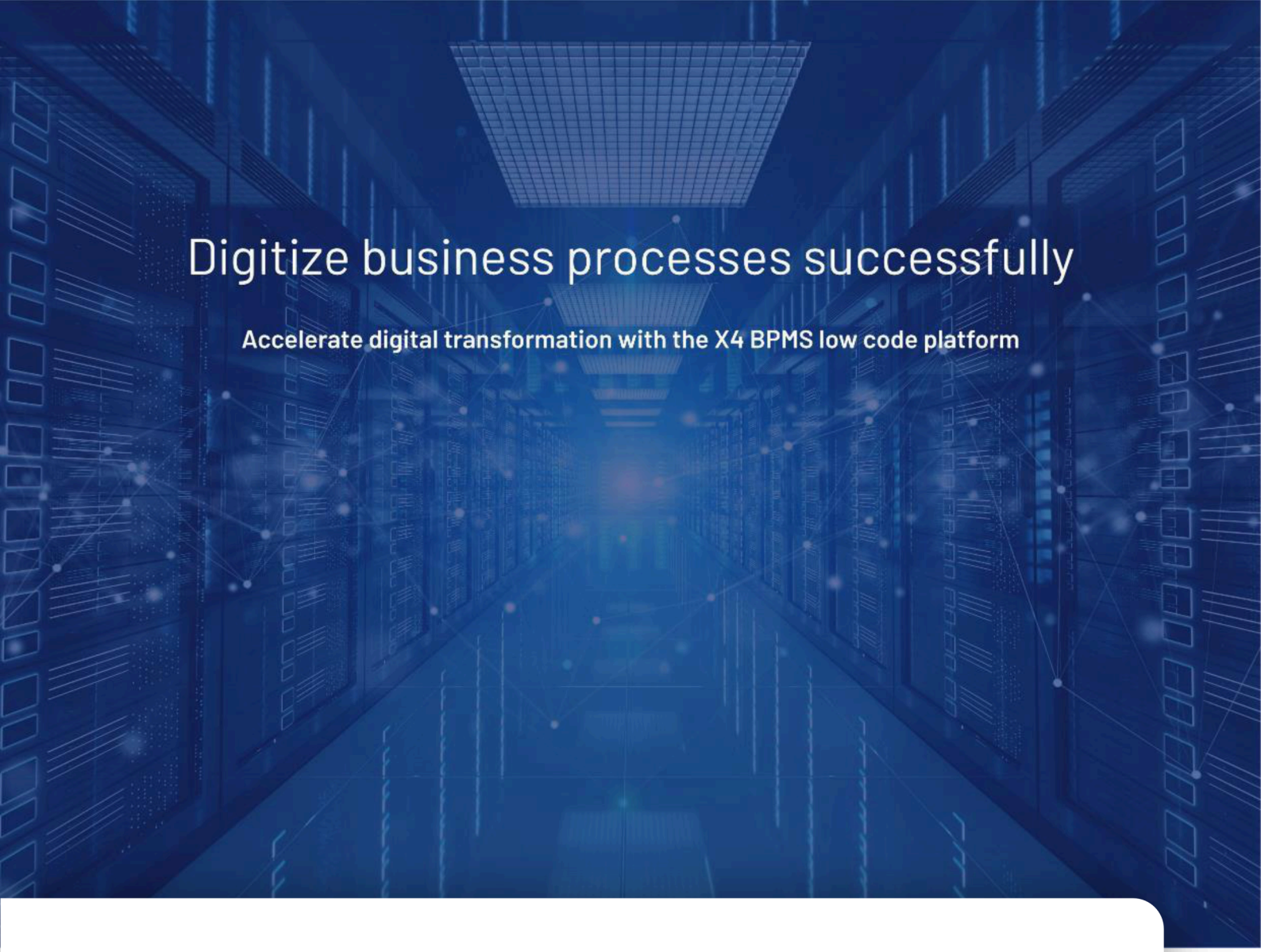
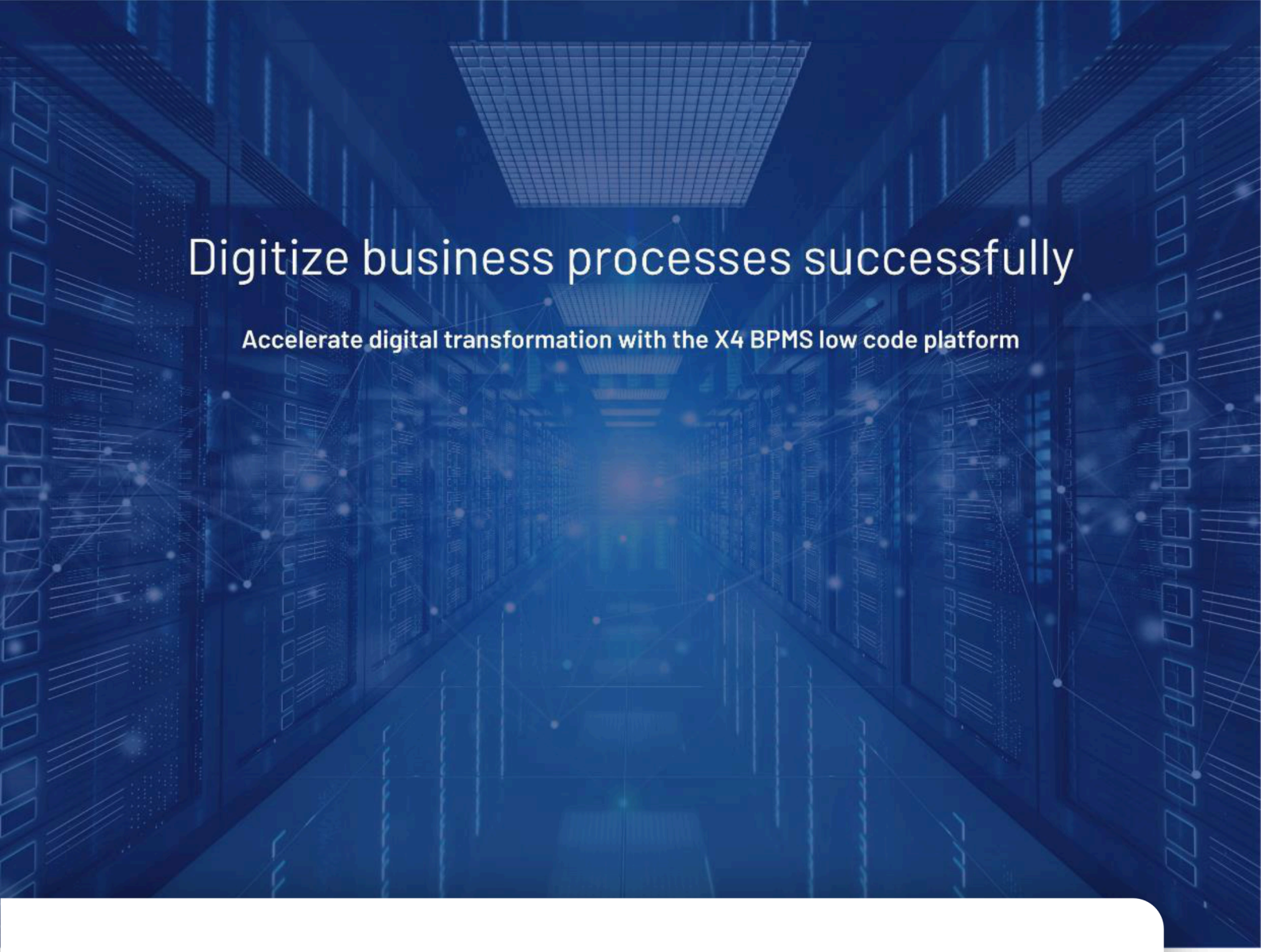


X4 Web Apps

SoftProject



Digitize business processes successfully

Accelerate digital transformation with the X4 BPMS low code platform

SoftProject

The information in this document is subject to change without notice. SoftProject GmbH assumes no responsibility for any errors that may appear in this document.

This document may not be copied, photocopied, reproduced, translated or converted to any electronic or machine-readable form in whole or in part without prior written approval of SoftProject GmbH.

Mentioned products are trademarks or registered trademarks of their respective owners.

Contact

SoftProject GmbH

Am Erlengraben 3

D-76275 Ettlingen – Germany

Website: www.softproject.de

Sales

Phone: +49 7243 56175-0

vertrieb@softproject.de

SoftProject Support

Phone: +49 7243 56175-333

support@softproject.de

© SoftProject GmbH. All rights reserved.

