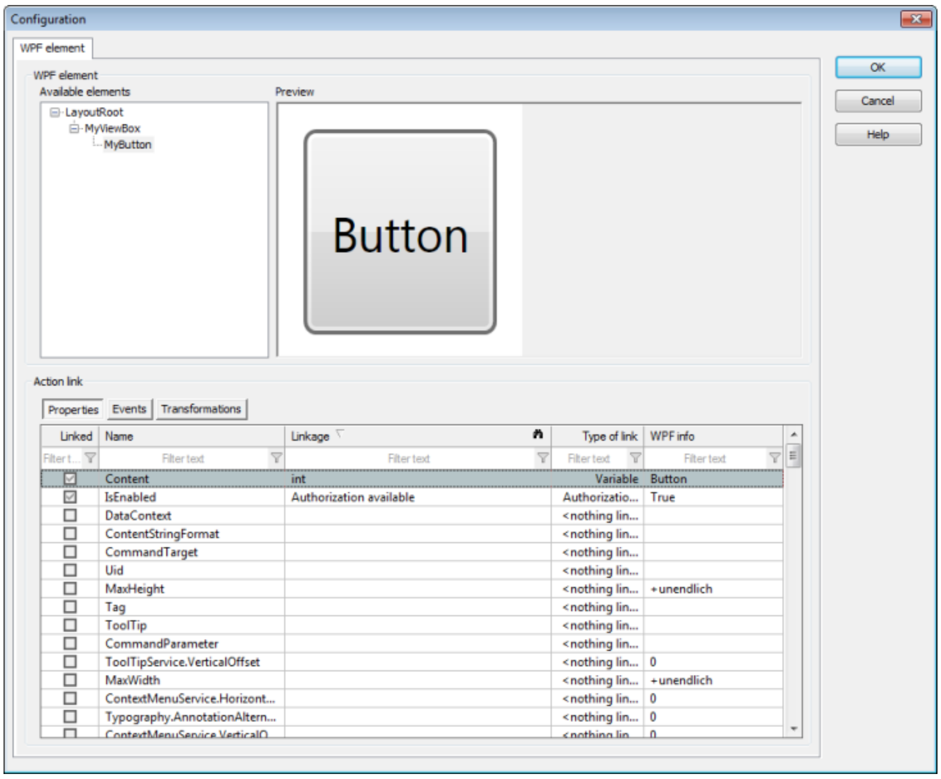
- The linking type cannot be changed here.

- New linkings can only be created via the configuration dialog.
- Insertion of a WPF elements into a symbol: WPF linkings cannot be exported.

6.3.1 Properties

The properties enable the linking of:

- Variables (on page 55)
- Values (on page 56)
- Authorizations and interlockings (on page 58)



Linked	Name	Linkage	Type of link	WPF info
☑	Content	int	Variable	Button
☑	IsEnabled	Authorization available	Authorizatio...	True
☐	DataContext		<nothing lin...	
☐	ContentStringFormat		<nothing lin...	
☐	CommandTarget		<nothing lin...	
☐	Uid		<nothing lin...	
☐	MaxHeight		<nothing lin...	+unendlich
☐	Tag		<nothing lin...	
☐	ToolTip		<nothing lin...	
☐	CommandParameter		<nothing lin...	
☐	ToolTipService.VerticalOffset		<nothing lin...	0
☐	MaxWidth		<nothing lin...	+unendlich
☐	ContextMenuService.Horizont...		<nothing lin...	0
☐	Typography.AnnotationAltern...		<nothing lin...	0
☐	ContextMenuService.VerticalO		<nothing lin	0

Parameter	Description
Name	Name of the property.
Linkage	Linked variable, authorization or linked value. Clicking in the column opens the respective selection dialog, depending on the entry in the **Link type** column.
Type of link	Selection of linking.
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	Shows if a property is currently being used. Not contained by default in the view, but can be selected using *Context menu->Column selection*.

CREATE LINK

To create a link:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select the desired link from the drop-down list.

   The following are available:

   - `<not linked>` (deletes an existing link)
   - `Authorization/Interlocking`
   - `Constant value`
   - `Variable`
4. Click in the **Link** cell
5. The dialog for configuring the desired link opens

Information

Properties of WPF and zenon can be different. If, for example the ***visibility*** *property is linked, there are three values available in .NET:*

- 0 - visible
- 1 - invisible
- 2- collapsed

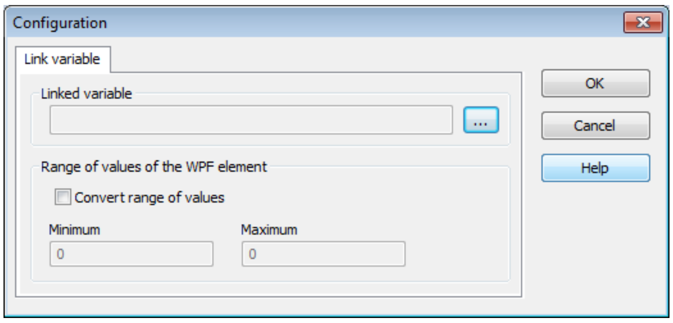
These values must be displayed via the linked zenon variable.

Link variable

To link a variable with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select from the **variable** drop down list
4. Click in the **Link** cell
5. The dialog for configuring the variables opens

This dialog also applies for the selection of variables with transformations (on page 61). The configuration also makes it possible to convert from zenon into WPF units.


Configuration

Link variable

Linked variable

Range of values of the WPF element

Convert range of values

Minimum: 0

Maximum: 0

Parameters	Description
Linked variables	Selection of the variable to be linked. A click on the **...** button opens the selection dialog.
Value range of WPF element	Data to convert variable values into WPF values.
Convert value range	`Active`: WPF unit conversion is switched on. **Effect on Runtime:** The current zenon value (incl. zenon unit) is converted to the WPF range using standardized minimum and maximum values. **For example:** The value of a variable varies from 100 to 200. With the variables, the standardized range is set to 100 - 200. The aim is to display this change in value using a WPF rotary knob. For this: - for **Transformations**, the **RotateTransform.Angle** property is linked to the variables - **Adjust value** activated - a WPF value range of `0` to `360` is configured Now the rotary knob can be turned at a value of 150, for example, by 180 degrees.
Minimum	Defines the lowest WPF value.
Maximum	Defines the highest WPF value.
OK	Accepts settings and ends the dialog.
Cancel	Discards settings and ends the dialog.
Help	Opens online help.

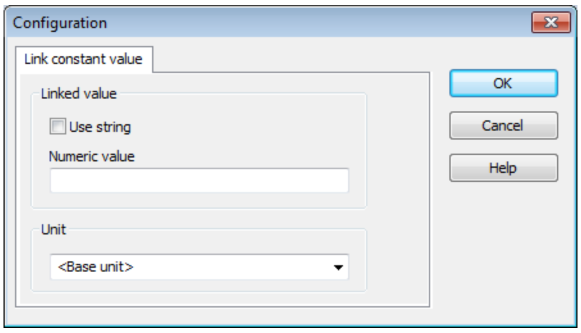
Link values

Linked values can either be a **String** or a numerical value of the type **Double**. When selecting the screen, the selected value is sent in WPF content after loading the WPF content.

To link a value with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select **Value linkings** from the drop-down list
4. Click in the **Link** cell

5. The dialog for configuration of value linking opens



Parameter	Description
Linked value:	Entry of a numerical value or string value.
Use string	`Active`: A string value is used instead of a numerical value. The language of string values can be switched. The text is translated in Runtime when the screen is called up and sent in WPF content. If the language is switched whilst the screen is opened, the string value is retranslated and sent.
String value/numerical value	Depending on what is selected for the **Use string** property, a numerical value or a string value is entered into this field. For numerical values, a unit of measurement can also be selected.
Unit:	Selection of a unit of measurement from the drop down list. You must have configured this in unit switching beforehand. The unit of measurement is allocated with the numerical value. If the units are switched in Runtime, the value is converted to the new unit of measurement and sent to WPF content.

CLOSE DIALOG

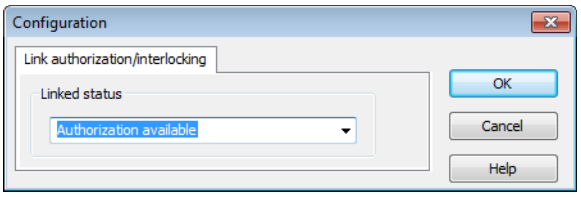
Options	Description
OK	Applies settings and closes the dialog.
Cancel	Discards all changes and closes the dialog.
Help	Opens online help.

Link authorization or interlocking

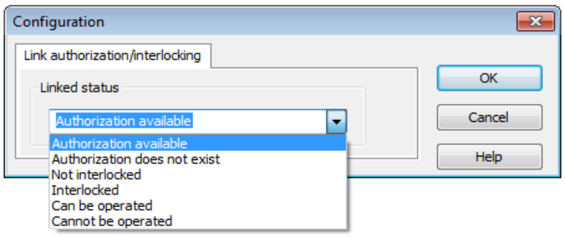
Authorizations cannot be granted for the whole WPF element. The element is allocated a user level. Authorizations are granted within the user level for individual controls. If an authorization is active, the value `1` is written to the element.

To link an authorization or interlocking with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select **Authorization/interlocking** from the drop down menu
4. Click in the **Link** cell
5. The dialog for configuring the authorizations opens



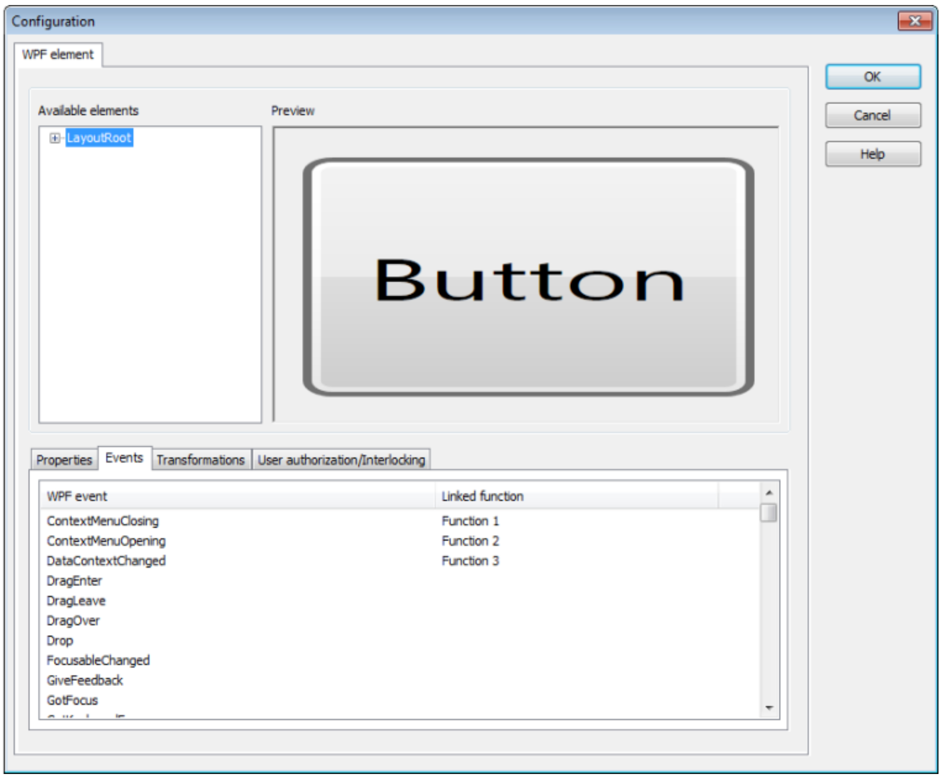
Parameters	Description
Link authorization/interlocking	Setting the authorizations.
Linked status	Selection of an authorization that is linked to a WPF control from the drop down list. For example, visibility and operability of a WPF button can depend on a user's status.



Authorization	Description
`Authorization available`	If the user has sufficient rights to operate the **WPF element**, a value of `1` is written to the property.
`Authorization does not exist`	If the user does not have sufficient rights to operate the **WPF element**, a value of `1` is written to the property.
`Not interlocked`	If the element is not locked, the value `1` is written to the property.
`Interlocked`	If the element is locked, the value `1` is written to the property.
`Can be operated`	If authorization is present and the element is not locked, then a value of `1` is written to the property.
`Cannot be operated`	If authorization is not present or the element is not locked, then a value of `1` is written to the property.

6.3.2 Events

Events make it possible to link zenon functions to a WPF element.



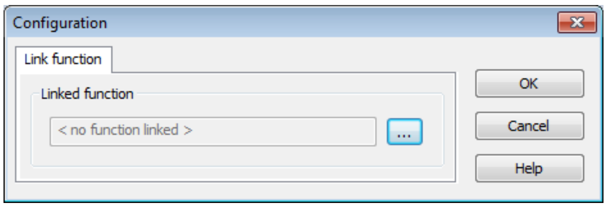
WPF event	Linked function
ContextMenuClosing	Function 1
ContextMenuOpening	Function 2
DataContextChanged	Function 3
DragEnter	
DragLeave	
DragOver	
Drop	
FocusableChanged	
GiveFeedback	
GotFocus	

Parameters	Description
Name	Name of the property.
Connection	Linked function. Clicking in the cell opens the configuration dialog**.**
Link type	Selection of linking. Clicking in the cell opens the selection dialog.
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	Shows if a property is currently being used. Not contained by default in the view, but can be selected using *Context menu->Column selection*.

LINK FUNCTIONS

To create a link:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select from the drop down list function
4. Click in the **Link** cell
5. The dialog for configuring the function opens

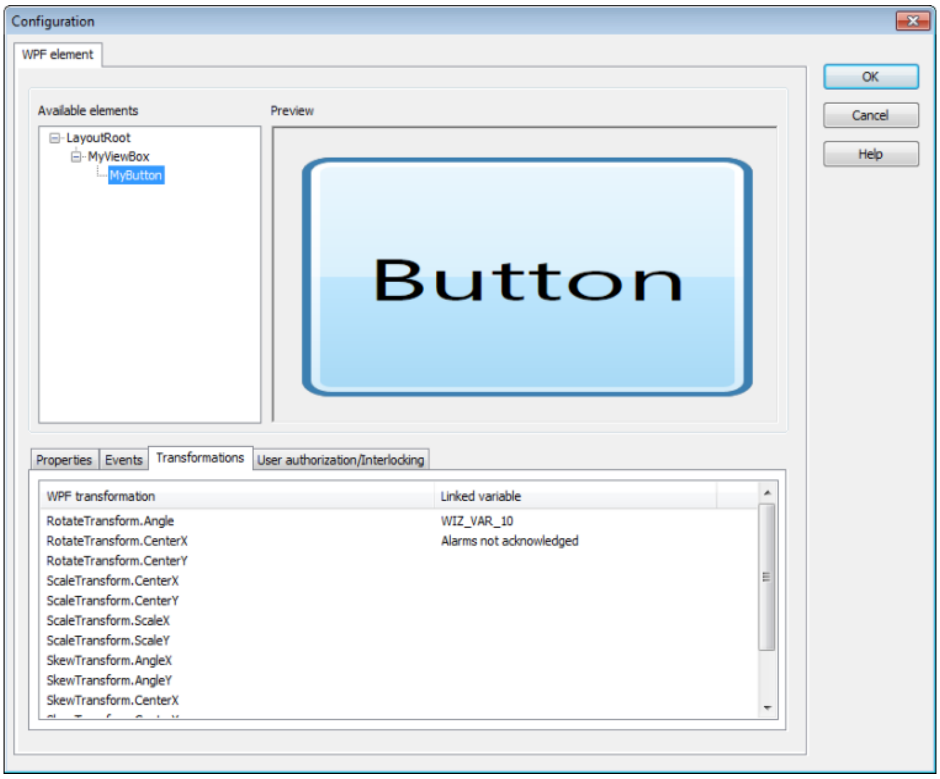


Parameters	Description
Linked function	Selection of the function to be linked. Clicking on the ... button opens the dialog for Function selection.
OK	Accepts selection and closes dialog.
Cancel	Discards changes and closes dialog.
Help	Opens online help.