Table of Contents

1 Structure of a Web App project 15
1.1 Back End 15
1.2 Front End 16
2 Creating and Declaring a New Web App Project 17
2.1 Templates 17
2.1.1 ESB Project 17
2.1.2 BPM Project 17
2.1.3 Web App Project 18
2.2 Creating a New Web App Project 18
2.3 Display Web App project 19
2.3.1 Open on mobile devices 20
2.3.2 Open in browser 20
2.3.3 Copying URL of the Web App 21
2.4 Declaring a Web App project 21
2.4.1 Referencing Components in the Web App Definition 22
2.5 Configuring the Menu Width 27
2.6 Imprint 28
2.6.1 Creating an Imprint 29
2.6.2 Provice content in the Imprint 29
2.7 Data Protection Statement 30
2.7.1 Creating a file for the Data Protection Statement 31
2.7.2 Provide content in the Data Protection Statement 32
2.8 Responsive Design 33
2.8.1 Page Layout on a Desktop PC 33
2.8.2 Mobile Page Layout 34
3 Theming 35
3.1 Defining color scheme 35
3.2 Colors 35
3.2.1 Main Theme 35

3.2.2 Chart Theme 38
3.2.3 Defining colors 39
3.3 Fonts 39
3.3.1 Defining fonts 40
4 Configuration 42
4.1 Create Configuration 42
4.2 Managing access to Web Apps 42
4.3 Declaring Configuration 42
4.3.1 <General> 43
4.3.2 <Access> 45
4.3.3 <Status> 48
4.3.3.1 dataFormatVersion 49
4.3.4 <Logout> 50
4.3.5 <Notifications> 50
4.3.5.1 Technical Process 50
4.3.6 <HTTPSecurityHeader> 52
4.3.6.1 HTTPSecurityHeader 52
4.3.6.2 strictTransportSecurity 53
4.3.7 <HeaderIncludes> 53
4.3.7.1 Include 54
5 Localization 55
5.1 Creating a New Language File 55
5.2 Translation Keys 56
6 Login page 60
6.1 Create login page 61
6.1.1 <Login> 61
6.2 Create Forgot Password button 62
6.2.1 Example 62
7 Modules 63
8 Data model and properties definition 67
8.1 Data from the Technical Process 70

8.1.1 URL 70
8.2 Example for a simple properties definition 71
8.3 Example for a complex property definition 71
8.4 Validation 72
8.4.1 Possible validation types 73
8.4.2 Possible operators 74
8.4.3 Example logical operations 74
8.4.4 Example <Message> 75
9 Components 77
9.1 Attributes for components 79
9.2 Detail Component 82
9.2.1 Defining a Detail Component 82
9.2.2 Example for a Detail Component 83
9.3 List component 84
9.3.1 Defining List component 85
9.3.2 Column definition 87
9.3.3 Periodic reloading 90
9.3.4 Mobile view of the list component 91
9.3.5 "Select" Action within a List component 92
9.3.6 Example list component 92
9.3.7 Paging 93
9.3.7.1 Sample request 94
9.3.7.2 Sample response 94
9.3.7.3 Example for paging 95
9.4 Calendar Component 98
9.4.1 Defining a Calendar Component 98
9.4.2 Calendar settings: Views 99
9.4.3 Calendar settings: Localization 100
9.4.4 Events 101
9.4.5 External Links in Events 105
9.4.6 Data for the calendar component 105

9.4.7 Example 106
9.5 Master/Detail component 108
9.5.1 Properties in Master/Detail 109
9.5.2 Defining Master/Detail component 110
9.5.3 Hide and display components 111
9.5.4 Master Structural Element 112
9.5.5 Detail Structural Element 112
9.5.6 Overlay Structural Element 113
9.6 ExternalWeb component 115
9.6.1 Example 117
9.6.1.1 URL in ExternalWeb 117
9.6.1.2 URL via Technical Process 117
9.7 Grid component 118
9.7.1 Defining Grid component 118
9.7.2 Sample of a grid component 126
9.7.3 Grid settings 126
9.7.4 GridColumn definition 129
9.7.5 Periodical reloading 130
9.7.6 "Select" action within the grid component 131
9.7.7 "Save" action within the grid component 131
9.7.8 Paging 131
9.7.8.1 Client-side paging 131
9.7.8.2 Server-side paging 132
10 Layouts 134
10.1 Box Layout 136
10.1.1 Examples 136
10.1.1.1 BoxLayout in combination with FlowLayout 136
10.1.2 <BoxLayout> 137
10.2 Flow Layout 142
10.2.1 Examples 143
10.2.2 <FlowLayout> 143

10.3 Grid Layout 147
10.3.1 Examples 148
10.3.2 <GridLayout> 148
10.3.3 <ColumnDefinitions> 152
10.3.4 <Column> 153
10.4 Tab Group 154
10.4.1 Examples 155
10.4.1.1 Desktop view 156
10.4.2 <TabGroup> 156
10.5 Tab layout 159
10.5.1 Examples 160
10.5.2 <TabLayout> 160
10.5.3 <Tab> 164
10.6 Responsive Layout 168
10.6.1 Examples 169
10.6.2 <ResponsiveLayout> 170
10.6.3 <Cell> 174
11 Controls 179
11.1 Button 179
11.1.1 Example 180
11.2 Charts 180
11.2.1 Line Chart 181
11.2.1.1 General Attributes 181
11.2.1.2 Line 184
11.2.1.3 Axis 185
11.2.1.4 LineData 186
11.2.1.5 Tooltip 187
11.2.1.6 Examples 188
11.2.2 Bar Chart 190
11.2.2.1 General Attributes 190
11.2.2.2 Bar 193

11.2.2.3 BarData 194
11.2.2.4 Axis 194
11.2.2.5 Tooltip 195
11.2.2.6 Empty data 195
11.2.2.7 Examples 197
11.2.3 Pie/Donut Chart 200
11.2.3.1 General Attributes 200
11.2.3.2 Pie 203
11.2.3.3 Tooltip 206
11.2.3.4 Example 206
11.2.4 Gauge Chart 207
11.2.4.1 General Attributes 208
11.2.4.2 Gauge 210
11.2.4.3 Tooltip 212
11.2.4.4 Examples 212
11.2.4.5 Functionality and further examples for gauge charts 215
11.3 Checkbox 216
11.3.1 "Select" Action within a Checkbox 221
11.3.2 Example for <Checkbox> 221
11.3.3 Example for <Checkbox> with a Technical Process 222
11.4 ComboBox 222
11.4.1 Examples 229
11.4.1.1 Example <ComboBox> 229
11.4.1.2 Example <ComboBox> with outsourced pick list 230
11.4.1.3 Example <ComboBox> with icon 231
11.5 ComboBoxAutocomplete 234
11.5.1 Sample 240
11.6 DateTimePicker 242
11.7 File link 246
11.7.1 Example <FileLink> (File download from a static resource) 251
11.8 File Upload 252

11.8.1 Input format for the Technical Process ... 257
11.8.2 <FileUpload> example ... 258
11.9 Header ... 258
11.9.1 Example <Header> ... 262
11.10 HTML Document ... 263
11.10.1 Example for <HtmlDocument> ... 264
11.11 Image ... 267
11.11.1 "Select" Action within an image control ... 270
11.11.2 Example <Image> from static resource ... 270
11.11.3 Example <Image> with data binding (dynamically from a Technical Process) ... 270
11.11.3.1 Images in Base64 format ... 270
11.11.3.2 Images from an external URL ... 271
11.12 Card ... 271
11.12.1 Create Card Control ... 271
11.12.2 Actions ... 276
11.12.3 Examples ... 276
11.13 Label ... 278
11.13.1 Example <Label> ... 282
11.14 Link ... 283
11.14.1 Example ... 284
11.15 ListView ... 284
11.15.1 "Select" Action within a ListView ... 289
11.15.2 Examples ... 289
11.15.2.1 List without actions ... 289
11.15.2.2 List with actions ... 291
11.16 Maps ... 292
11.16.1 Create map control ... 292
11.16.2 Search ... 295
11.16.3 Routing ... 296
11.16.4 Single marker ... 297
11.16.5 Marker set ... 299

11.16.6 "Select" Action within a Map 302
11.16.7 Examples 302
11.16.8 Set marker by click in the map 307
11.16.9 Using the user's location 308
11.17 PasswordBox 308
11.18 Radio button 313
11.18.1 "Select" Action within a radio button 317
11.18.2 Example <RadioButton> 317
11.19 TextBlock 318
11.19.1 Paragraph 318
11.19.2 Span 322
11.19.3 Break 326
11.19.4 Binding 326
11.19.5 Translation 327
11.19.6 Link 327
11.20 TextBox 328
11.20.1 Example <TextBox> 334
11.21 Video 335
11.21.1 Data Binding for value 338
12 Actions 339
12.1 Action "New" 344
12.2 Action "Save" 345
12.3 Action "Delete" 345
12.4 Action "Upload" 346
12.5 Action "Download" 348
12.6 Custom Action 349
12.7 Action "Reload" 351
12.8 Action "Cancel" 352
12.9 Action "Select" 353
12.9.1 Example 354
12.10 Action "Forgot password" 354

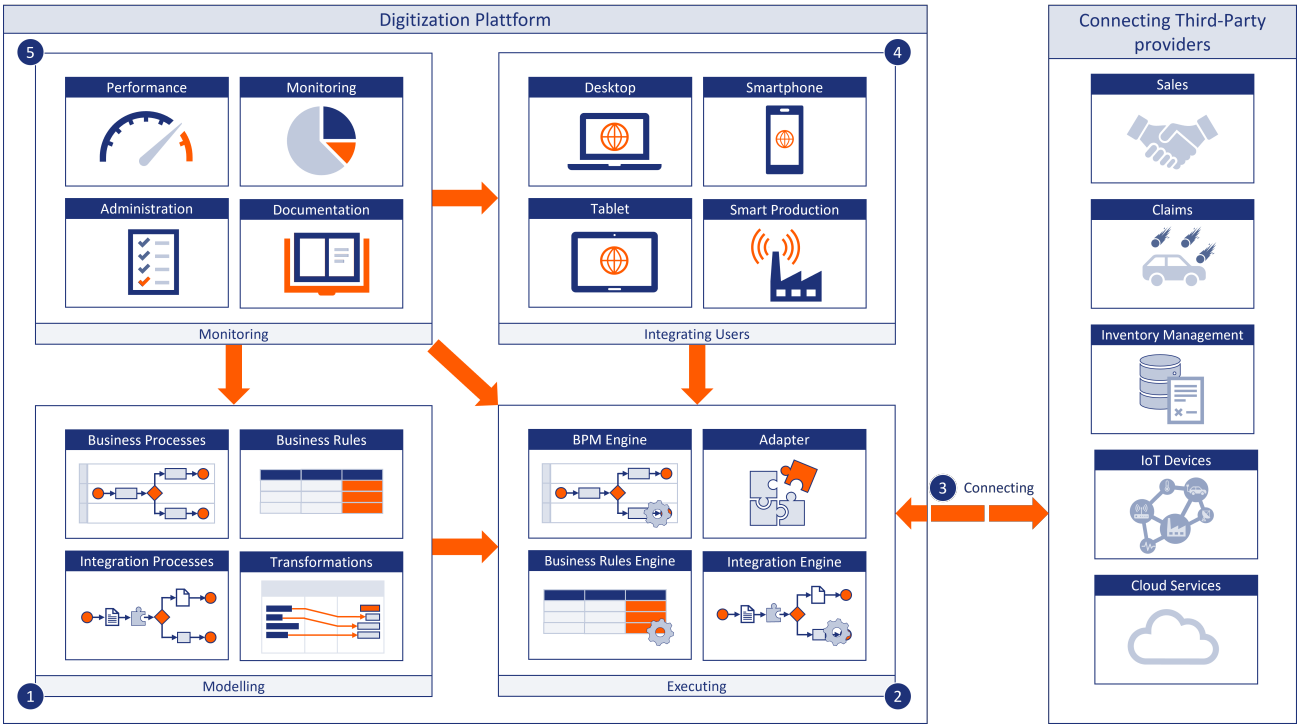
12.10.1 Example 355
12.11 Action "LogoutAction" 355
12.11.1 Example 355
13 HeaderSection 356
13.1 Create HeaderSection 356
13.1.1 <HeaderSection> 356
14 Integrating JavaScript in Web Apps 357
14.1 Working with Web App Control Projects 357
14.1.1 Creating a Web App Control Project 357
14.1.2 Using Custom Controls in a Web App 358
14.1.3 Available API Functions 360
15 Data Binding 363
16 Navigation within a Web Application 365
16.1 Navigation with an Action 365
16.2 Navigation by overwriting the select action 365
17 Sorting 367
18 Messages 368
19 Profiling 370
19.1 Create profiling file 370
19.2 Configure profiling 370
19.3 Example of profiling configuration file 371

About the X4 BPMS

Digitalization requires a holistic approach, which presupposes that also the used solution has to reflect that. X4 BPMS supports you as a central platform in solving these challenges. The focus is on modeling, implementing and monitoring your business processes. Therefore, the X4 BPMS contains all necessary tools and is compatible with a variety of interfaces and formats. That helps to avoid isolated information silos and media breaks that inhibit productivity, and accelerate digitization at the same time.

Implementing business processes without programming effort enables a large number of users to enter into the management of business processes. That's important, since employees of the specialist department usually know best what is important in the respective business processes. Therefore, you should rely on the X4 BPMS as a platform whose tools reduce complexity to such an extent that business processes can be analyzed, optimized, modeled, as well as controlled and documented even without programming knowledge. All tools support integrated, graphical process modeling and implementation and generate processes that are executed by the X4 BPMS with high performance.

- **X4 Designer:** Modelling processes and rules graphically
- **X4 Server:** Simulating and executing processes and rules
- **X4 Adapter:** Integrating third-party systems into processes
- **X4 Activities**: Providing web apps for employees and customers



With the X4 BPMS component X4 Web Apps, a web application can be created without programming. Processes can be directly integrated into these web applications and data can be processed in the

background. The web applications are optimized for display and use on smartphones, tablets and classic desktop computers and can be displayed in multiple languages using language files.

To create a web application, the desired page structure is described in XML. This structure is automatically implemented by X4 BPMS as a web application.

Features of X4 Web Apps:

- Central app definition completely in X4 Designer
- Easy integration of Technical Processes in X4 Designer
- Dynamic web application that can be extended on the fly
- Login page available by default
- Responsive display "out of the box" for smartphones, tablets and desktop browsers
- Customizable interface design (colors, logo, icons, etc.)
- Usable with all major desktop browsers and mobile platforms

The X4 Web Apps component supports the following browsers:

- Google Chrome (Desktop/Mobile)
- Apple Safari (Desktop/Mobile)
- Mozilla Firefox (Desktop/Mobile)
- Microsoft Edge (Desktop/Mobile)

X4 Designer makes it possible to create Web App projects with an already pre-configured project structure and give them a modern look using the integrated Theme Editor.

The existing business logic and static resources can be easily integrated into the web applications. Additionally legal matters such as an imprint and information on data protection can be stored.

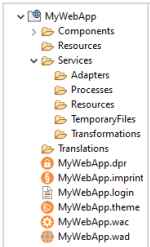
Who is the target group of this document?

This documentation is for developers who want to implement web applications based on web technologies in combination with X4 ESB and its web service interface in a very short time. In addition to detailed technical knowledge of the data and workflows, common web technologies such as HTML, CSS and JavasScript, basic knowledge of XML as well as knowledge of XSLT and XPath are required.

1 Structure of a Web App project

1.1 Back End

When a Web App project is created, the following project structure is created in X4 Designer:



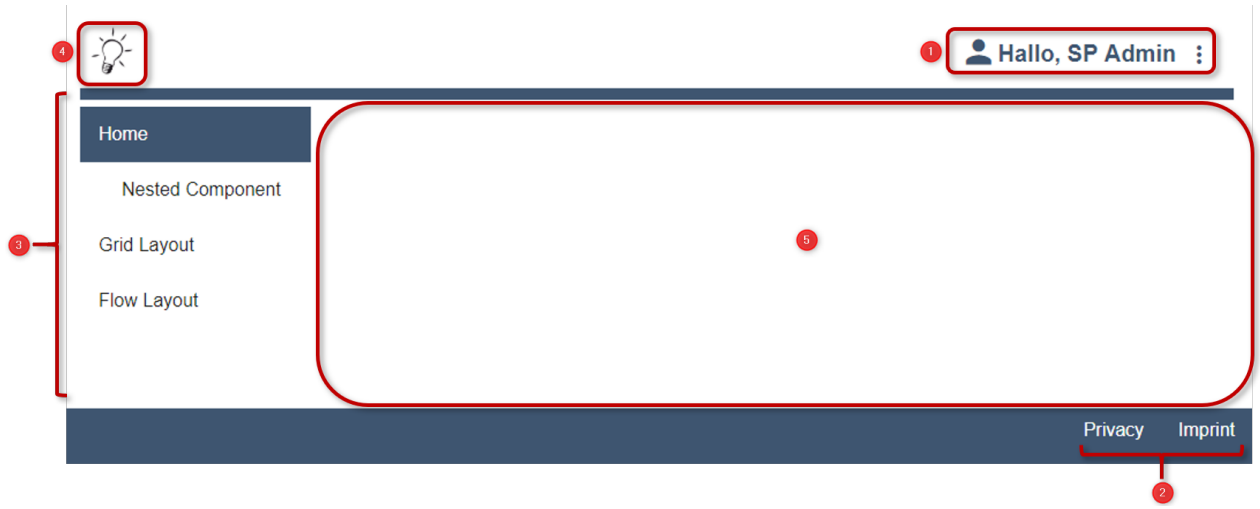
- *Projekt* Project Web App project
- `Components`: Separate definitions for the individual components. The folder may only contain definitions for detail components, list components, grid components, calendar components and master/detail components.
- `Resources`: All resources of the web application, e.g. images. May contain subfolders. The contents of this folder can be accessed via the Internet.
- `Services`:
  - `Adapters`: Adapters used in the project. Can contain subfolders.
  - `Processes`: Technical Processes. May contain subfolders.
  - `Resources`: All resources used in technical processes. Can contain subfolders. Can contain text, XML, XHTML, SQL, and binary files.
  - `TemporaryFiles`: Temporary files. Can contain subfolders. Can contain text and XML files.
  - `Transformations`: Mappings for processes. Can contain subfolders. Can contain XSLT and REP files.
- `Translations`: Contains the individual language files for multilingual interfaces
- *Project*`.dpr`: Data Protection Information, either as text or linking to an external page.
- `Project.headersection`: Definition of custom header used instead of the standard header.
- *Project*`.imprint`: Imprint Information, either as text or linking to an external page.
- *Project*`.theme`: Theme of the application, can be defined using a theme designer
- *Projekt*`.wad`: File with the Web App Definition
- `Projekt.wac`: File with the Web App Configuration

The `Resources` , `Services` and `Translations` folders do not contain any additional files when a project is created. The files for the data protection information and for the imprint also initially have no content and can be deleted if they are not needed in the project.

> **Security note**
> Note that all file resources within the `Resources` folder are served via HTTP(S) relative to the path of the web application!

1.2 Front End

The structure of the front end of a Web App is as follows:



1	Welcome text and user menu
2	Automatically generated links to the data protection and imprint
3	Automatically generated menu of the Web App. Is formed from the display names of the components.
4	Web App logo
5	Area for the content of the component. The display depends on the type of component, the layout used and the controls used.

2 Creating and Declaring a New Web App Project

This section explains how to create and declare a new Web App project.A Web App project (*Web App Project*) is a project type within the X4 BPMS. In addition to a predefined folder structure, it contains the central definition file `<WebApp>.wad` and is automatically registered as a web application in the X4 server.