6.3.3 Transformation

The **WPF element** does not support rotation. If, for example, the **WPF element** is in a symbol and the symbol is rotated, the **WPF element** does not rotate with it. Therefore there is a different mechanism for **Transformation** with WPF to turn elements or to otherwise transform them. These transformations are configured in the **Transformation** tab.

Attention: If the content is outside of the **WPF element** area, this part of the contents is lost or it is not shown.



WPF transformation	Linked variable
RotateTransform.Angle	WIZ_VAR_10
RotateTransform.CenterX	Alarms not acknowledged
RotateTransform.CenterY	
ScaleTransform.CenterX	
ScaleTransform.CenterY	
ScaleTransform.ScaleX	
ScaleTransform.ScaleY	
SkewTransform.AngleX	
SkewTransform.AngleY	
SkewTransform.CenterX	

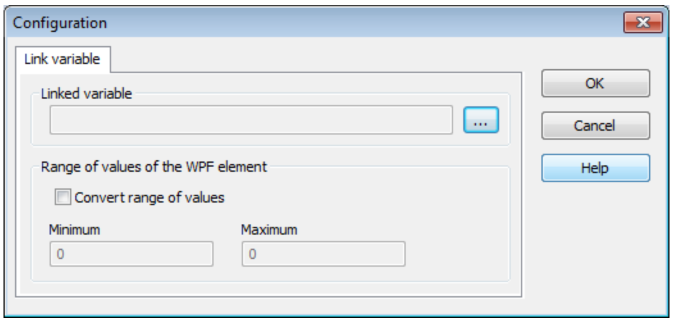
Parameters	Description
Name	Name of the property.
Connection	Selection of the linked variables. Transformations are displayed in XAML as transformation objects with their own properties. If an element supports a transformation, then the possible properties of the transformation object are displayed in list view. (more on this in: Integrate button as WPF XAML in zenon (on page 102) For example, if the linked variable is set at the value of `10`, then this value is written as a WPF target and the WPF element is rotated by 10°.
Link type	Selection of transformation link type.
WPF info	Shows the current value for properties in WPF content. For the user, it is directly visible what type of property it is (Boolean, string, etc.).
Linked	Shows if a property is currently being used. Not contained by default in the view, but can be selected using *Context menu->Column selection*.

LINK TRANSFORMATIONS

To link a transformation with a WPF property:

1. Highlight the line with the property that is to be linked
2. Click in the **Link type cell**
3. Select from the **Transformation** drop down list
4. Click in the **Link** cell
5. The dialog for configuring the variables opens

The configuration also makes it possible to convert from zenon into WPF units.



Parameters	Description
Linked variables	Selection of the variable to be linked. A click on the **...** button opens the selection dialog.
Value range of WPF element	Data to convert variable values into WPF values.
Convert value range	`Active`: WPF unit conversion is switched on. **Effect on Runtime:** The current zenon value (incl. zenon unit) is converted to the WPF range using standardized minimum and maximum values. **For example:** The value of a variable varies from 100 to 200. With the variables, the standardized range is set to 100 - 200. The aim is to display this change in value using a WPF rotary knob. For this: ▸ for **Transformations**, the **RotateTransform.Angle** property is linked to the variables ▸ **Adjust value** activated ▸ a WPF value range of `0` to `360` is configured Now the rotary knob can be turned at a value of 150, for example, by 180 degrees.
Minimum	Defines the lowest WPF value.
Maximum	Defines the highest WPF value.
OK	Accepts settings and ends the dialog.
Cancel	Discards settings and ends the dialog.
Help	Opens online help.

6.4 Validity of XAML Files

XAML files are valid subject to certain requirements:

- Correct name spaces
- No class references
- Scalability

CORRECT NAME SPACE

The **WPF element** can only display WPF content, i.e.:

Only XAML files with the correct WPF namespace can be displayed by the **WPF element**. Files that use a Silverlight namespace cannot be loaded or displayed. However, in most cases it is suffice to change the Silverlight namespace to the WPF namespace.

WPF-Namespaces:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

NO USE OF CLASS REFERENCES

Because the XAML files can be loaded dynamically, it is not possible to use XAML files that contain references to classes ("class" key in header). Functions that have been programmed in independently-created C#- files cannot be used.

In order to use WPF user controls with code behind, the process as described in the Creating a simple WPF user control with code behind funciton (on page 24) must be carried out.

SCALABILITY

If the content of a **WPF element** is adjusted to the size of the **WPF element**, then the controls of the **WPF element** are interlaced in a control that offers this functionality, such as a **view box** for example. In addition, it must be ensured that the **height** and **width** for this elements are configured as **automatic**.

CHECKING AN XAML FILE TO SEE IF IT IS CORRECT

To check if an XAML file has the correct format:

- Open XAML file in Internet Explorer
  - If it can be opened without additional plug-ins (Java or similar), then it can be assumed with a high degree of certainty that this file can be loaded or displayed by zenon
  - if problems occur during loading, these are then shown in Internet Explorer and the lines in which problems arise can be clearly seen

The scaling can also be tested in this manner: If the file has been created correctly, the content will adjust to the size of the Internet Explorer window.

ERROR MESSAGE

If an invalid file is used in zenon, then an error message is displayed in the output window when loading the file in the WPF element.

For example:

"error when loading
```
xaml-Datei:C:\ProgramData\COPA-DATA\SQL\781b1352-59d0-437e-a173-08563c3142e9\
FILES\zenon\custom\media\UserControl1.xaml

The attribute "Class" cannot be found in XML namespace
"http://schemas.microsoft.com/winfx/2006/xaml". Line 7 Position 2."
```

6.5 Pre-built elements

zenon is already shipped with several WPF elements. More are available for download in the web shop.

All WPF elements have properties which determine the graphical design of the respective element (Dependency Properties). Setting the values via an XAML file or linking the property via zenon can directly change the look in the Runtime. The tables in the description of the individual elements contain the respective Dependency Properties, depending on the control.

Available elements:

- Analog clock (on page 66)
- Vertical bar graph (on page 67)
- Comtrade Viewer (on page 68)
- Energy class diagram (on page 79)
- Progress bar (on page 68)
- Pareto diagram (on page 80)
- Sankey Diagram (on page 87)
- Round display (on page 84)
- Temperature control (on page 89)
- Universal slider (on page 90)
- Waterfall diagram (on page 91)

REPLACING ASSEMBLY WITH A NEWER VERSION

Per project only one `Assembly` for a WPF element can be used in the zenon Editor as well as in the Runtime. If two versions of an `Assembly` are available in a project, then the first loaded file is used. A user enquiry is made as to which version should be used. No further actions are needed for the maintenance of the versions used up until now. If a newer version is chosen, all corresponding CDWPF files in all symbols and images in all projects must be adapted.

Note for Multi-Project Administration: If an `Assembly` in a project is replaced by a new version, it must also be replaced in all other projects that are loaded in the Editor or in Runtime.

6.5.1 Analog clock - AnalogClockControl

Property	Function	Value
ElementStyle	Shape/type of element.	`Enum:` - `SmallNumbers` - `BigNumbers` - `No`
ElementBackgroundBrush	Color of element background.	`Brush`
ElementGlasReflection	Activate the glass effect on the element.	`Visibility`
Offset	Value in hours (h) which displays the time lag to the system clock.	`Int16`
OriginText	Text which is displayed in the clock (e.g. location).	`String`

6.5.2 Bar graph vertical - VerticalBargraphControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	`Double`
MinValue	Minimum value of the scale.	`Double`
MaxValue	Maximum value of the scale.	`Double`
MajorTicksCount	Number of main ticks on the scale.	`Integer`
MinorTicksCount	Number of sub ticks on the scale.	`Integer`
MajorTickColor	Color of main ticks on the scale.	`Color`
MinorTickColor	Color of sub ticks on the scale.	`Color`
ElementBorderBrush	Color of the element border.	`Brush`
ElementBackgroundBrush	Color of element background.	`Brush`
ElementGlasReflection	Activate the glass effect on the element.	`Visibility`
ElementFontFamily	Element font.	`Font`
ScaleFontSize	Font size of the scale.	`Double`
ScaleFontColor	Font color of the scale.	`Color`
IndicatorBrush	Bar graph fill color.	`Brush`
BargraphSeparation	Number of bar graph dividion.	`Integer`
BargraphSeparationColor	Color of the scale division.	`Color`

6.5.3 Progress bar - ProgressBarControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	`Double`
MinValue	Minimum value of the value area.	`Double`
MaxValue	Maximum value of the value area.	`Double`
ProgressbarDivisionCount	Number of divisions of the progress bar.	`Integer`
VisibilityText	Visibility of the value display.	`Boolean`
TextSize	Font size of the value display.	`Double`
TextColor	Color of the value display.	`Color`
ProgressBarBoxedColor	Color of the border of the progress bar.	`Color`
ProgressBarMarginDistance	Distance of the progress bar box from the element edge (left, top, right, down).	`Double`
ProgressBarInactiveBrush	Indicator color not active.	`Brush`
ProgressBarActiveBrush	Indicator color active.	`Brush`
ProgressBarPadding	Distance of the progress bar from the progress bar box (left, top, right, down).	`Double`
ElementBorderBrush	Color of the element border.	`Brush`
ElementBackgroundBrush	Color of element background.	`Brush`

6.5.4 COMTRADE-Viewer

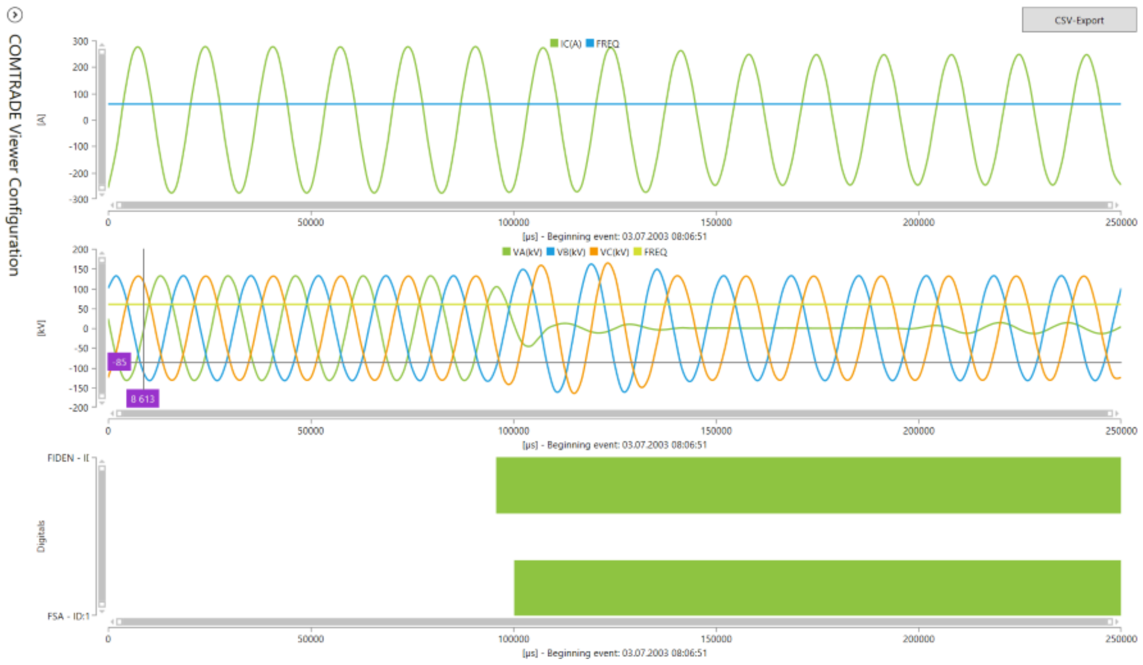
The **COMTRADE-Viewer** WPF element is available to partners of COPA-DATA and is available to them via the COPA-DATA Partner Community (https://www.copadata.com/en-us/partner-community/).

It is for the graphical analysis of digital error and result logging of a COMTRADE file.

Information

The control supports IEEE C37.111 (IEEE Standard Common Format for Transient Data Exchange (COMTRADE) for Power Systems) standards-compliant files. ASCII or binary files in accordance with the 1999 or 2013 edition can be visualized.

Older files or files without a year identification are not supported. A warning dialog is called up when an invalid/unsupported file is selected.


COMTRADE Viewer Configuration
CSV-Export
IC(A) FREQ
VA(kV) VB(kV) VC(kV) FREQ
[µs] - Beginning event: 03.07.2003 08:06:51
Digitals

Possibilities of the **COMTRADE-Viewer** WPF control in zenon Runtime:

- Selection of a file in the COMTRADE file format
- Visualization of the selected COMTRADE file:
  Note: The display colors can be configured in the zenon Editor.
  - Current (sinus wave display)
  - Voltage (sinus wave display)
  - Digital signals (binary bar chart display)
  - Display of values at a selected cursor position.
  - If an element that represents neither current or voltage is selected, (such as frequency), this is visualized in both analog areas again (current and voltage).
- Navigation:
  - Zoom in and zoom out using the mouse wheel, scroll bar and Multi-Touch gestures
  - Enlargement of the area
    Selection of the area by clicking the mouse
  - Move the display area using the right mouse button, scroll bar or Multi-Touch gestures.
- Exports selected objects as an CSV file.

Hint

To be able to transport COMTRADE files to the zenon Runtime computer, you can also use the file transfer of the **IEC 61850 driver** or the **FTP function block** of zenon Logic.

You can find further information about this in the driver documentation of the IEC 61850 driver or in the zenon Logic documentation.

Display during Runtime

The **COMTRADE** WPF element offers two views in Runtime:

- Configuration view
  - Selection of a COMTRADE configuration file
  - Selection of the elements to be displayed
- Graph view
  - Zoom in and zoom out
  - Display of values at a selected cursor position.
  - Export of the selected elements as an CSV file

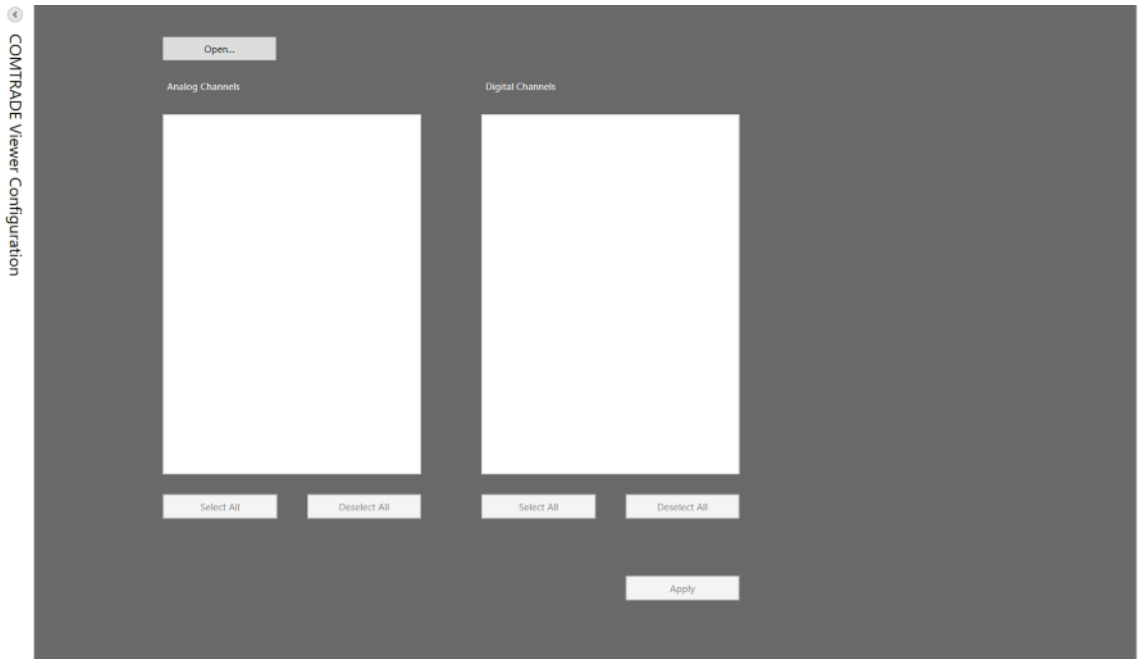
Information

The switch between the views is integrated in the WPF element. Additional project configuration of a screen switching function is not necessary.

Runtime view - configuration page

If a screen with a configured **COMTRADE-Viewer** WPF element is called up, the display of the respective configuration page is empty.

Note: This also applies if, in zenon Runtime, there is a switch from one screen to another screen with the screen switching function.



COMTRADE VIEWER CONFIGURATION

The **COMTRADE Viewer Configuration** switching, arranged vertically on the side, switches the display of the configuration to graphic view and vice versa.

SELECT FILE

The **Open...** button opens the file selection dialog to select a file.

There is a pre-selection for display in the file selection:

- In doing so, file pairs of `*.cfg`– and `*.dat` files are detected.
  Note: Optional `*.hdr` or `*.inf` files are not taken into account.
- Only the corresponding `*.dat` files are displayed.
- All attendant files (`*.dat, *.cfg`) are loaded by clicking on the desired file and the **OK** button.
- One file can be loaded.
- After loading the file, the contents of the file are shown in the **Analog Channels** and **Digital Channels** columns.
  The labels and units of the elements originate from the **COMTRADE** configuration and cannot be changed.

FURTHER INFORMATION ON THE EDITING OF _*.CFG- AND *.DAT FILES

The information from the *.cfg file allows the evaluation of the *.dat file. It contains the data from various analog and digital series of measurements of currents and voltages. The data is broken down into individual data sets and shown in hex format.

***.cfg files**

- The last entry of a file of this data type is a time multiplier. This entry is multiplied by the time stamp of one of each entry from the *.dat file when a disturbance (error message) is read in. If there is no time multiplier, a factor with the value of 1 is assumed internally. The *.cfg file is not changed in the process.
- Certain standards apply for the entries of the digital measured values. Example of a standard-compliant entry of a digital measured value: 1,LOPHC,,,0. However, if there is no zero at the end of the entry, the **COMTRADE-Viewer** adds this internally. The *.cfg file is not changed in the process.

***.dat files**

- The **COMTRADE-Viewer** is in a position to read in files of this data type that start with the index 0 or >1. In doing so, a check is constantly carried out to see whether these data sets are numbered continually in discrete steps from 1. If there are data sets that are not correctly numbered, the file cannot be read in.

ANALOG CHANNELS

Parameter	Description
`[Liste der verfügbaren Kanäle]`	Selection of the elements to be visualized. Multiple selection by clicking on the desired entry in the list. Selected elements are shown with a colored background. Another mouse click undoes the selection of the entry.
Select All	Selects all elements from the list.
Deselect All	Deactivates the existing selection of elements.

DIGITAL CHANNELS

Parameter	Description
`[Liste der verfügbaren Kanäle]`	Selection of the elements to be visualized Multiple selection by clicking on the desired entry in the list. Selected elements are shown with a colored background. Another mouse click undoes the selection of the entry.
Select All	Selects all elements from the list.
Deselect All	Deactivates the existing selection of elements.

SHOW SELECTION

To show your selection in the graphic view, click on the **Apply** button.

Note: Clicking on the vertically-arranged **COMTRADE Viewer Configuration** switching only changes the view. An amended selection of the channels is not taken into account in the process.

Runtime view - visualization of COMTRADE data

The selected channels are visualized in the graph view of the **COMTRADE-Viewer** WPF element. The coloring can be configured in the zenon Editor.

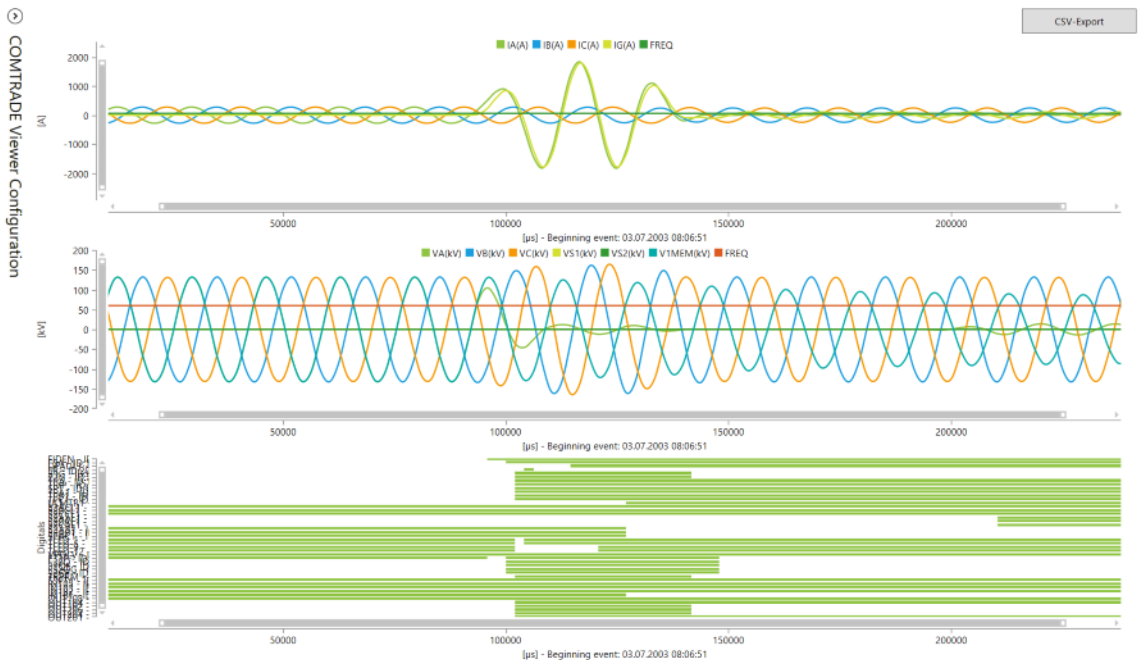
EXPORT OF THE SELECTED DATA

The selected analog and digital channels can be exported to a CSV file with the **CSV-Export** button.

GRAPH VIEW

The graph view of the **COMTRADE-Viewers** is divided into three sections:

- Current amperage
  Upper area
- Voltage
  Mid area
- Digital channels
  Lower area


COMTRADE Viewer Configuration
CSV-Export
IA(A) IB(A) IC(A) IG(A) FREQ
[A]
[kV]
VA(kV) VB(kV) VC(kV) VS1(kV) VS2(kV) V1MEM(kV) FREQ
Digitals
[µs] - Beginning event: 03.07.2003 08:06:51
50000
100000
150000
200000

AXIS LABELING

- Horizontal axis
  The horizontal axis represents the complete time period as illustrated in the COMTRADE file (`*.dat`).
  The scaling of this time axis depends on the enlargement level. The higher the enlargement selected, the more detailed the time display.
- Vertical axis
  The vertical axis represents the values.
  - The scaling of the value axis depends on the enlargement level. The greater the enlargement selected, the more detailed the display of values.
  - The labeling of the analog channels is shown vertically next to the values and corresponds to the measuring unit as defined in the COMTRADE file (`*.cfg`).
  - The digital channels are displayed in the sequence as defined in the COMTRADE file (`*.cfg`).
    The `Channel identifier` of the COMTRADE file serves as an identifier.

KEY

🟩 IA(A) 🟦 IB(A) 🟧 IC(A) 🟨 IG(A)

The color key of the graphs is shown at the head of the graph.

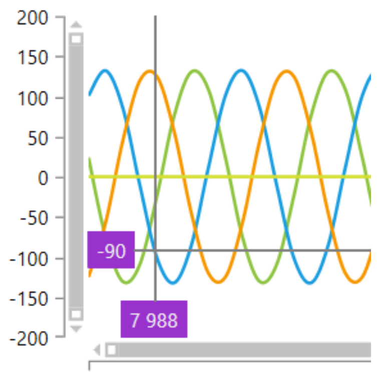
- The labeling of the digital channels corresponds to the channel description as defined in the COMTRADE file (`*.cfg`).
- The colors for each channel are assigned automatically with the configured color palette.
- The time is displayed in a footer under the graph. The start time is displayed as a text.

NAVIGATION AND ZOOM

Navigation (scroll and zoom) is always applied to all three areas of the graphic display.

- You can move the display within the horizontal time line with the scroll bar.
- Zoom in and zoom out
  - You can zoom at the current position of the mouse pointer in the graphics view or reduce the enlargement.
  - The selected area is displayed by selecting a display area with the mouse button held down.
    Note: The display of the values is always amended to the selected area. As a result, this can lead to a flattening of the curve in the enlarged graphic view.
  - Double clicking on the scroll bar resets the enlargement.

ANALYSIS



The precise values at the position of the mouse pointer are visualized with a display in value blocks. A crosshair offers additional visual support with the exact determination of the reading position.

Configurable control properties - color display

ENGINEERING IN THE EDITOR

The element with the name **COMTRADE.CDWPF** can be configured and placed in each zenon screen type.
The project configuration of **Width [pixels]** and **Height [pixels]** of the element depend on the proportions. This prevents the **COMTRADE-Viewer** being displayed as distorted in Runtime.

Note: When configuring the project, ensure that there is sufficient size to guarantee a clear overview.

GRAPHICAL AMENDMENTS

You configure the graphic design in the properties of the WPF element.
You can find further information in the configuration of the linking (on page 50) chapter in this manual.

Possible color values:

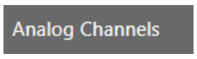
- Hexadecimal color values
  `#RRGGBB`
  Example color values: #000000 = black , #FFFFFF = white, #FF0000 = red
- Color values by name
  Reference: https://msdn.microsoft.com/en-us/library/system.drawing.color.aspx (https://msdn.microsoft.com/en-us/library/system.drawing.color.aspx)

Hint

The properties for the **COMTRADE-Viewer** WPF element have a "`z`" as a starting color. Use name filtering for a clear display when configuring the linking.

CONFIGURATION PAGE

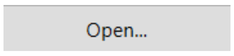
Text and background color of the configuration page.



Parameters	Description	Value
zConfiguratinPageTextColor	Text color of the configuration page	`String`
zConfigurationPageBackgroundColor	Background color of the configuration page	`String`

BUTTONS

Text and background color of the button.



Parameters	Description	Value
zButtonTextColor	Text color of the button	`String`
zButtonBackgroundColor	Background color of the button	`String`

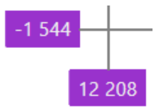
CHART

Text color of the axis labeling or key and background color.

Parameters	Description	Value
zChartTextColor	Text color of the axis labeling.	`String`
zChartBackgroundColor	Background color of the axis labeling	`String`

LABEL

Text and background color of the display of a selected cursor position.



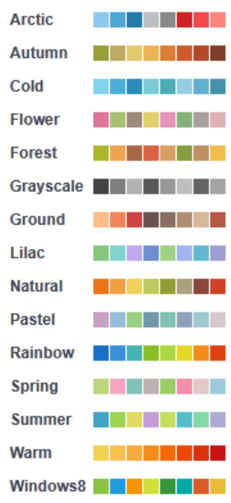
Parameters	Description	Value
zChartLabelTextColor	Text color of the value display	`String`
zChartLabelBackgroundColor	Background color of the value display	`String`

CHART

Color palette of the graph view and the attendant keys.