If errors occur during creating and declaring a web application, these are not only displayed in the server log, but also directly in the browser.

2.1 Templates

In X4 Designer, you can choose from several templates when creating a project.

2.1.1 ESB Project

Category	Template	Description
Basics	ESB Introduction	This project contains a brief introduction to Technical Process modeling.
	XSLT Introduction	This project contains a brief introduction to creating XML data transformations.
Database	Create Data	This project contains an example of creating data in a database using the JDBC Connector adapter.
	Delete Data	This project contains an example of deleting data from a database using the JDBC Connector adapter.
	Read Data	This project contains an example of reading data from a database using the JDBC Connector adapter.
	Update Data	This project contains an example of updating data in a database using the JDBC Connector adapter.
E-Mail	Send e-mails with the POP/SMTP adapter	This project contains an example of sending e-mail using the POP/SMTP adapter.

2.1.2 BPM Project

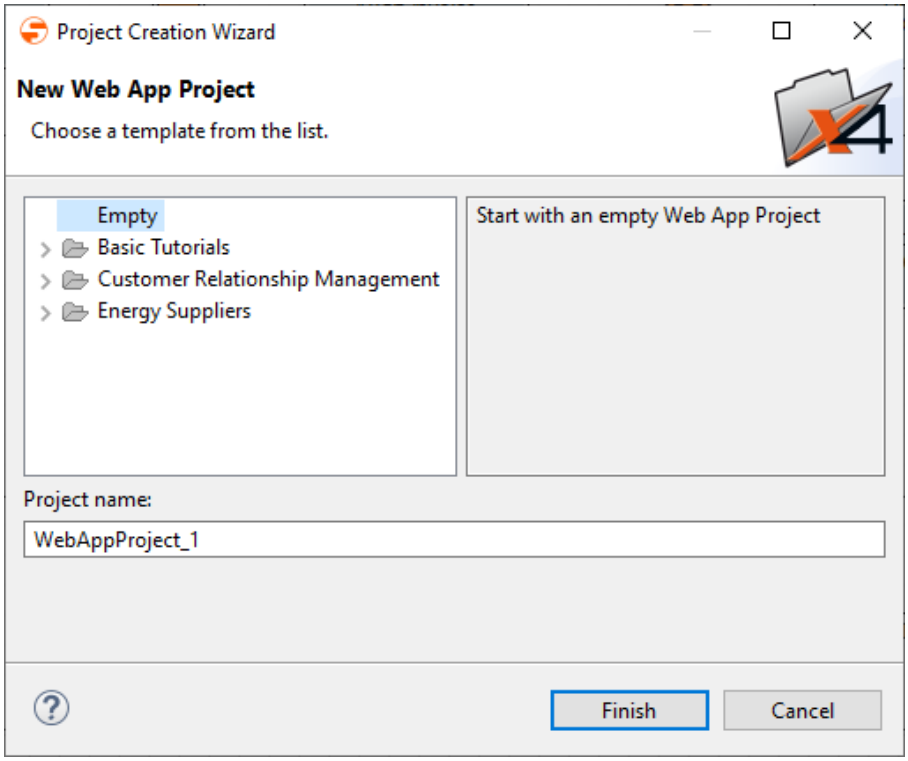
Category	Template	Description
Basics	BPMN Introduction	This project contains a brief introduction to Business Process modeling.
Demos	Approve electronic invoice	This demo includes an approval process for electronic invoices. It includes Business Processes, Business Rules (DMN) and Human Tasks.

2.1.3 Web App Project

Category	Template	Description
Components	Calendar Component	This Web App is an example of a Calendar Component with appointments.
	Grid Component	This Web App is an example of displaying data in tabular form in the Grid Component.
Controls	How to use Controls	This project contains examples of how to use the most common Controls in X4 Web Apps.
	TextBlock	This project contains examples of how to use the TextBlock control in X4 Web Apps.
Demos	Manage your customer data	In this demo you can manage your customers, their address data as well as contact persons.
	Report a street lighting malfunction	In this demo, a customer can report a malfunction in the street lighting system. The data is stored in a database.
Layouts	How to use layouts	This project contains examples of using layouts in X4 Web Apps.
Properties	Data Context	This project contains an example of how to use properties of the same context in several components.

2.2 Creating a New Web App Project

1. In the *X4 Designer* menu, select **File > New > WebApp Project** to open the **Project Creation Wizard**.



Alternatively, right-click in the repository > **New** > **Web App Project**.

2. Enter the name of the project in **Project name**.
3. Click **Finish**.

A new Web App project with a predefined structure is created. The Web App project is automatically registered as a Web application in the X4 server.

2.3 Display Web App project

The front end of the web application can be displayed at any time, even during editing, as well as in the browser or on a mobile device.

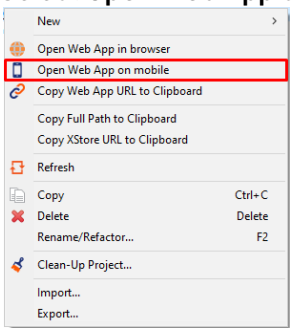
Certain changes within the Web App project are only displayed after a new login. This includes:

- Colors
- Language
- Menu

For further information please visit the section

2.3.1 Open on mobile devices

1. Right-click on the Web App project.
2. Select **Open Web App on mobile** in the context menu.



A QR code is generated and displayed.

3. Scan QR code with mobile device.
4. Log in with your credentials.

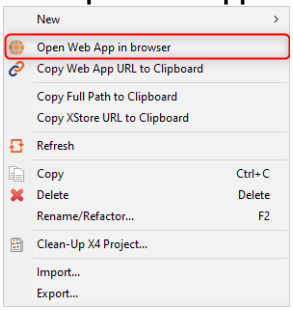
> ⚠ When the access to the Web App is not restricted, any user can access the Web App.

> ✓ Spaces that are inserted at the end of the user name, e.g. by an auto-complete when used on the smartphone, are automatically removed.

> ✓ For further information please visit the section

2.3.2 Open in browser

1. Right-click on the Web App project.
2. Click **Open Web App in browser** in the context menu.



The Web App is opened in the default browser.

3. Log in with your credentials.

> ⚠ When the access to the Web App is not restricted, any user can access the Web App.

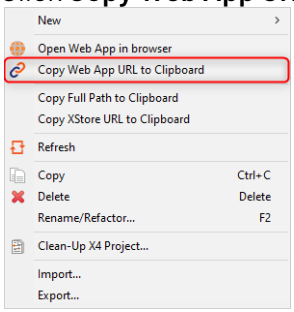
Spaces that are inserted at the end of the user name, e.g. by an auto-complete when used on the smartphone, are automatically removed.

For further information please visit the section

2.3.3 Copying URL of the Web App

If you want to open the web app in a browser other than the default browser, or if you want to send the URL, for example, then you can also copy the URL of the web app directly to the clipboard.

1. Right-click on the Web App project.
2. Click **Copy Web App URL to Clipboard** in the context menu.



For further information please visit the section

2.4 Declaring a Web App project

Web applications are structured hierarchically:

- Web application
- Module(s)
- Component(s)
- Layout(s)
- Control(s)

The web application and the modules available in the application with their respective access rights are declared in the `.wad` Web App definition. The individual components with their respective layout and available controls are described within the respective component and referenced in the Web App definition.

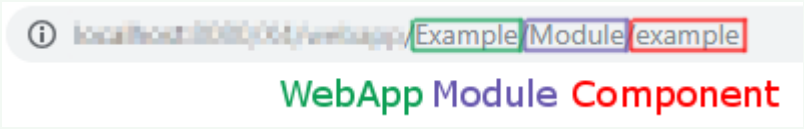
When declaring `.wad` files, the file is automatically validated. Thus, possible sources of errors can be reduced already during the creation of web applications.

The root element of the `.wad` file is `<WebApp>`. The element contains all other elements of the Web App. Attributes control the appearance and behavior of the Web App.

Possible attributes

Attribute	Description
`path`	*Required*. Path to the Web App. Is displayed in the address bar of the browser. Must be unique for each Web App project. **Possible values:** String consisting of alphanumeric characters (no umlauts, dots, etc.) ⓘ The path is part of the URL that is used to access the Web App via the browser. The name of the Web App is used as the default value for the `path` attribute. However, the path can contain any string consisting of alphanumeric characters. If the Web App project is renamed, the `path` attribute **is not adjusted** and the Web App can still be accessed using the previous URL.

✓ The path specifications for the project, the modules and the components (`path` attribute) can be used to navigate within the web application.



2.4.1 Referencing Components in the Web App Definition

Each individual Component used in the Web App is declared in a separate definition. These individual definitions are then referenced in the Web App Definition via the `<ComponentReference>` element.

Within the `<ComponentReference>` further `<Components>` with their respective references (`<ComponentReference>`) can be contained. This way a hierarchical navigation can be implemented.

The `<ComponentReference>` element has the following attributes:

Attribute	Description
`default`	Required for exactly one component. Defines which component is displayed first ("Home"). **Possible values:** *`true`* / *`false`*
`displayName`	Display name of the component in the menu. **Possible values:** Any string with the display name of the component

Attribute	Description
`fontFamily`	Defines the font family within the component. The property is inherited by all controls and actions of the component. **Possible values:** Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size within the component. The property is inherited by all controls and actions of the component. **Possible values:** • Font size in pixels, e.g. *`20px`* • Font size in point, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Key words, e.g. *`small`* or *`larger`*
`fontStretch`	Defines the width of each character. The property is inherited by all controls and actions of the component. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the slant of the font. The property is inherited by all controls and actions of the component. **Possible values:** • `italic`: italic font • `normal`: normal font (default) • `oblique`: oblique font style (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. The property is inherited by all controls and actions of the component. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`iconUrl`	Path to the image file or specification of a Material Icon that will be used as the component's icon. ⓘ • The image file must be contained in the `Resources` folder directly below the Web App project. Path specification relative to the `Resources` folder. • The material icon must be specified with the prefix `icon`, e.g. `icon:<MaterialIconName>`. **Possible values:** • String (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* ✓ With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons.