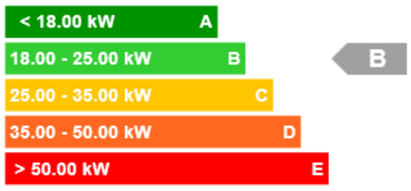
Parameters	Description	Value
zChartPalette	Color palette of the colors for graphs and keys. Referencing with color palette name (see overview). Default: if no color palette is configured, the color palette of the computer's operating system is used.	`String`

POSSIBLE COLOR PALETTES - OVERVIEW



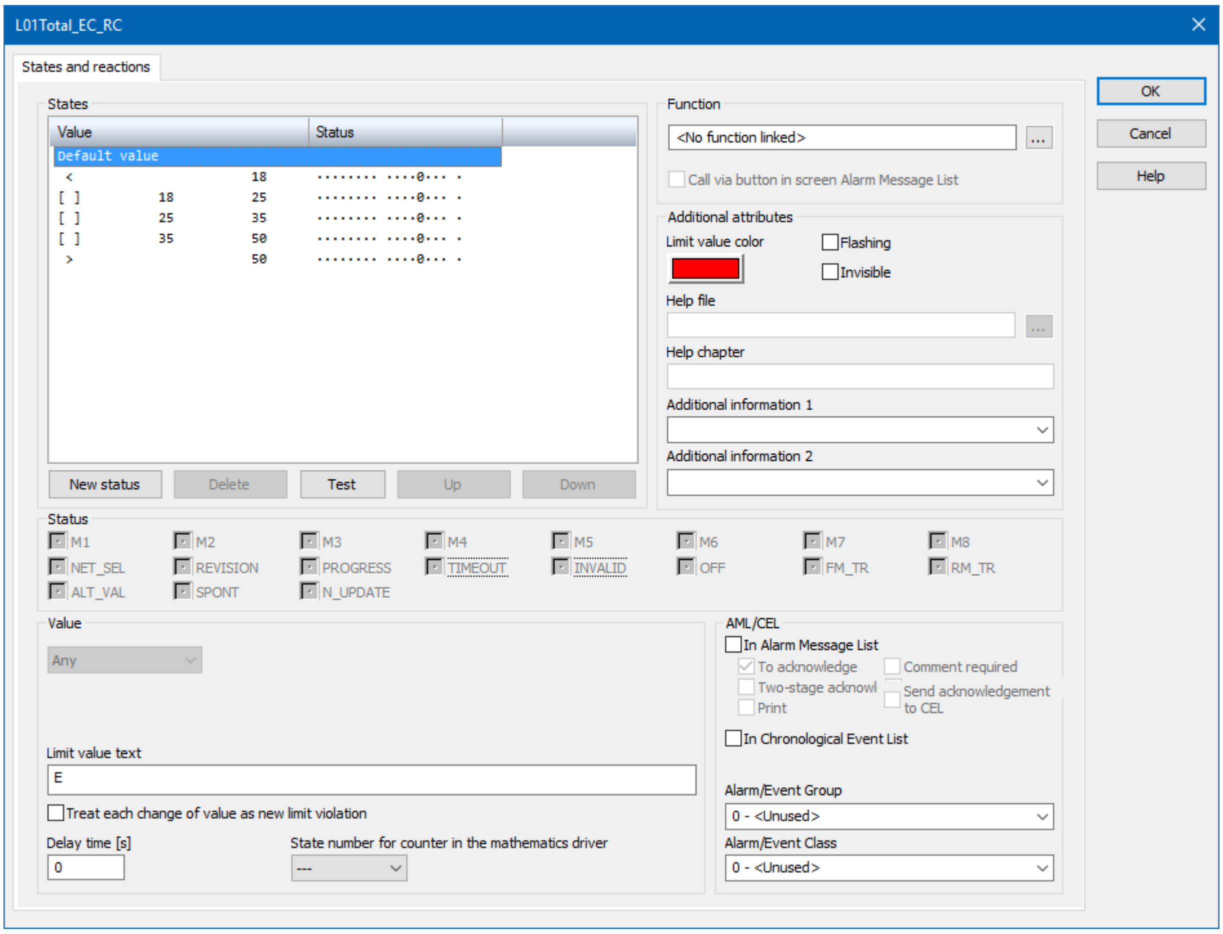
6.5.5 Energy class diagram

The energy class diagram, WPF element is available to partners of COPA-DATA and is available to them via the COPA-DATA Partner Community (https://www.copadata.com/en-us/partner-community/).



A reaction matrix must be used to model an energy class diagram. This reaction matrix must be linked to the variable whose value is envisaged for display and distribution in energy classes. The name of the variable must be transferred to the "**zVariableName**" property.

REACTION MATRIX FOR ENERGY CLASS DIAGRAM


L01Total_EC_RC

States and reactions

States

```
Value                       Status
Default value
 <                   18     ········ ····0··· ·
[ ]         18       25     ········ ····0··· ·
[ ]         25       35     ········ ····0··· ·
[ ]         35       50     ········ ····0··· ·
 >                   50     ········ ····0··· ·
```

Function: <No function linked>

Call via button in screen Alarm Message List

Additional attributes

Limit value color: 🟥

Flashing

Invisible

Help file:

Help chapter:

Additional information 1:

Additional information 2:

Status: M1, M2, M3, M4, M5, M6, M7, M8, NET_SEL, REVISION, PROGRESS, TIMEOUT, INVALID, OFF, FM_TR, RM_TR, ALT_VAL, SPONT, N_UPDATE

Value: Any

Limit value text: E

Treat each change of value as new limit violation

Delay time [s]: 0

State number for counter in the mathematics driver: ---

AML/CEL

In Alarm Message List

✓ To acknowledge

Comment required

Two-stage acknowl

Send acknowledgement to CEL

Print

In Chronological Event List

Alarm/Event Group: 0 - <Unused>

Alarm/Event Class: 0 - <Unused>

The linked reaction matrix must correspond to the following schematic:

- The first status must be an area, or a "less than" definition

- Then as many different areas as desired can be defined.
- The last status must be an area or a "greater than" definition.

The following is applicable for project configuration:

1. If the first status is an area and the value of the variable comes under this area, the first status in the diagram is shown nevertheless. The same is applicable for the last status the other way round.
2. The colors that the WPF diagram uses for the classes are the limit value colors that were defined in the reaction matrix.
3. The letters for the classes are set in alphabetical order starting with "A".

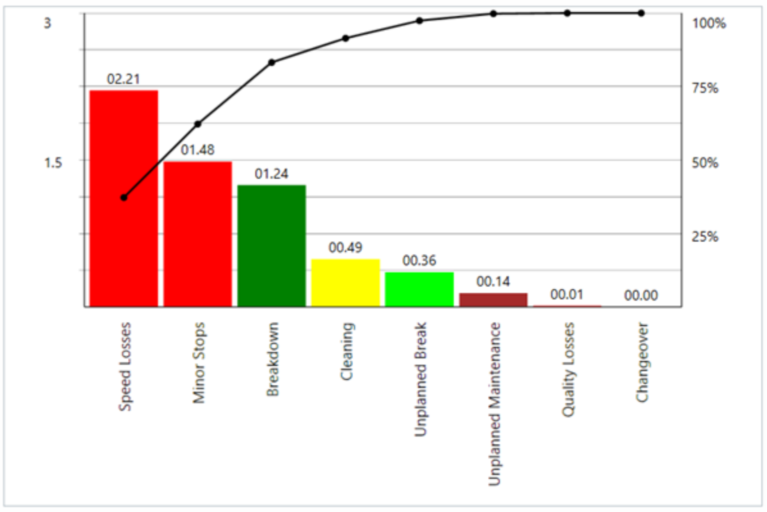
Property	Description	Value
zenonFontID	ID for a font from the first font list (font size is not taken into account)	Integer
zenonNumberOfDecimalPlaces	Number of displayed decimal points	Integer
zenonVariableName	Name of the variable to be displayed.	String

Note: Additional VSTA programming is necessary for the display of the energy class diagram in the zenon web client. You can find details on this in the display of WPF elements in the zenon web client (on page 93).

6.5.6 Pareto diagram

The Pareto diagram, WPF element is available to partners of COPA-DATA and is available to them via the COPA-DATA Partner Community (https://www.copadata.com/en-us/partner-community/).

An example of a Pareto diagram in Runtime is shown below:



The following settings can be made in the WPF configuration window under **COPADATA-ELEMENT**:

Property	Function	Value

zenonBarColor1	Color of the first Bar	`Color (String)`
zenonBarColor2	Color of the second Bar	`Color (String)`
zenonBarColor3	Color of the third Bar	`Color (String)`
zenonBarColor4	Color of the fourth Bar	`Color (String)`
zenonBarColor5	Color of element fifth Bar	`Color (String)`
zenonBarColor6	Color of element sixth Bar	`Color (String)`
zenonBarColor7	Color of element seventh Bar	`Color (String)`
zenonBarColor8	Color of element eighth Bar	`Color (String)`
zenonBarColor9	Color of element ninth Bar	`Color (String)`
zenonBarColor10	Color of element tenth Bar	`Color (String)`
zenonColorPercentageLine	Color of the percentage line (relative sum frequency).	`Color (String)`
zenonLineVisibility	Visibility of the percentage line (relative sum frequency).	`Boolean`
zenonVariable1_Label	Labeling for the 1st Bar	`String`
zenonVariable1_Value	Value of the 1st Bar	`Double`
zenonVariable2_Label	Labeling for the 2nd Bar	`String`
zenonVariable2_Value	Value of the 2nd Bar	`Double`
zenonVariable3_Label	Labeling for the 3rd Bar	`String`
zenonVariable3_Value	Value of the 3rd Bar	`Double`
zenonVariable4_Label	Labeling for the 4th Bar	`String`
zenonVariable4_Value	Value of the 4th Bar	`Double`
zenonVariable5_Label	Labeling for the 5th Bar	`String`
zenonVariable5_Value	Value of the 5th Bar	`Double`
zenonVariable6_Label	Labeling for the 6th Bar	`String`
zenonVariable6_Value	Value of the 6th Bar	`Double`
zenonVariable7_Label	Labeling for the 7th Bar	`String`
zenonVariable7_Value	Value of the 7th Bar	`Double`

zenonVariable8_Label	Labeling for the 8th Bar	`String`
zenonVariable8_Value	Value of the 8th Bar	`Double`
zenonVariable9_Label	Labeling for the 9th Bar	`String`
zenonVariable9_Value	Value of the 9th Bar	`Double`
zenonVariable10_Label	Labeling for the 10th Bar	`String`
zenonVariable10_Value	Value of the 10th Bar	`Double`

The following events can be used and linked to zenon functions:

Event	**Function**	**Value**
zenonBar1Click	Function that is executed when the 1st bar is clicked on.	Function
zenonBar2Click	Function that is executed when the 2nd bar is clicked on.	Function
zenonBar3Click	Function that is executed when the 3rd bar is clicked on.	Function
zenonBar4Click	Function that is executed when the 4th bar is clicked on.	Function
zenonBar5Click	Function that is executed when the 5th bar is clicked on.	Function
zenonBar6Click	Function that is executed when the 6th bar is clicked on.	Function
zenonBar7Click	Function that is executed when the 7th bar is clicked on.	Function
zenonBar8Click	Function that is executed when the 8th bar is clicked on.	Function
zenonBar9Click	Function that is executed when the 9th bar is clicked on.	Function
zenonBar10Click	Function that is executed when the 10th bar is clicked on.	Function

6.5.7 Circular gauge control

Property	Function	Value
CurrentValue	Current value which should be displayed.	`Double`
IsReversed	Scale orientation - clockwise or anti-clockwise.	`Boolean`
ElementFontFamily	Element font.	`Font`
MinValue	Minimum value of the scale.	`Double`
MaxValue	Maximum value of the scale.	`Double`
ScaleRadius	Radius of the scale.	`Double`
ScaleStartAngle	Angle at which the scale starts.	`Double`
ScaleLabelRotationMode	Alignment of the scale caption.	`Enum:` ▸ `None` ▸ `Automatic` ▸ `SurroundIn` ▸ `SurroundOut`
ScaleSweepAngle	Angel area which defines the size of the scale.	`Double`
ScaleLabelFontSize	Font size of the scale caption.	`Double`
ScaleLabelColor	Font color of the scale caption.	`Color`
ScaleLabelRadius	Radius on which the scale caption is orientated.	`Double`
ScaleValuePrecision	Accuracy of the scale caption.	`Integer`
PointerStyle	Shape of the pointer displaying the value.	`Enum:` ▸ `Arrow` ▸ `Rectangle` ▸ `TriangleCap` ▸ `Pentagon` ▸ `Triangle`
MajorTickColor	Color of main ticks on the scale.	`Color`
MinorTickColor	Color of sub ticks on the scale.	`Color`
MajorTickSize	Size of main ticks on the scale.	`Size`
MinorTickSize	Size of sub ticks on the scale.	`Size`
MajorTicksCount	Number of main ticks on the scale.	`Integer`
MajorTicksShape	Shape/type of main ticks on the scale.	`Enum:` ▸ `Rectangle`

		▸ `Trapezoid` ▸ `Triangle`

MinorTicksShape	Shape/type of sub ticks on the scale.	`Enum:` ▸ `Rectangle` ▸ `Trapezoid` ▸ `Triangle`
MinorTicksCount	Number of sub ticks on the scale.	`Integer`
PointerSize	Size of the pointer.	`Size`
PointerCapRadius	Size of the pointer fastening point.	`Double`
PointerBorderBrush	Color of pointer border.	`Brush`
PointerCapStyle	Shape/type of pointer fastening point.	`Enum:` ▸ `BackCap` ▸ `FrontCap` ▸ `Screw`
PointerCapBorderBrush	Color of pointer fastening point.	`Brush`
PointerBrush	Color of pointer.	`Brush`
GaugeBorderBrush	Color of the element border.	`Brush`
GaugeBackgroundBrush	Color of element background.	`Brush`
PointerCapColorBrush	Color of pointer fastening point.	`Brush`
GaugeMiddlePlate	Radius of the element background middle plate.	`Double`
PointerOffset	Offset of the pointer (displacement).	`Double`
RangeRadius	Radius of the total range display.	`Double`
RangeThickness	Thickness of the total range display.	`Double`
RangeStartValue	Start value of the total range display.	`Double`
Range1EndValue	End value of the 1st area and start value of the 2nd range.	`Double`
Range2EndValue	End value of the 2nd area and start value of the 3rd range.	`Double`
Range3EndValue	End value of the 3rd area and start value of the 4th range.	`Double`
Range4EndValue	End value of the 4th area and start value of the 5th range.	`Double`
Range5EndValue	End value of the 5th area and start value of the 6th range.	`Double`
Range6EndValue	End value of the 6th range.	`Double`
Range1ColorBrush	Color of the first range.	`Brush`
Range2ColorBrush	Color of the second range.	`Brush`
Range3ColorBrush	Color of the third range.	`Brush`
Range4ColorBrush	Color of the fourth range.	`Brush`
Range5ColorBrush	Color of element fifth range.	`Brush`
Range6ColorBrush	Color of element sixth range.	`Brush`

ScaleOuterBorderBrush	Color of the scale border.	`Brush`
ScaleBackgroundBrush	Color of scale background.	`Brush`
ValueTextFrameStyle	Shape/type of value display.	`Enum:` ▸ `LargeFrame` ▸ `SmallFrame` ▸ `None`
ValueTextContent	Content of the value display.	`Enum:` ▸ `Text` ▸ `TextValue` ▸ `Value`
ValueTextSize	Font size of the value display.	`Double`
ValueTextColor	Font size of the value display.	`Color`
IsGlasReflection	Activate the glass effect on the element.	`Boolean`
GaugeOffsett	Lowering the rotation point of the whole element.	`Double`

6.5.8 Sankey Diagram

The Sankey diagram, WPF element is available to partners of COPA-DATA and is available to them via the COPA-DATA Partner Community (https://www.copadata.com/en-us/partner-community/).

The Sankey wizard must be used to model a Sankey diagram. The wizard creates an XML file that is then evaluated by the WPF element. To do this, the **zSankeyName** property must be given the name of the XML file. The XML file must be in the `Other` folder of a project. This is saved there by the wizard.

An example of a Sankey diagram in Runtime is shown below:

The following settings can be made in the WPF configuration window under **COPADATA-ELEMENT**:

Property	Function	Value
FontSize	Font size of the texts.	`Integer`
zBackgroundColor	Background color of the diagram.	`Color (String)`
zFontColor	Color of the texts.	`Color (String)`
zFontFamily	Font of all texts.	`Font (String)`
zLossDetectionActive	Automatic loss detection activated/deactivated. If `true`, then losses are automatically shown at a node points as flows.	`Bool`
zNoDataText	Text that is displayed if there are no values to display and **zPrevireActive** is `false`.	`String`
zNoValidXMLText	Text that is displayed if no valid XML file with entered name has been found and **zPreviewActive** is false.	`String`
zNumberOfDecimalPlaces	Denotes how many decimal places are to be displayed.	`Integer`
zPreviewActive	Display of a preview activated/deactivated. The preview can be displayed if There is no data present (the modeled diagram is filled with default values) or the XML file was not found or this does not contain a valid definition (an example Sankey diagram is displayed).	`Bool`
zRefreshRate	Rate at which the diagram is refreshed in ms.	`Integer`
zSankeyName	Name of the XML file with the modeling of the diagram.	`String`
zShowRelativeValues	Display of the values in absolute `false` or relative values `true`.	`Bool`

Note: Additional VSTA programming is necessary for the display of the Sankey diagrams in the zenon Web Client. You can find details on this in the display of WPF elements in the zenon Web Client (on page 93).

6.5.9 Temperature indicator - TemperatureIndicatorControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	`Double`
MinValue	Minimum value of the scale.	`Double`
MaxValue	Maximum value of the scale.	`Double`
MajorTicksCount	Number of main ticks on the scale.	`Integer`
MinorTicksCount	Number of sub ticks on the scale.	`Integer`
TickNegativColor	Color of the negative main tick (gradient to TickPositivColor).	`Color`
TickPositivColor	Color of the positive main tick (gradient to TickNegativColor).	`Color`
MinorTickColor	Color of the sub ticks.	`Color`
ElementBorderBrush	Color of the element border.	`Brush`
ElementBackgroundBrush	Color of element background.	`Brush`
ElementGlasReflection	Activate the glass effect on the element.	`Visibility`
ElementFontFamily	Element font.	`Font`
IndicatorColor	Color of the indicator fill color.	`Color`
IndicatorBorderColor	Color of the indicator border.	`Color`
MajorTickSize	Size of main ticks on the scale.	`Size`
MinorTickSize	Size of sub ticks on the scale.	`Size`
ScaleLetteringDistance	Distance of the scale caption (vertical), each x. main tick should be captioned.	`Integer`
IndicatorScaleDistance	Distance between indicator and scale (horizontal).	`Double`
ScaleFontSize	Font size of the scale.	`Double`
ScaleFontColor	Font color of the scale.	`Color`
Unit	Unit.	`String`
ElementStyle	Shape/type of element.	`Enum:` ▸ `SmallFrame` ▸ `Unit` ▸ `None`

6.5.10 Universal slider - UniversalReglerControl

Property	Function	Value
CurrentValue	Current value which should be displayed.	`Double`
ElementFontFamily	Element font.	`Font`
MinValue	Minimum value of the scale.	`Double`
MaxValue	Maximum value of the scale.	`Double`
Radius		`Double`
ScaleRadius	Radius of the scale.	`Double`
ScaleStartAngle	Angle at which the scale starts.	`Double`
ScaleLabelRotationMode	Alignment of the scale caption.	`Enum:` ▸ `None` ▸ `Automatic` ▸ `SurroundIn` ▸ `SurroundOut`
ScaleSweepAngle	Angel area which defines the size of the scale.	`Double`
ScaleLabelFontSize	Font size of the scale caption.	`Double`
ScaleLabelColor	Font color of the scale caption.	`Color`
ScaleLabelRadius	Radius on which the scale caption is orientated.	`Double`
ScaleValuePrecision	Accuracy of the scale caption.	`Integer`
ElementStyle	Display type of the element	`Enum:` ▸ `Knob` ▸ `Plate` ▸ `None`
MajorTickColor	Color of main ticks on the scale.	`Color`
MinorTickColor	Color of sub ticks on the scale.	`Color`
MajorTickSize	Size of main ticks on the scale.	`Size`
MinorTickSize	Size of sub ticks on the scale.	`Size`
MajorTicksCount	Number of main ticks on the scale.	`Integer`
MajorTicksShape	Shape/type of main ticks on the scale.	`Enum:` ▸ `Rectangle` ▸ `Trapezoid` ▸ `Triangle`

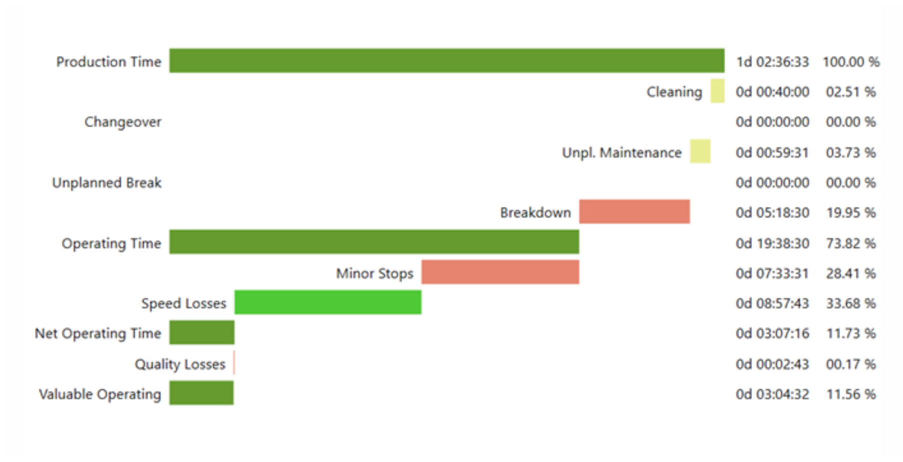
MinorTicksShape	Shape/type of sub ticks on the scale.	`Enum:` ▸ `Rectangle` ▸ `Trapezoid` ▸ `Triangle`
MinorTicksCount	Number of sub ticks on the scale.	`Integer`
BackgroundBorderBrush	Color of the element border.	`Brush`
BackgroundBrush	Color of element background.	`Brush`
PointerCapColorBrush	Color of pointer fastening point.	`Brush`
GaugeMiddlePlate	Radius of the element background middle plate.	`Double`
ValueFontSize	Font size of the value display.	`Double`
ValueFontColor	Font size of the value display.	`Color`
IsGlasReflection	Activate the glass effect on the element.	`Boolean`
KnobBrush	Color of the knob.	`Brush`
IndicatorBrush	Color of the indicator.	`Brush`
IndicatorBackgroundBrush	Background color of the inactive indicator.	`Brush`
KnobSize	Diameter of the knob.	`Double`
KnobIndicatorSize	Indicator size of the knob.	`Size`
ElementSize	Size of the element.	`Size`
VisibilityKnob	Activating of the knob.	`Boolean`
ValuePosition	Position of the value display.	`Double`
ValueVisibility	Activating the value display.	`Boolean`

6.5.11 Waterfall diagram

The waterfall diagram, WPF element is available to partners of COPA-DATA and is available to them via the COPA-DATA Partner Community (https://www.copadata.com/en-us/partner-community/).

The Meaning and waterfall chart Wizard must be used to model a waterfall diagram. A waterfall can be modeled with this wizard. The information is saved directly to the variables in the **Parameters for waterfall diagram** property (**Analyzer** variable properties group).

An example of a waterfall diagram in Runtime is shown below:



Note: This screenshot is only available in English.

The following settings can be made in the WPF configuration window under **COPADATA-ELEMENT**:

Property	Function	Value
zenonRefreshRate	Time between the refreshes of the diagram in miliseconds.	`Integer`
zenonWaterfallIdentifier	Name of the waterfall diagram.	`String`
zenonZSystemModel	Equipment group of the variables used.	`String`

Note: Additional VSTA programming is necessary for the display of the waterfall diagram in the zenon Web Client. You can find details on this in the display of WPF elements in the zenon Web Client (on page 93).

LINK BARS TO ZENON FUNCTION

The bars of a waterfall diagram can be linked to a function in Runtime. In Runtime, both the bars, as well as the labeling and value display for executing the function, can be clicked on.

Carry out the following configuration to link the columns of your waterfall diagram to a function:

1. Configure the WPF element for the waterfall diagram.
   Note: To do this, use the Meaning and Waterfall Chart wizard if possible.
2. Engineer a zenon function.
   a) Create a new function:

      In the toolbar or in the context menu of the Functions node, select **New function**.
      The dialog to select a function is opened.

   b) Select the desired function.

   c) Set the parameters for function.

3. Name the function in the **Name** property.
   Please note: The function name must contain the variables for the waterfall diagram without color code!
   You can also find these parameters in the **Parameters for waterfall diagram** variable property in the **Analyzer** properties group.
4. Link the function to the exact same equipment group as the variables.
   Note: You can find this linking in the **Equipment Groups** property of the function.

The following is applicable for this project configuration:

- The function and the linked variables must be present in the same zenon project.
- The variables must be linked to an equipment group.
- The function must be linked to the same equipment group as the variables.

Example

For a bar with the waterfall definition `WF= WF1,02,05,#E9ED92;` The function name, for example `Function_WF1,02,05`, is to be used.

6.6 Display of WPF elements in the zenon web client

In order to also be able to also use the pre-made WPF elements **"energy class diagram"**, **"Sankey diagram"** and **"waterfall chart"** for the display in a zenon web client, amendments are necessary in the project:

- Engineering in the zenon Editor (on page 93)
- Adapt VSTA code (on page 94)

6.6.1 Engineering in the zenon Editor

Carry out the following project configuration steps in the zenon Editor, in order to also be able to display certain WPF elements in the zenon web client:

PLACE WPF IN THE ZENON SCREEN:

- Place the WPF element in a zenon screen.
- Give it a unique name in the **Element name** property.
  You can find this property in the **General** properties group.

Note: A warning dialog appears if the name for an element has already been issued in another screen.

- Use the element name issued here in the VSTA code.

6.6.2 VSTA code (complex)

In order to add the programmer code for the display of WPF elements in the zenon web client, carry out the following steps:

1. In the zenon Editor, switch to the **programmer interfaces** node.
2. Select the **VSTA** node and select the **Open VSTA Editor with project add-in...** with a right mouse click
3. The dialog to create a VSTA project is opened.
4. Select the C# entry in the **Create new VSTA project** dialog.
5. Create (copy) the code below.
6. Enter the name of the WPF element in the code.

Note: When opening the VSTA editor, note whether the content of the following code is already present in the project configuration. For the display of the WPF element in the web client, compare the existing code and undertake the necessary additions. Please note the comments in relation to this in the model code.

VSTA CODE

```
//As member:
zenOn.IDynPictures zScreens = null;
string[] WPFElements ={"WPF_Control", "WPFWebclient_1", "WPFWebclient_2" }; //Names of the WPF screen elements that appear in the zenon project and that need access to the API (as many/few as you want)

//Add the following three lines of code in the project archive function:
void ThisProject_Active()
{
  zScreens = this.DynPictures();
  zScreens.Open += new zenOn.DDynPicturesEvents_OpenEventHandler(zScreens_Open);
  zScreens.Close += new zenOn.DDynPicturesEvents_CloseEventHandler(zScreens_Close);
}
```

```
//Add the following two lines of code in the project inactive function:
void ThisProject_Inactive()
{
  zScreens.Open -= new zenOn.DDynPicturesEvents_OpenEventHandler(zScreens_Open);
  zScreens.Close -= new zenOn.DDynPicturesEvents_CloseEventHandler(zScreens_Close);

  //Final release and garbage collection of any API-Objects.
  FreeObjects();
}

//Add two new event handlers:
void zScreens_Open(zenOn.IDynPicture obDynPicture)
{
  foreach (string element in WPFElements)
  {
    if (obDynPicture.Elements().Item(element) != null)
    {
      obDynPicture.Elements().Item(element).set_WPFProperty("ELEMENT",
"zenonVariableLink", this.Variables().Item(0));
    }
  }
}
void zScreens_Close(zenOn.IDynPicture obDynPicture)
{
  foreach (string element in WPFElements)
  {
    if (obDynPicture.Elements().Item(element) != null)
    {
      zenOn.IElement zWPFElement= obDynPicture.Elements().Item(element);
      zWPFElement.set_WPFProperty("ELEMENT", "zenonTrigger", true);
      zWPFElement = null;
    }
  }
}
```

6.6.3 VSTA code (simplified)

If only one WPF element is used in a zenon screen, the following more streamlined code can be used as an alternative. To do this, the names of the WPF element, and the screen in which the element is used, must be entered. This code is then recommended if, for each project, only one of the pre-made WPF elements is used.

VSTA CODE

```
zenOn.IDynPicture zScreen = zero;
string wpfElement = "WPF_Control"; //Name of the WPF element in the screen
string wpfPicture = "@Details_Overview_Online"; //Name of the zenon screen

//Add to the project active function:
void ThisProject_Active()
{
  zScreen = this.DynPictures().Item(wpfPicture);
  zScreen.Open += new zenOn.OpenEventHandler(zScreen_Open);
  zScreen.Close += new zenOn.CloseEventHandler(zScreen_Close);
}

//Add to the project inactive function:
void ThisProject_Inactive()
{
  zScreen.Open -= new zenOn.OpenEventHandler(zScreen_Open);
  zScreen.Close -= new zenOn.CloseEventHandler(zScreen_Close);

  //Final release and garbage collection of any API-Objects.
  FreeObjects();
}

void zScreen_Open()
{
  if (zScreen.Elements().Item(wpfElement) != null)
  {
       zScreen.Elements().Item(wpfElement).set_WPFProperty("ELEMENT",
"zenonVariableLink", this.Variables().Item(0));
  }
}
```

```
void zScreen_Close()
{
  if (zScreen.Elements().Item(wpfElement) != null)
  {
       zenOn.IElement zWPFElement = zScreen.Elements().Item(wpfElement);
       zWPFElement.set_WPFProperty("ELEMENT", "zenonTrigger", true);
       zWPFElement = null;
  }
}
```

6.7 Examples: Integration of WPF in zenon

You can see how XAML files are created and integrated as WPF elements in zenon from the following examples:

- Integrate button as WPF XAML in zenon (on page 102)
- Integrate bar graph as WPF XAML in zenon (on page 97)
- Integrate DataGrid Control in zenon (on page 108)

6.7.1 Integrate bar graph as WPF XAML in zenon

Example structure:

- Creating a bar graph (on page 16) in Adobe Illustrator and converting it to WPF
- Integrate into zenon
- Linking with variables
- Adapting the bar graph WPF element

CREATE BAR GRAPH

The first step is to generate a bar graph as described in the Workflow with Adobe Illustrator (on page 16) chapter. To be able to use the XAML file in zenon, insert this in the project tree in the **Files/graphics** folder.