Attribute	Description
`iconColor`	Defines the color of the component's icon. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash before the color value or a shortened notation of the color value! • Colour code from the colour palette of the web app (cf. Theming), e.g. *`A200`*
`name`	Unique name of the Component. **Possible values:** Any string with the name of the Component
`path`	*Required*. Unique URL of the component, visible in the browser address bar. **Possible values:** String of alphanumeric characters (no umlauts, dots, etc.)
`showInMenu`	Defines if the component is shown in the menu. ⚠ This attribute has no effect on a Master/Detail Component. **Possible values:** `true` (default) / `false` ⓘ This setting affects also all subcomponents. If no component is displayed in the menu, the entire menu is hidden. The content area then extends across the entire width.
`source`	Path to the definition file relative to the `Components` folder. **Possible values:** String (URI)
`title`	Title of the component. Displayed as header in the component. • Data Binding possible **Possible values:** Any string with the title of the component

Attribute	Description
`titleBackground`	Defines a title background color. **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` ⚠ Do not use a hash before the colour value and do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`
`titleForeground`	Defines a color for the title foreground. **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` ⚠ Do not use a hash before the colour value and do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`

Example

Definition of the Simple CRM integrated sample project

```
<?xml version="1.0" encoding="UTF-8"?>
<WebApp xmlns="http://softproject.de/webapp/1.0"
        path="SimpleCRM">
    <Translations>
        <Translation displayName="Deutsch" name="de"/>
        <Translation displayName="English" name="en"/>
    </Translations>
    <Modules>
        <Module displayName="My Module" path="Module">
            <Components>
                <ComponentReference default="true" displayName="$Customers"
                    iconUrl="logo.png" name="Home"
                    path="Home" source="Customer/Customer.masterdetail"/>
                <ComponentReference displayName="Statistik" iconUrl="Statistic.png"
                    path="Statistic" source="Statistic/Statistic.detail"/>
                <ComponentReference displayName="$Administration" iconUrl="Gear.png"
                    path="Administration" source="Administration/
Administration.detail">
                  <Components>
                      <ComponentReference displayName="$Country" iconUrl="Country.p
ng"
                          name="Countries" path="Country"
                          source="Administration/Country/Country.masterdetail"/>
                      <ComponentReference displayName="$Category" iconUrl="Category
.png"
                          name="Categories" path="Category"
                          source="Administration/Category/Category.masterdetail"/>
                      <ComponentReference displayName="$LegalForm" iconUrl="LegalFo
rm.png"
                          name="LegalForm" path="LegalForm"
                          source="Administration/LegalForm/LegalForm.masterdetail"
/>
                      <ComponentReference displayName="$Event" iconUrl="Event.png"
                          name="Event" path="Event"
                          source="Administration/Event/Event.masterdetail"/>
                  </Components>
                </ComponentReference>
            </Components>
        </Module>
     </Modules>
</WebApp>
```

2.5 Configuring the Menu Width

The width of a Web application's menu can be defined in the `.wad` file using the attributes of the `Menu` element. The `Menu` element is a direct child element of the `WebApp` root element.

Possible Attributes

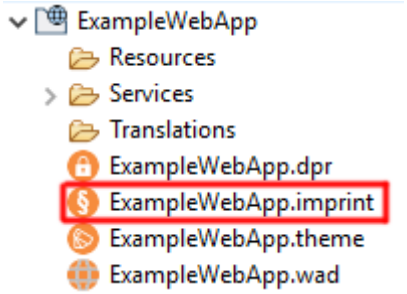
Attribute	Description
`unit`	Unit for the value specified in attribute `width`. **Possible values:** • `pixels`: Specification in pixels • `percents`: Specification in percent ⓘ If no unit is defined, the specified value for `width` is interpreted as pixel
`width`	Menu width **Possible values:** • Any integer • `auto`: Automatic width, which is automatically calculated based on the defined menu items

Sample configuration for the menu width

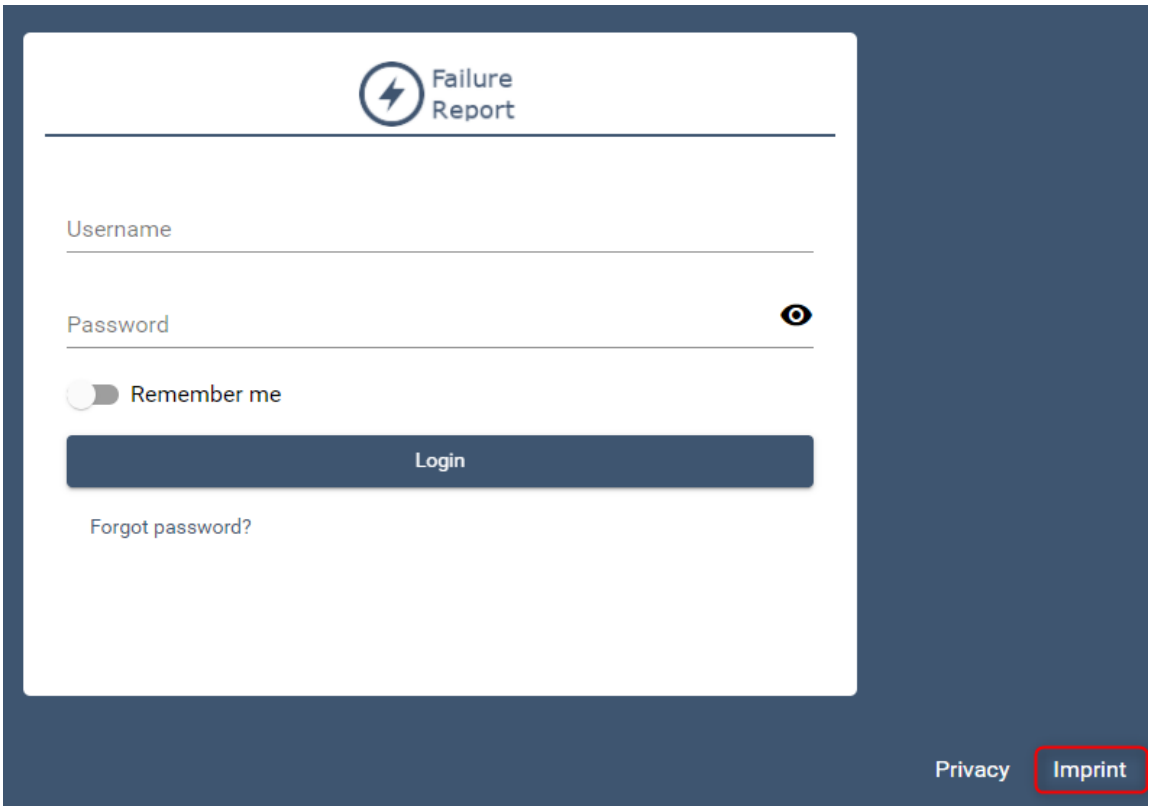
```
<WebApp xmlns="http://softproject.de/webapp/1.0">
	...
	<Menu width="30" unit="percents"/>
	<Modules>
		...
	</Modules>
</WebApp>
```

2.6 Imprint

The imprint of a Web application is automatically generated. It does not initially contain any text and can therefore be edited, manually deleted and recreated. The file that contains the imprint is always named after the Web App project and has the file extension `.imprint`. Only one `.imprint` file can be created within a Web App project.



A link to the imprint is displayed on the login page and on all pages within the web application. The link is only displayed if there is an `.imprint` file within the Web App project.



> ⚠ It is also possible to include external pages as imprint. Define the linking in the Web App Configuration `.wac`.

2.6.1 Creating an Imprint

> ⚠ The value of the `title` attribute is also displayed in the footer area of the Web App.

1. Right-click on the Web App project.
   The context menu opens.
2. Click on **New** > **Imprint**.
   The file for the imprint is created.