INTEGRATE BAR GRAPH

Note: A zenon project with the following content is used for the following description:

- An empty screen as a start screen
- Four variables from the internal driver for
  - Scale `0`
  - Scale `central`
  - Scale `high`
  - Current value
- A variable from the mathematics driver for displaying the current value (`255`)

To integrate the bar graph:

1. open the empty screen
2. place a **WPF element** (on page 50) in the screen
3. select **XAML file** in the properties window
4. Select the desired XAML file (for example **bar graph_vertical.xaml**) and close the dialog

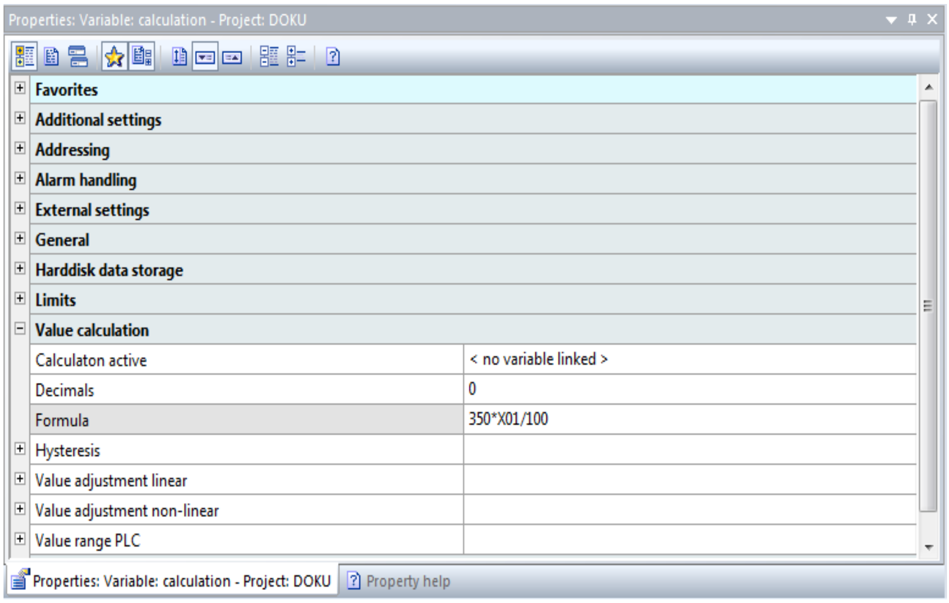
ADJUST BAR GRAPH

Before configuration, the scale of the XAML file is adapted if necessary:

To do this:

- Create a new mathematics variable that calculates the new value in relation to the scaling, for example:
- Variable: 0-1000

- Mathematic variable {value created in xaml file}*Variable/1000


Properties: Variable: calculation - Project: DOKU

- Favorites
- Additional settings
- Addressing
- Alarm handling
- External settings
- General
- Harddisk data storage
- Limits
- Value calculation

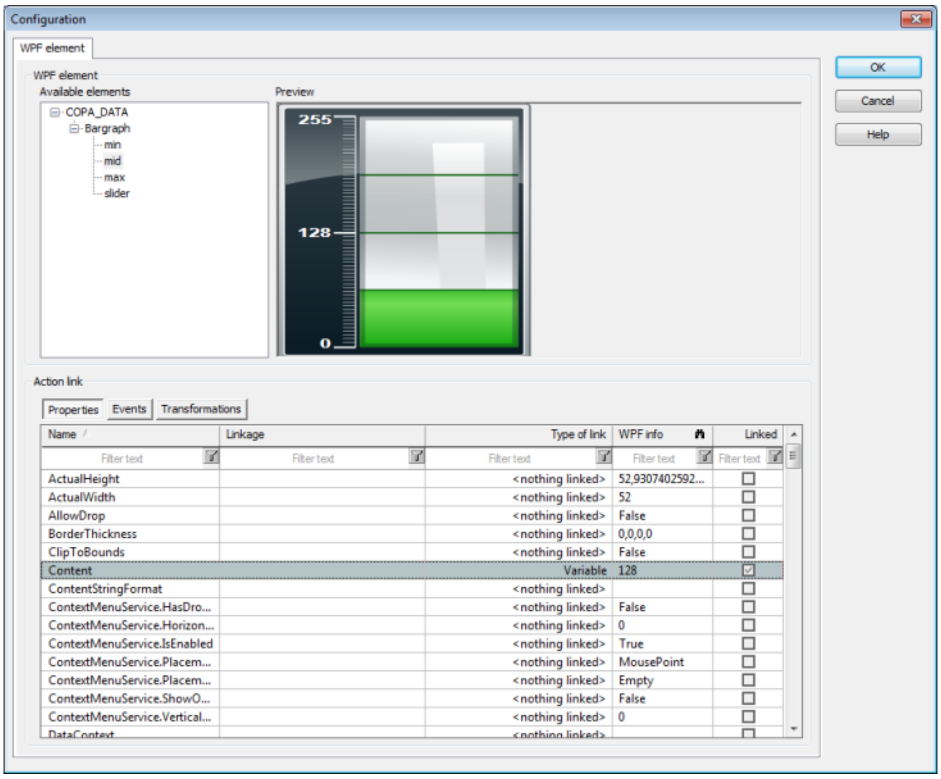
Calculaton active	< no variable linked >
Decimals	0
Formula	350*X01/100
Hysteresis	
Value adjustment linear	
Value adjustment non-linear	
Value range PLC	

The XAML file is then configured.

CONFIGURE BAR GRAPH

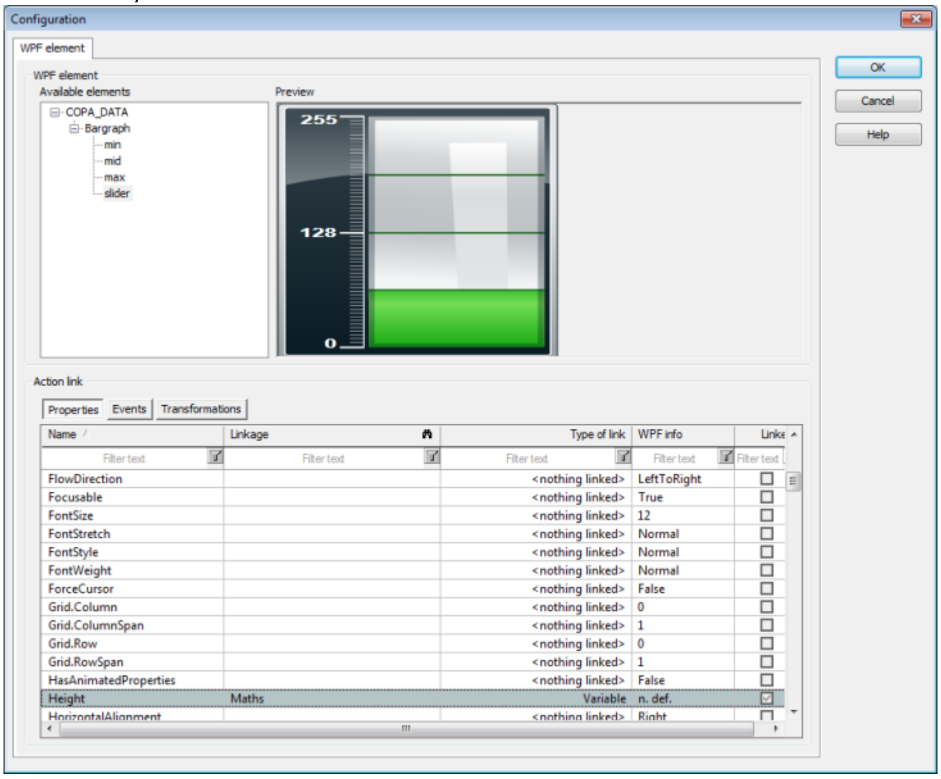
1. Click on the WPF element and select the **Configuration** property
2. The configuration dialog shows a preview of the selected XAML file.

3. Select the minimum value, the average value and the maximum value and link each of these to the corresponding variable in the **Content** property



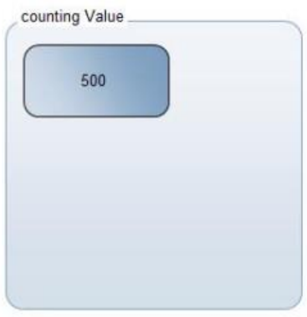
Name	Linkage	Type of link	WPF info	Linked
ActualHeight		<nothing linked>	52,9307402592...	☐
ActualWidth		<nothing linked>	52	☐
AllowDrop		<nothing linked>	False	☐
BorderThickness		<nothing linked>	0,0,0,0	☐
ClipToBounds		<nothing linked>	False	☐
Content		Variable	128	✓
ContentStringFormat		<nothing linked>		☐
ContextMenuService.HasDro...		<nothing linked>	False	☐
ContextMenuService.Horizon...		<nothing linked>	0	☐
ContextMenuService.IsEnabled		<nothing linked>	True	☐
ContextMenuService.Placem...		<nothing linked>	MousePoint	☐
ContextMenuService.Placem...		<nothing linked>	Empty	☐
ContextMenuService.ShowO...		<nothing linked>	False	☐
ContextMenuService.Vertical...		<nothing linked>	0	☐
DataContext		<nothing linked>		☐

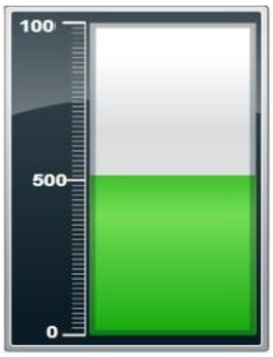
4. Select the **Slider** and link the **Value** property to the mathematics variables (in our example: calculation)



Name	Linkage	Type of link	WPF info	Linke
FlowDirection		<nothing linked>	LeftToRight	☐
Focusable		<nothing linked>	True	☐
FontSize		<nothing linked>	12	☐
FontStretch		<nothing linked>	Normal	☐
FontStyle		<nothing linked>	Normal	☐
FontWeight		<nothing linked>	Normal	☐
ForceCursor		<nothing linked>	False	☐
Grid.Column		<nothing linked>	0	☐
Grid.ColumnSpan		<nothing linked>	1	☐
Grid.Row		<nothing linked>	0	☐
Grid.RowSpan		<nothing linked>	1	☐
HasAnimatedProperties		<nothing linked>	False	☐
Height	Maths	Variable	n. def.	✓
HorizontalAlignment		<nothing linked>	Right	☐

5. Check the project planning in Runtime:





6.7.2 Integrate button as WPF XAML in zenon

Example structure:

- Creating a button (on page 12) in Microsoft Expression Blend
- Integrate into zenon
- Link to a variable and a function
- adjust the button to the size of the element
- Create button

As a first step, create a button as described in the Create button as XAML file with Microsoft Expression Blend (on page 12) chapter. To be able to use the XAML file in zenon, insert this in the project tree in the **Files/graphics** folder.

INTEGRATE BUTTON

Note: A zenon project with the following content is used for the following description:

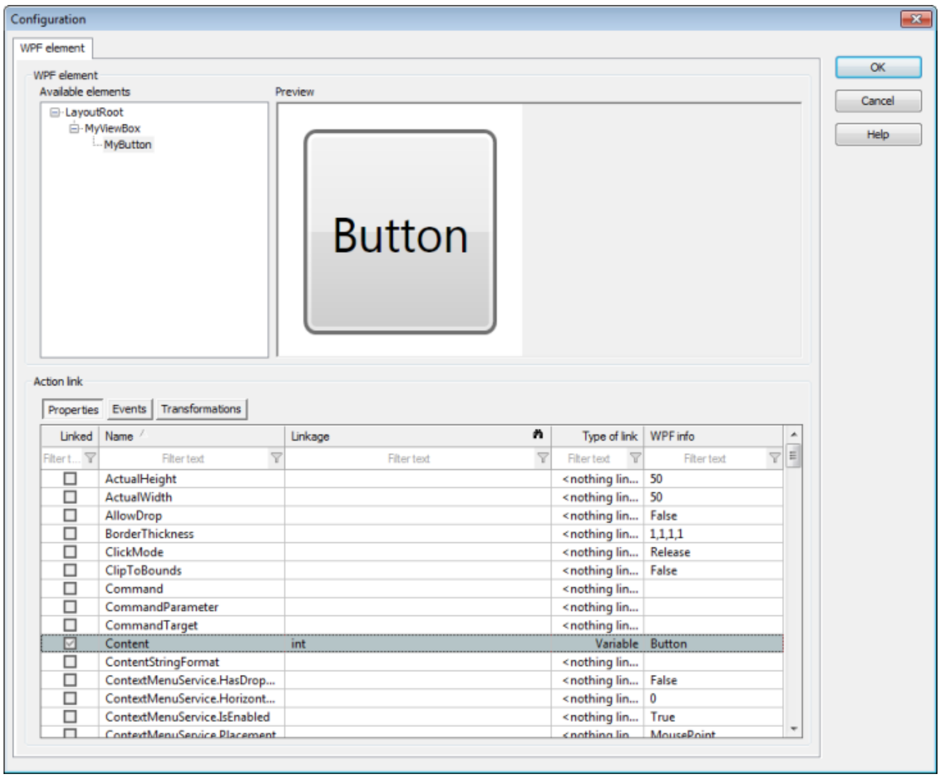
- An empty screen as a start screen
- an internal variable **int** of type **Int**
- a function **Funktion_0** of type**Send value to hardware** with:
  - **Direct to hardware** option activated
  - Set was set to `45`

To integrate the button:

1. open the empty screen
2. place a **WPF element** (on page 50) in the screen
3. select **XAML file** in the properties window
4. select the XAML file (e. g. **MyButton.xaml** and close the dialog
5. select the **Configuration** property

CONFIGURE THE BUTTON

The configuration dialog shows a preview of the selected XAML file. All elements named in the XAML file are listed in the tree:



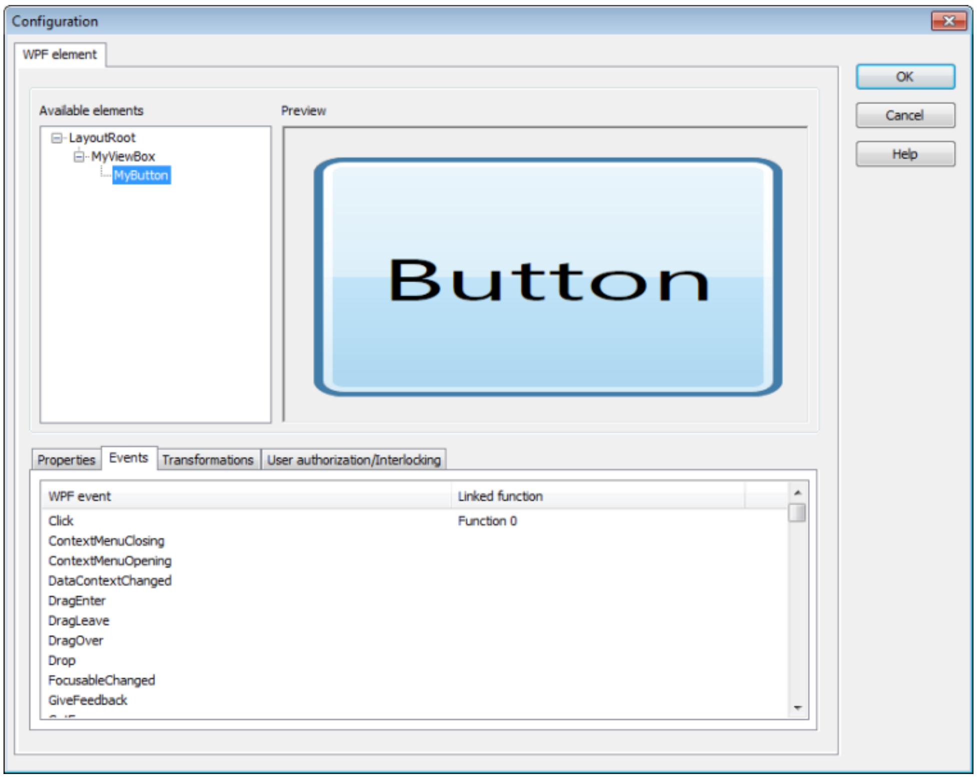
Linked	Name	Linkage	Type of link	WPF info
Filter t...	Filter text	Filter text	Filter text	Filter text
☐	ActualHeight		<nothing lin...	50
☐	ActualWidth		<nothing lin...	50
☐	AllowDrop		<nothing lin...	False
☐	BorderThickness		<nothing lin...	1,1,1,1
☐	ClickMode		<nothing lin...	Release
☐	ClipToBounds		<nothing lin...	False
☐	Command		<nothing lin...	
☐	CommandParameter		<nothing lin...	
☐	CommandTarget		<nothing lin...	
✓	Content	int	Variable	Button
☐	ContentStringFormat		<nothing lin...	
☐	ContextMenuService.HasDrop...		<nothing lin...	False
☐	ContextMenuService.Horizont...		<nothing lin...	0
☐	ContextMenuService.IsEnabled		<nothing lin...	True
☐	ContextMenuService.Placement		<nothing lin...	MousePoint

1. select the WPF button, which is in *LayoutRoot->MyViewBox->MyButton*
2. Look in the **Properties** Entry**Content** tab; this contains the button's text
3. Click the **Link type** column
4. Select **Variable** from the drop down list
5. Click in the **Link** column
6. the variable selection dialog is opened
7. select the `int` variable to link this variable with the **Content** property

EVENTS

To also assign events:

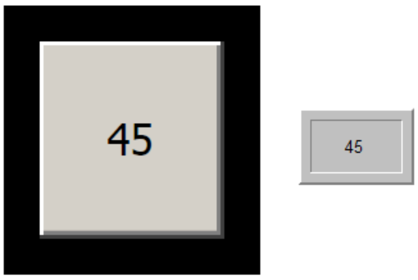
1. select the tab Events



WPF event	Linked function
Click	Function 0
ContextMenuClosing	
ContextMenuOpening	
DataContextChanged	
DragEnter	
DragLeave	
DragOver	
Drop	
FocusableChanged	
GiveFeedback	

2. look for the 'Click' entry, this event is triggered by the WPF element, as soon as the button is clicked
3. Click in the **Link type** column
4. Select **Function** from the drop down list
5. Click in the **Link** column
6. the **function selection dialog** is opened
7. select **Function_0**
8. Confirm the changes with **OK**
9. Insert a **numerical value element** into the screen
10. Link this **numerical value element** to the `int` variables too.
11. Compile the Runtime files and start Runtime.