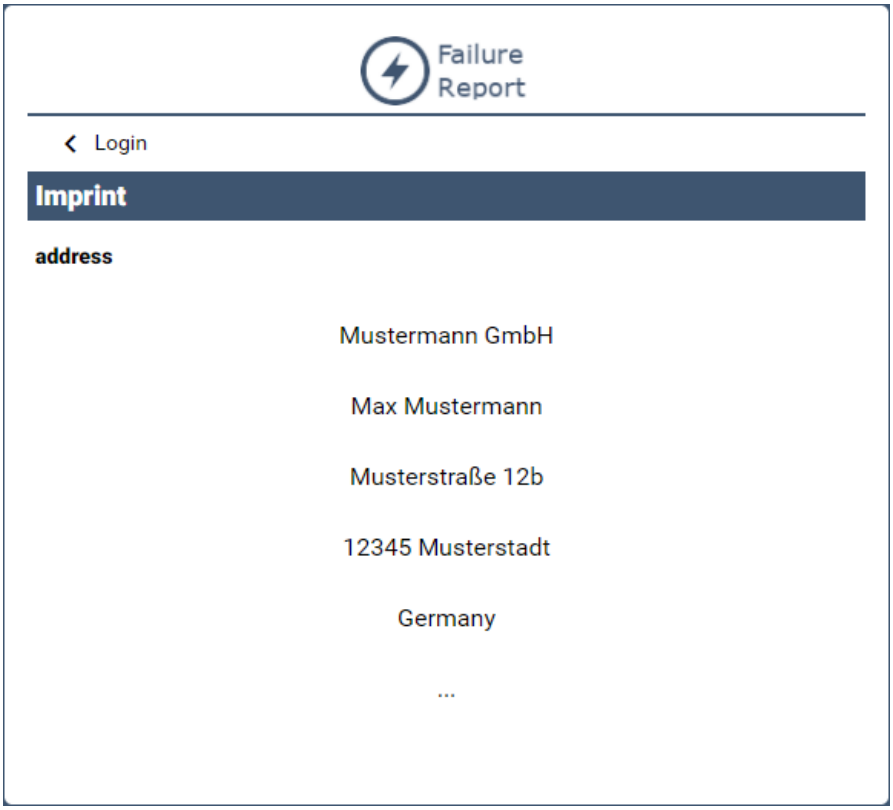
2.6.2 Provice content in the Imprint

The imprint file can contain any layout. Within the layout the content of the imprint can be provided:

Imprint (.imprint)

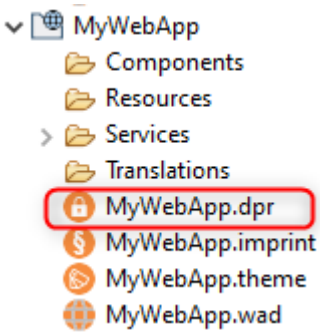
```
<?xml version='1.0' encoding='UTF-8'?>
<Imprint xmlns="http://softproject.de/webapp/1.0" title="Imprint">
    <FlowLayout>
        <Header value="Imprint" />
        <Header titleLevel="subtitle" value="address" />
        <Label value="Mustermann GmbH" />
        <Label value="Max Mustermann" />
        <Label value="Musterstraße 12b" />
        <Label value="12345 Musterstadt" />
        <Label value="Germany" />
        <Label value="..." />
    </FlowLayout>
</Imprint>
```

The above code creates the following imprint:

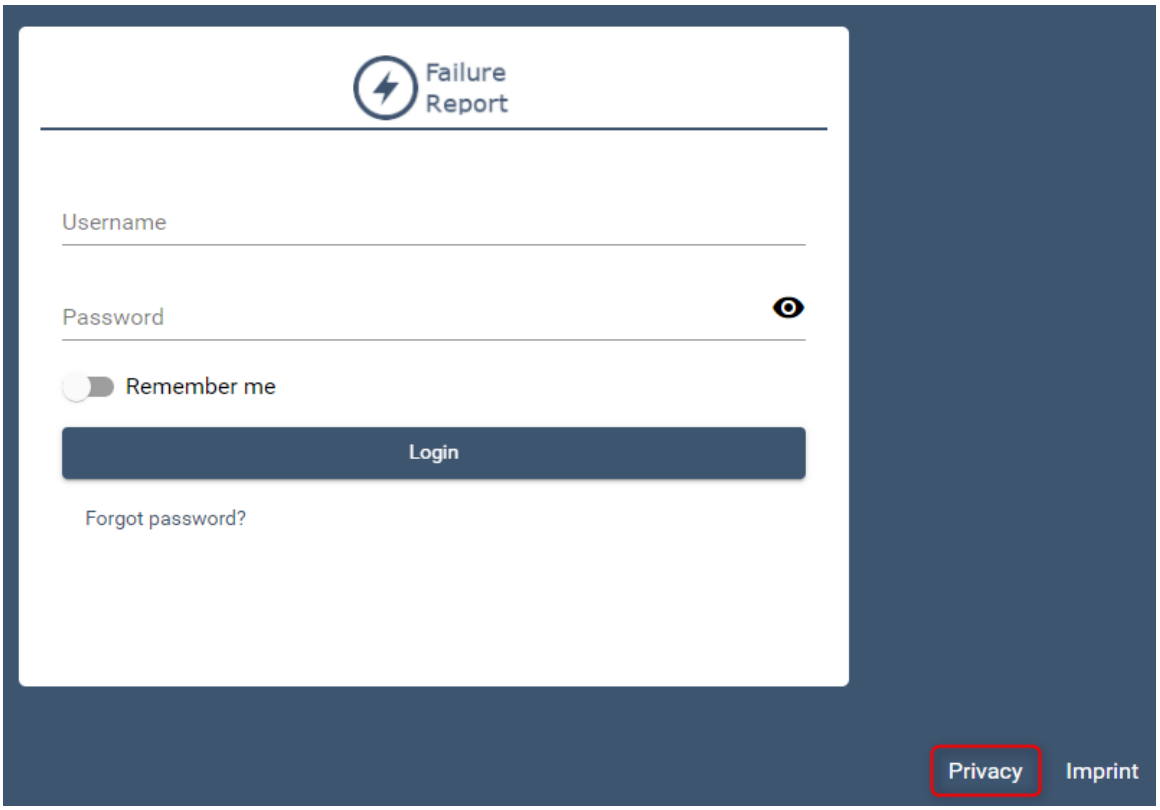


2.7 Data Protection Statement

The data protection statement of a web application that is created as Web App project is generated automatically. It does not initially contain any text and can therefore be edited, manually deleted and recreated. The file that contains the data protection statement is always named after the Web App project and has the file extension `.dpr` (Data Protection Statement). Only one `.dpr` file can be created within a Web App project.



A link to the data protection statement is displayed on the login page and all pages within the web application. The link is only displayed if there is an `.dpr` file within the Web App project.



⚠ It is also possible to include external pages as data protection statement. Define the linking in the Web App Configuration `.wac`.

2.7.1 Creating a file for the Data Protection Statement

⚠ The value of the `title` attribute is also displayed in the footer area of the Web App.

1. Right-click on the Web App project.
   The context menu opens.
2. Click on **New** > **Data Protection Statement**.
   The file for the data protection statement is created.

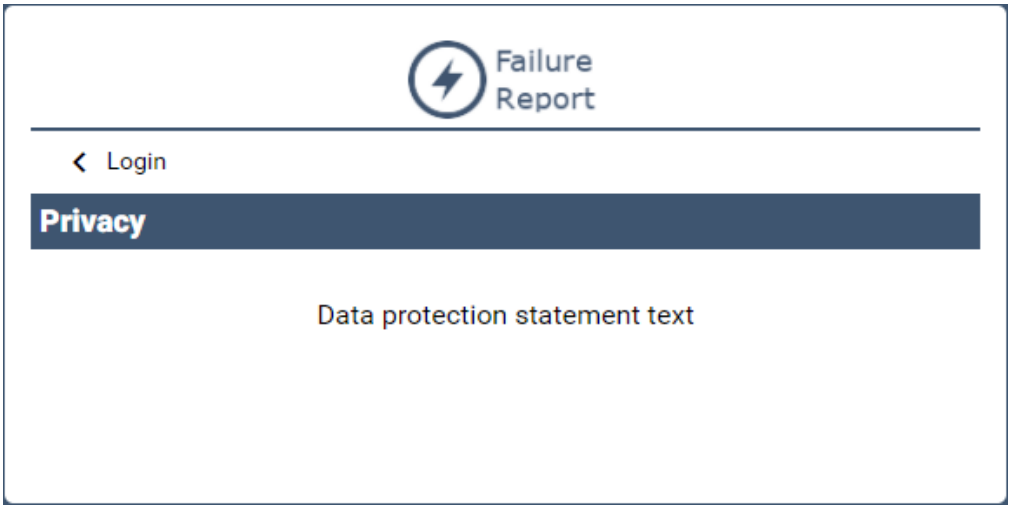
2.7.2 Provide content in the Data Protection Statement

The file for the data protection statement can contain any layout. Within the layout the content of the data protection statement can be provided:

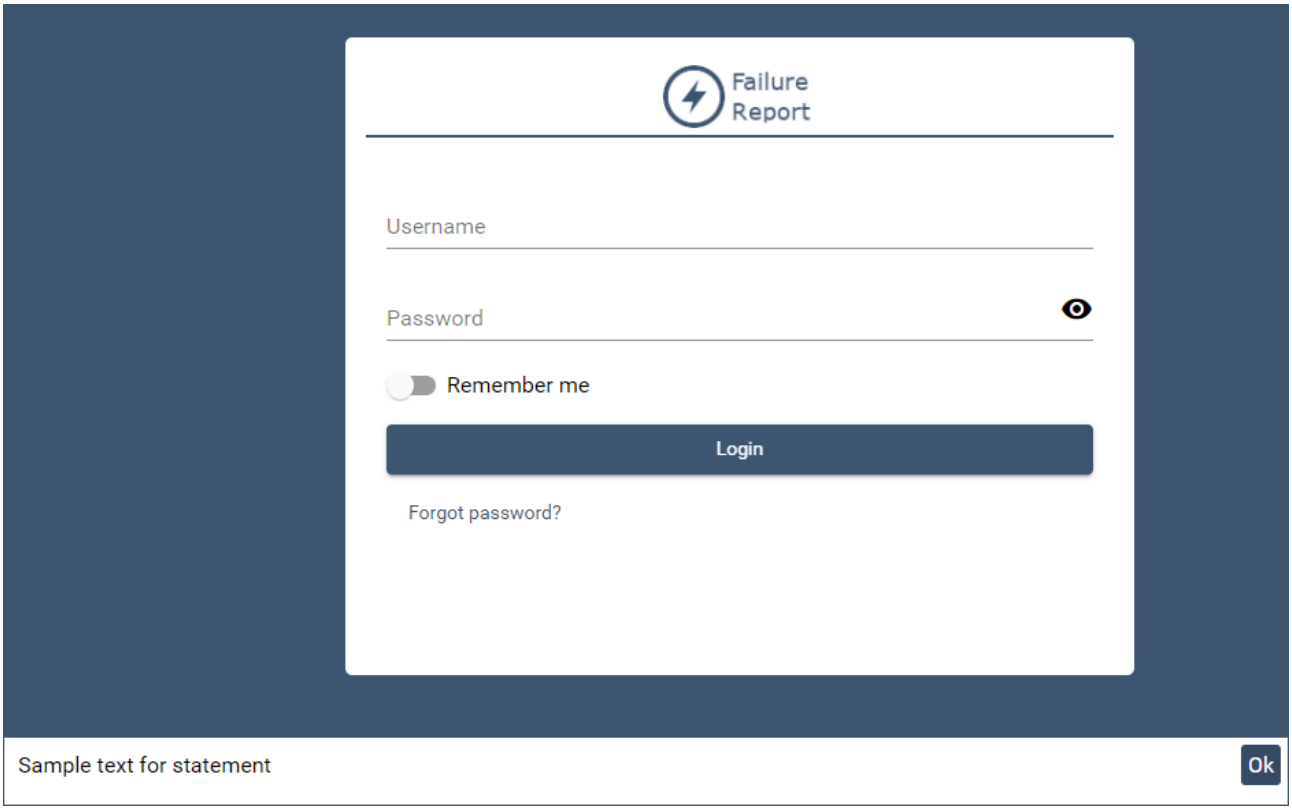
Data Protection Statement (.dpr)

```
<?xml version='1.0' encoding='UTF-8'?>
<DataProtectionStatement xmlns="http://softproject.de/webapp/1.0" title="Privacy">
    <FlowLayout>
        <Header value="Privacy" />
        <Label value="Data protection statement text"/>
    </FlowLayout>
    <Hint text="Sample text for statement" buttonText="Ok"></Hint>
</DataProtectionStatement>
```

The above code creates the following data protection statement:



The element `Hint` is optional. It creates a message box at the bottom of the page. Information on the use of cookies, for example, can be deposited here. It is possible to refer to an external page within the message box with the attribute `linkText`.

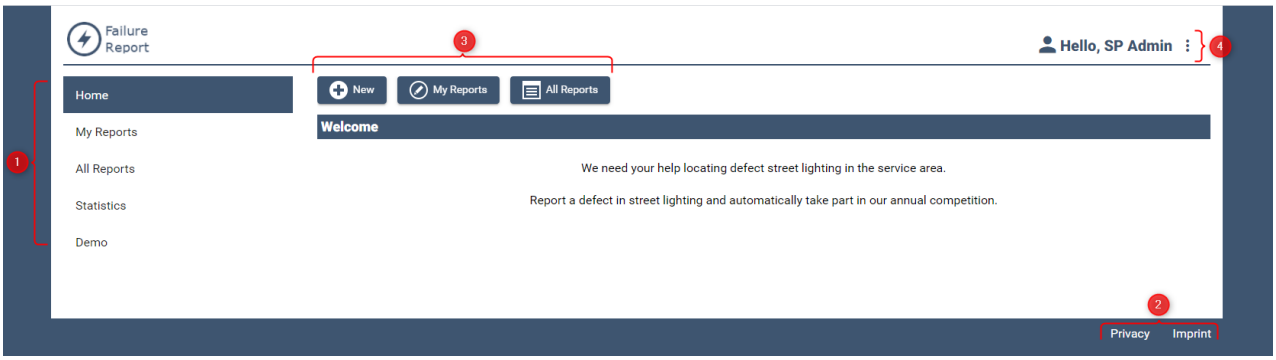


2.8 Responsive Design

Web applications that are created with Web App are optimized for display on smartphones an tablets as well as on classic desktop PCs. The responsive design affects:

- the page layout
- the appearance of checkboxes
- the display of information within a list component / grid component
- the display of master/detail components

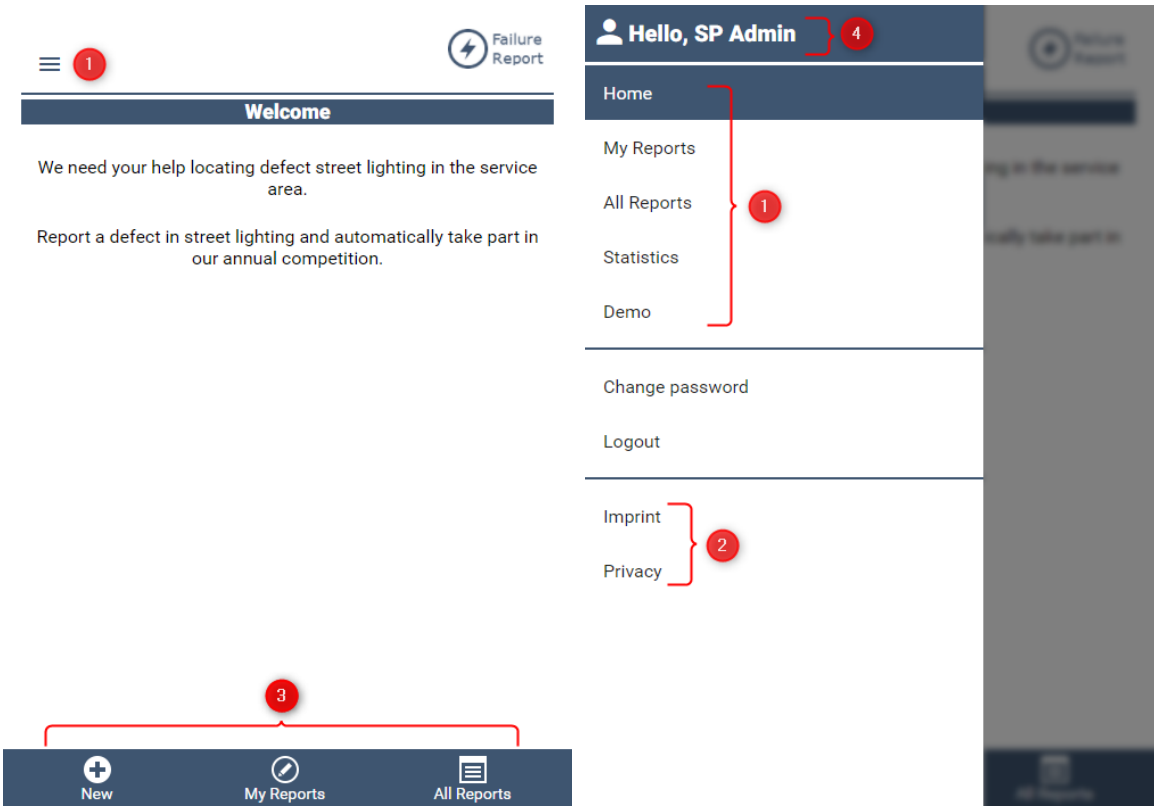
2.8.1 Page Layout on a Desktop PC



1	The menu is created automatically using the components. Each component creates a menu item.

2	Links to the data protection statement and the imprint. Links are generated automatically and refer to the files *projectname*.dpr bzw. *projectname*.imprint . If the files do not exist, no links are displayed here.
3	Buttons for actions
4	Details of the logged in user