The **WPF element** is displayed in Runtime, the button text is `0`. As soon as you click on the button, the **click** event is triggered and the **set value** function is carried out. The value `45` is sent directly to the hardware and both **numerical value** and **button** display the value `45` .

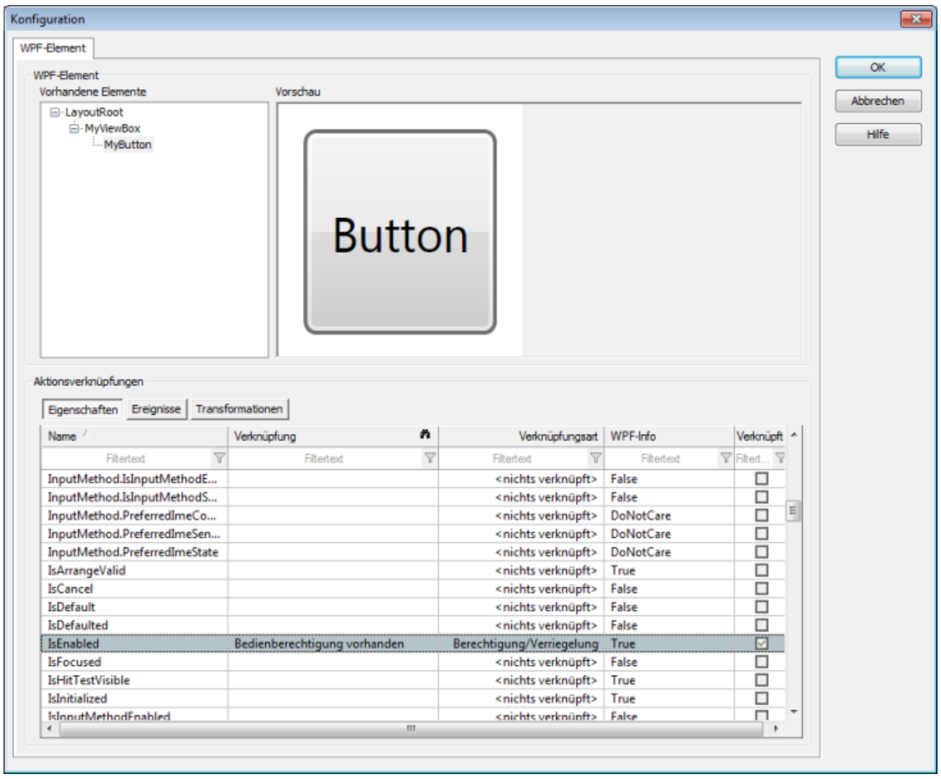


Define a set value of `30` via the **numerical value element**; this value is then also assumed by the **WPF element**.

AUTHORIZATION

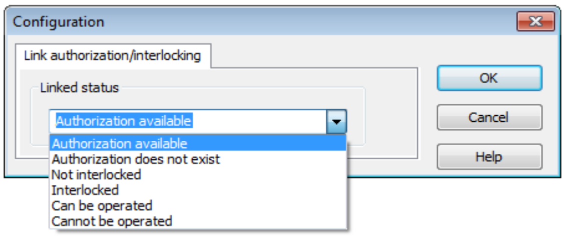
Similar to a **numerical value,** a **WPF** element can be locked according to authorizations (lock symbol) or switched to be operable. Set the user authorization level to `1` for the **WPF element** and create a user called **Test** with **authorization level 1**. In addition, set up the functions **Login with dialog** and **Logout** . You link these two functions with 2 new text buttons on the screen.

In the **WPF element** configuration dialog, select the **MyButton** WPF button and select the **Properties:** tab



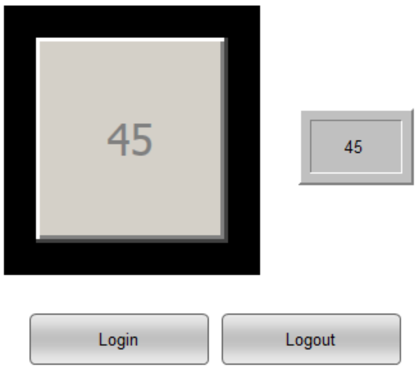
Name	Verknüpfung	Verknüpfungsart	WPF-Info	Verknüpft
InputMethod.IsInputMethodE...		<nichts verknüpft>	False	☐
InputMethod.IsInputMethodS...		<nichts verknüpft>	False	☐
InputMethod.PreferredImeCo...		<nichts verknüpft>	DoNotCare	☐
InputMethod.PreferredImeSen...		<nichts verknüpft>	DoNotCare	☐
InputMethod.PreferredImeState		<nichts verknüpft>	DoNotCare	☐
IsArrangeValid		<nichts verknüpft>	True	☐
IsCancel		<nichts verknüpft>	False	☐
IsDefault		<nichts verknüpft>	False	☐
IsDefaulted		<nichts verknüpft>	False	☐
IsEnabled	Bedienberechtigung vorhanden	Berechtigung/Verriegelung	True	☑
IsFocused		<nichts verknüpft>	False	☐
IsHitTestVisible		<nichts verknüpft>	True	☐
IsInitialized		<nichts verknüpft>	True	☐
IsInputMethodEnabled		<nichts verknüpft>	False	☐

1. Select the **IsEnabled** element
2. Click in the **Link type** column
3. Select **Authorizations/interlocking** from the drop down list
4. Click in the **Link** column
5. In the drop-down list, select the `Authorized` option


Configuration
Link authorization/interlocking
Linked status: Authorization available

6. Close the dialog with **OK**

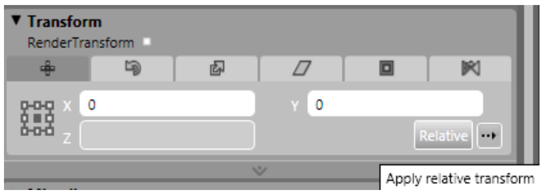
Compile the Runtime file and note that Authorizations to be Transferred must also be selected. After Runtime has been started, the WPF button is displayed as deactivated on the screen and cannot be operated. If you now log in as the user **Test**, the button is activated and can be operated. The button is locked again as soon as you log out.



TRANSFORMATION

The XAML files must still be adapted to use transformations:

1. switch to the **Expression Blend** program
2. select **MyButton**, so that the properties of the element are visible in the events window

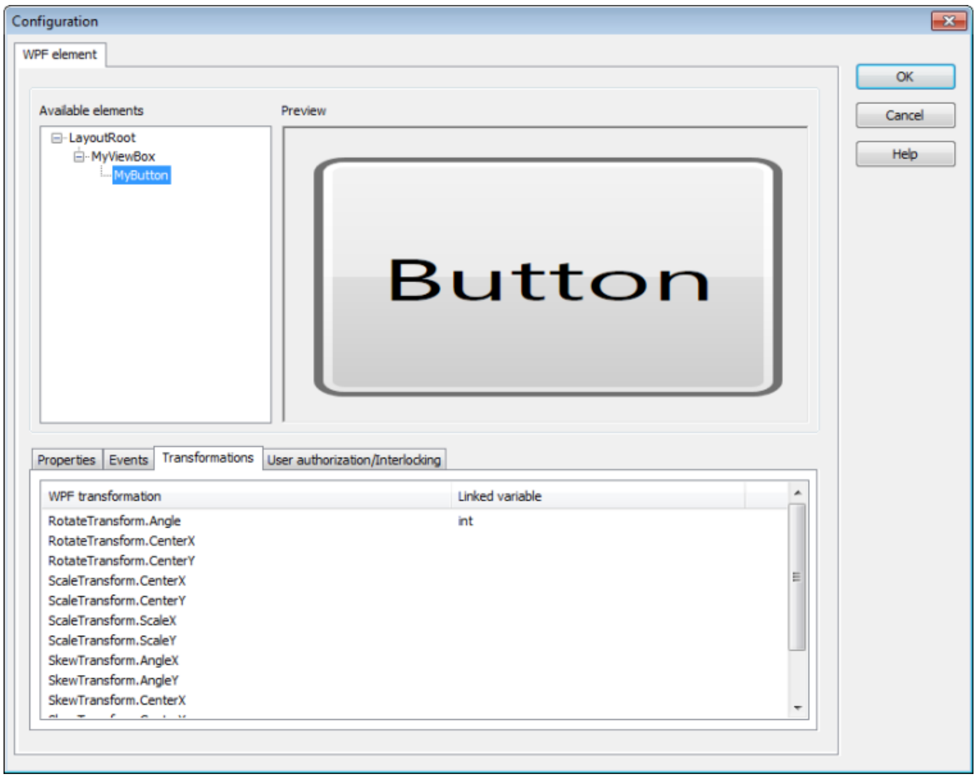


3. Under **Transform** at **RenderTransform** select the **Apply relative transform** option

As a result of this, a block is inserted into the XAML file, which save the transformation settings in runtime.

```
<Button.RenderTransform>
    <TransformGroup>
        <ScaleTransform ScaleX="1" ScaleY="1"/>
        <SkewTransform AngleX="0" AngleY="0"/>
        <RotateTransform Angle="0"/>
        <TranslateTransform X="0" Y="0"/>
    </TransformGroup>
</Button.RenderTransform>
```

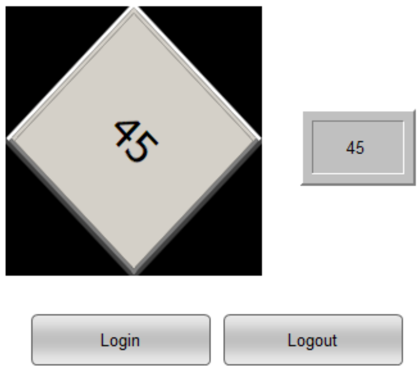
4. Save the file and replace the old version in zenon with this new file.
5. Open the **WPF element** configuration dialog again:
   a) select the **MyButton** button
   b) select the **Transformations** tab



WPF transformation	Linked variable
RotateTransform.Angle	int
RotateTransform.CenterX	
RotateTransform.CenterY	
ScaleTransform.CenterX	
ScaleTransform.CenterY	
ScaleTransform.ScaleX	
ScaleTransform.ScaleY	
SkewTransform.AngleX	
SkewTransform.AngleY	
SkewTransform.CenterX	

   c) select the element **RotateTransform.Angle**
   d) Click in the **Link type** column
   e) Select **Transformations** from the drop down list
   f) Click in the **Link** column
   g) the variable selection dialog is opened
   h) select the **int** variable to link this variable with the **RotateTransform.Angle** property

Compile the Runtime files and start Runtime. Log in as the **Test** user and click on the button. The button has the value `45` and the **WPF element** rotates by 45°.



6.7.3 Integrate DataGrid Control in zenon

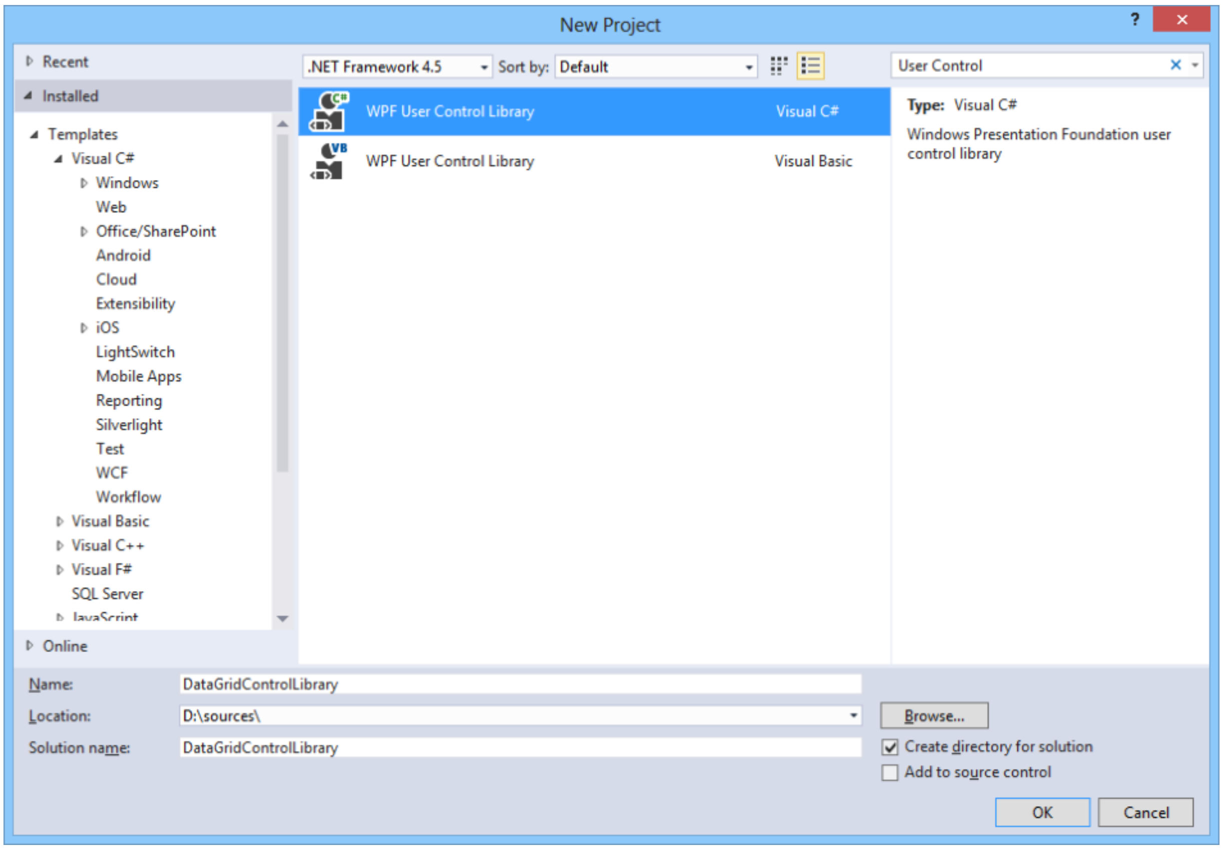
To create DataGrid control for zenon, you need:

- Visual Studio (Visual Studio 2015 in this example)

CREATE WPF USER CONTROL

1. in Visual Studio, create a new **Solution** and a **WPF User Control Library** project in .NET Framework version 4 or higher therein.

   Info: If the corresponding project template does not appear in the list of available templates, this can be added by means of the search (field at the top right of the dialog).



Type: Visual C#

Windows Presentation Foundation user control library

Name: DataGridControlLibrary

Location: D:\sources\

Solution name: DataGridControlLibrary

☑ Create directory for solution

☐ Add to source control

In our example, the project is given the name **DataGridControlLibrary**.

2. Create a new data connection in the **Server Explorer**.

   In our example, the database `Northwind` is used, which is provided by Microsoft as an example database that can be downloaded for free.

   Te set up the database connection:

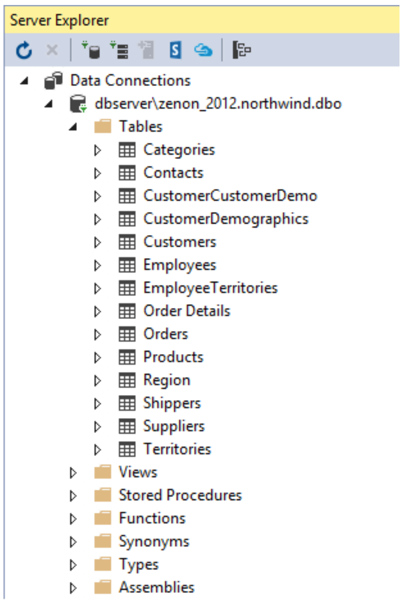
   a) Right-click on **Data Connections**.

   b) Select **Add connection...**.

   c) Select `Microsoft SQL Server (SQLClient)` as **Data source**.

   d) Select the corresponding server and database name.

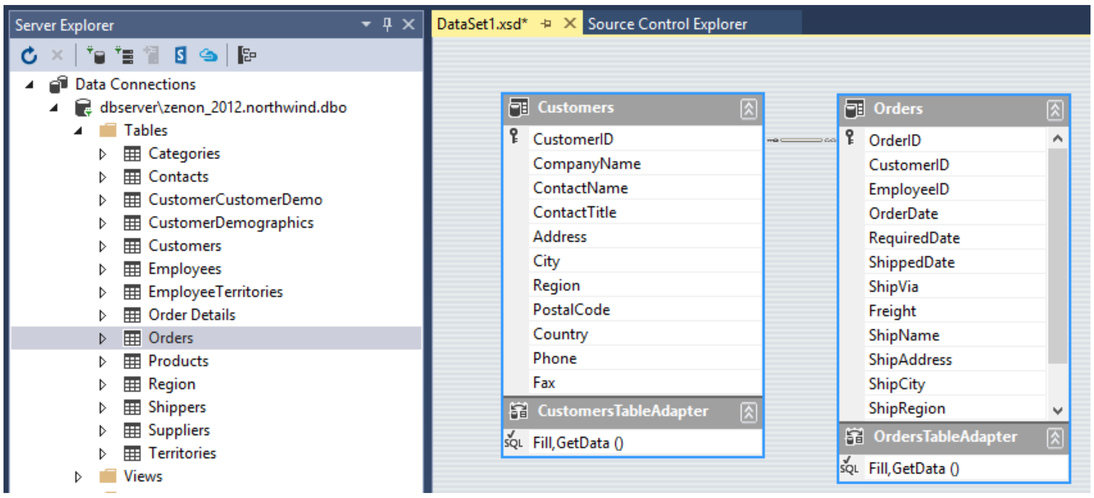
After adding the connection, the Server Explorer window should look a little like this:



A new DataSet is created in the next step.

CREATING A DATASET

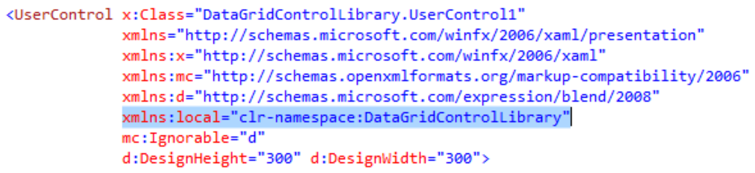
1. Right-click on the project
2. Select **Add – New Item...** in the context menu
3. Create a new **DataSet** with the name `DataSet1`.
4. Double click on the DataSet in order to open it in the Designer.
5. Drag the tables that you need (`Customers` and `Orders` in this example) to the DataSet design window.



The XAML file is modified in the next step.

CONFIGURATION OF THE XAML FILE

1. If not already there, add the **Namespace** as a reference to the class in the XAML file:



```
<UserControl x:Class="DataGridControlLibrary.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataGridControlLibrary"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
```

2. Define the resources and the DataGrid that is to be used in the WPF:

   <UserControl.Resources>

   <local:DataSet1 x:Key="DataSet1"/>

   <CollectionViewSource x:Key="CustomersViewSource" Source="{Binding Path=Customers, Source={StaticResource DataSet1}}"/>

   </UserControl.Resources>

   <Grid DataContext="{StaticResource CustomersViewSource}">

   <DataGrid Name="DataGrid1" DisplayMemberPath="CompanyName" ItemsSource="{Binding}" SelectedValuePath="CustomerID" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>

   </Grid>

3. Open the code-behind file (**UserControl1.xaml.cs**) and insert the following lines in the constructor:

   public UserControl1()

   {

   InitializeComponent();

   DataSet1 ds = ((DataSet1)(FindResource("DataSet1")));

   DataSet1TableAdapters.CustomersTableAdapter ta = new DataSet1TableAdapters.CustomersTableAdapter();

   ta.Fill(ds.Customers);

   CollectionViewSource CustomersViewSource = ((CollectionViewSource)(this.FindResource("CustomersViewSource")));

   CustomersViewSource.View.MoveCurrentToFirst();

   }

   In doing so, the following happens:

   - The DataSet is obtained
   - A new TableAdapter is created
   - The DataSet is filled

- The information is provided to the DataGrid control

The solution can now be built.

BUILD

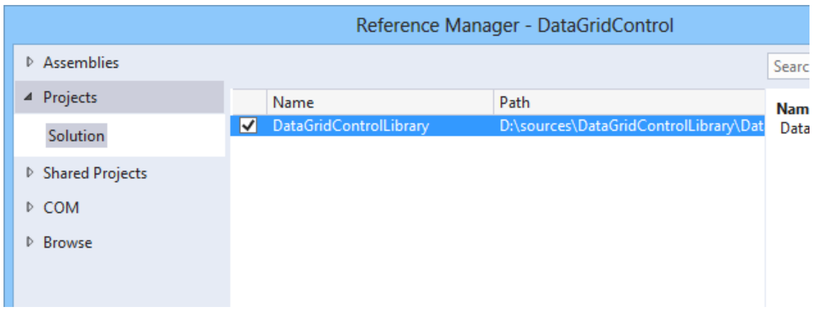
Now build the solution. The corresponding DLL (**DataGridControlLibrary.dll**) is created in the output folder of the project.

Now you have a DLL with the necessary functionality available.

However zenon can only display XAML files that cannot be linked to the code behind file, which is why an additional XAML file is needed that references the DLL that has just been created.

To do this:

1. Create a further project, again as a **WPF User Control Library**
2. It was called **DataGridControl** in our example.
3. Insert a reference to the project that has just been built into this new project.



	Name	Path
✓	DataGridControlLibrary	D:\sources\DataGridControlLibrary\Dat

4. The XAML files (**UserControl1.xaml**) looks as follows:

```
<UserControl x:Class="DataGridControl.UserControl1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataGridControl"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
  <Grid>

  </Grid>
</UserControl>
```

5. Because all necessary content is contained in the DLL that has been created and no code-behind is necessary, delete the following lines:

   x:Class=`"DataGridControl.UserControl1"`

   xmlns:local=`"clr-namespace:DataGridControl"`

6. Also delete (for the positioning) the following lines:

   mc:Ignorable=`"d"`

   d:DesignHeight=`"300"` d:DesignWidth=`"300"`

7. Delete the code-behind file (**UserControl1.xaml.cs**) in this project.
8. Define what is to be displayed in the XAML file.

   To do this, modify the XAML file as follows:

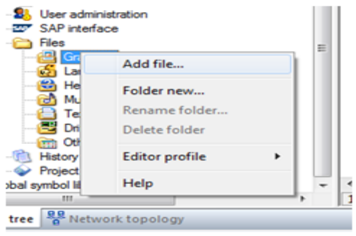
   ```
   <UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
         xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
         xmlns:dataGridLibrary="clr-namespace:DataGridControlLibrary;assembly=DataGridControlLibrary">
   <Grid x:Name="Grid1">
         <dataGridLibrary:UserControl1 Name="DataGridControl" HorizontalAlignment="Left"
         VerticalAlignment="Top"/>
   </Grid>
   </UserControl>
   ```

   The line`xmlns:dataGridLibrary="clr-namespace:DataGridControlLibrary;assembly=DataGridControlLibrary"` defines the namespace **dataGridLibrary** and stipulates that this should use the assembly that has been created.
9. Assign a name for the grid.
10. Insert the control **dataGridLibrary:UserControl1** from our library and give it a name, because zenon can only modify objects that have a name.
11. Build the solution.

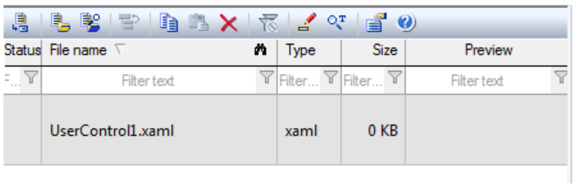
In the next step, how the DLL and XAML file are added to zenon is explained.

STEPS IN ZENON

1. Open the zenon Editor
2. Go to `File -> Graphics`.
3. Select **Add file...** in the context menu



4. Select the XAML file at the save location (**UserControl1.xaml** from the **DataGridControl** project) and add this:



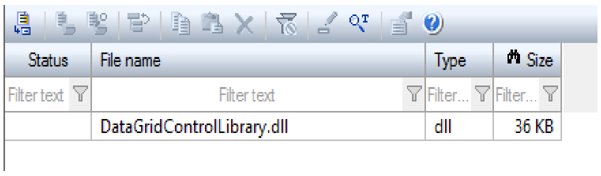
Status	File name	Type	Size	Preview
	UserControl1.xaml	xaml	0 KB	

5. Insert the DLL with the functionality for the XAML file.

   To do this:

   a) Select, in the context menu, File -> Other**Add file...**.

   b) Select the file **DataGridControlLibrary.dll** of the first project (**DataGridControlLibrary**).

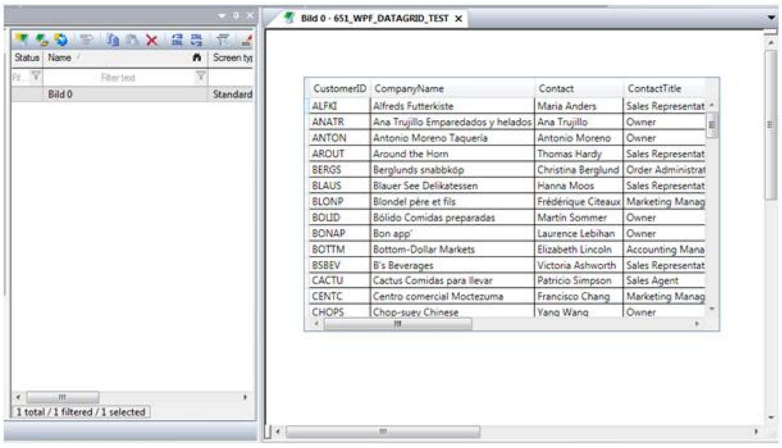


Status	File name	Type	Size
	DataGridControlLibrary.dll	dll	36 KB

6. Create a zenon screen.

7. Add a WPF element and select the previously-incorporated XAML file.

   You should now see the following in the zenon Editor:



Status	Name	Screen ty
	Bild 0	Standard

1 total / 1 filtered / 1 selected

Bild 0 - 651_WPF_DATAGRID_TEST

CustomerID	CompanyName	Contact	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representat
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representat
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrat
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representat
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manag
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Mana
BSBEV	B's Beverages	Victoria Ashworth	Sales Representat
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manag
CHOPS	Chop-suey Chinese	Yang Wang	Owner

8. Start zenon Runtime in order to also test the control there.

Hint

DLLs that belong to a WPF element (referenced by the linked XAML file) can also be replaced in the Editor during ongoing operation.
To replace a DLL:

- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In Explorer, replace the DLL in the `\wpfache` folder of the Editor files.
  You can find this folder in the SQL directory under
  `...\PROJECT-GUID\FILES\zenon\custom\wpfcache`

As an alternative to replacement using Explorer, you can also replace the file in the zenon Editor directly; to do this:

- In the Visual Studio project settings, increase the file version of the DLL.
- Create the new DLL.
- Close all zenon screens in which the WPF element is used.
- Close all symbols that use a desired WPF element.
- In the zenon Editor, delete the DLL from the `\Files\Other` folder and add the file with the higher version number.

6.8 Error handling

ENTRIES IN LOG FILES

Entry	Level	Meaning
Xaml file found in %s with different name, using default!	Warning	The name of the collective file and the name of the XAML file contained therein do not correspond. To avoid internal conflicts, the file with the name of the collective file and the suffix **.xaml** is used.
no preview image found in %s	Warning	The collective file does not contain a valid preview graphic (**preview.png** or **[names of the XAML file].png**). Thus no preview can be displayed.
Xaml file in %s not found or not unique!	Error	The collective file does not contain an XAML file or several files with the suffix **.xaml**. It cannot be used.
Could not remove old assembly %s	Warning	There is an assembly that is to be replaced with a newer version, but cannot be deleted.
Could not copy new assembly %s	Error	A new version is available for an assembly in the work folder, but it cannot be copied there. Possible reason: The old example is still loaded, for example. The old version continues to be used, the new version cannot be used,
file exception in %s	Error	A file error occurred when accessing a collective file.
Generic exception in %s	Error	A general error occurred when accessing a collective file.