2.8.2 Mobile Page Layout

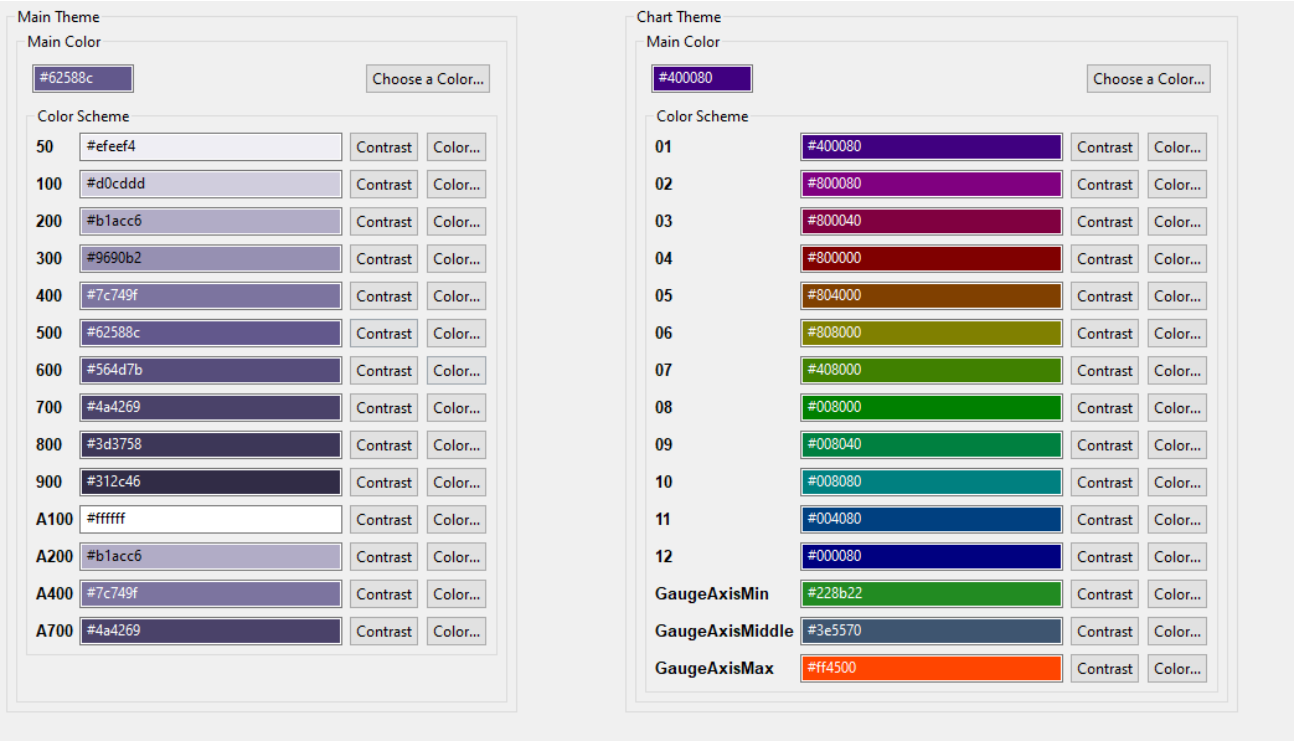


1	Menu, is created automatically using the components . Each component creates a menu item. In mobile mode the menu becomes visible when you click on the menu button.
2	Links to the data protection statement and the imprint. Links are generated automatically and refer to the files *projectname*.dpr respectively *projectname*.imprint. If the files do not exist, no links are displayed here.
3	Buttons for actions
4	Details for the logged-in user

3 Theming

3.1 Defining color scheme

The color scheme and fonts of the web application are defined in the `<project name>.theme` file. Double-clicking the file opens the *Theme Editor*, where the desired settings can be made.



In the *Theme Editor* you can define both the color scheme for the application and the color scheme for charts.

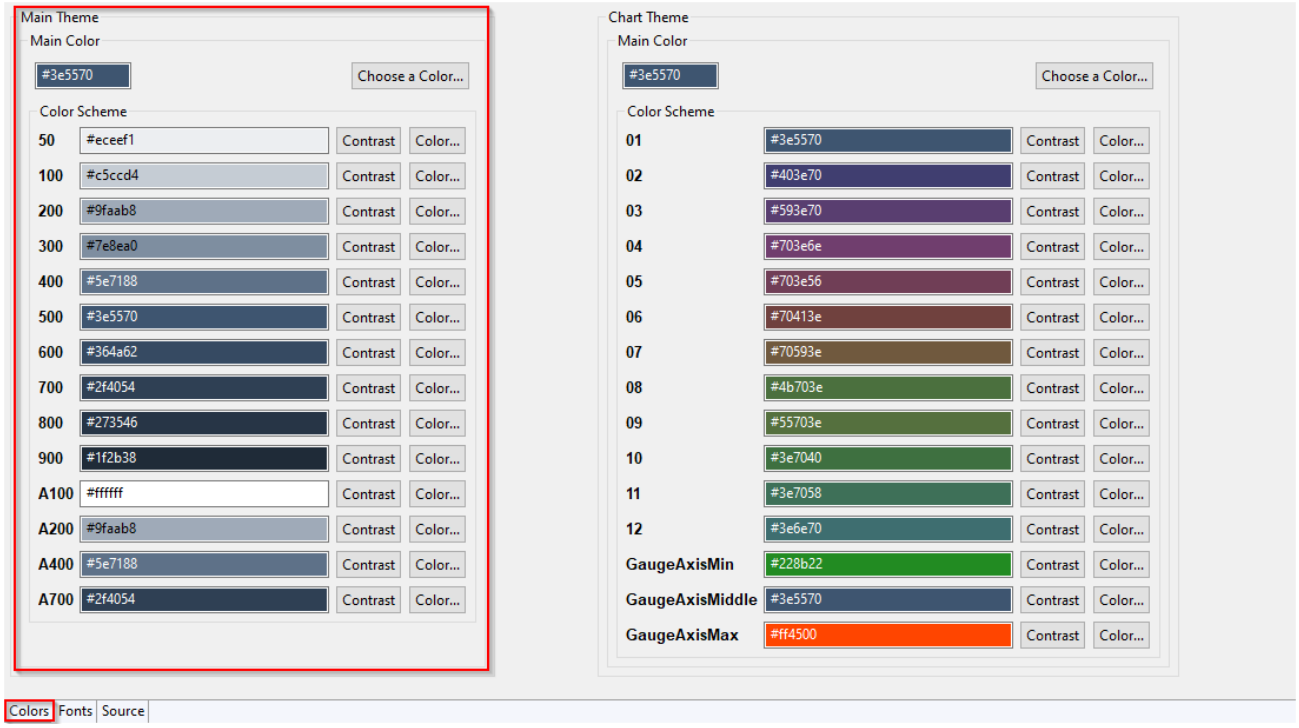
> ⓘ The color values from the color scheme of a Web App can be used to store custom foreground and background colors for Actions or Controls. This can be used to highlight certain elements.

3.2 Colors

Colors are applied in Web Apps using the **Color Scheme**. Each color scheme has a unique name. The color scheme name can be used in attributes to apply the color scheme to an element.

3.2.1 Main Theme

In the Main Theme you can define the colors of the Web App. A main color can be defined by specifying a color in `Main Color`. Based on the Main Color, the remaining colors are calculated automatically. However, the colors can also be defined individually.



In the following table you can see the default usage of the **Color** and **Contrast** defined for the color scheme:

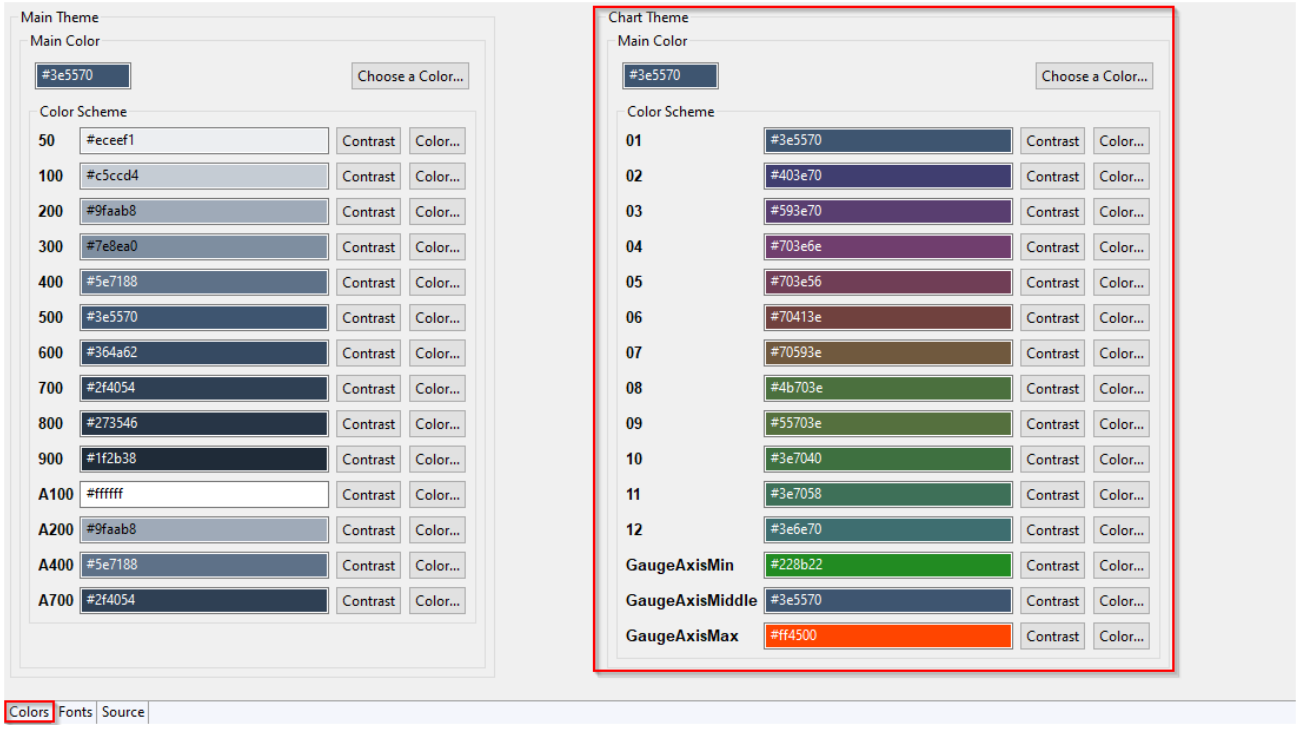
> With **Contrast** you can switch between black and white.

Color Scheme	Color	Contrast
`50`	Zebra table • background of the even row	Zebra table • font color of the even row
`100`	Zebra table • background of the odd row	Zebra table • font color of the odd row
`200`	Zebra table • background on mouse hover	Zebra table • font color on mouse hover
`400`	`Button` • background on mouse hover	`Button` • font color on mouse hover

Color Scheme	Color	Contrast
500	website • Background Header • Background Menu • background of the selected menu item TextBox • labels of the active TextBox • selected date in the date picker Button • Background ComboBox • selected entry in open list Datepicker • selected date Login • Background TabGroup • active tab	website • Font color Header • Font color Menu • font color of the selected menu item Button • Font color
A100	Web App • Background • RadioButton • CheckBox	
A200	Menü: • Background on mouse hover Zebra table • background of the selected row	Zebra table • font color of the selected row
A400	Zebra table • background of the table header Link • Font color	Zebra table • font color of the table header

3.2.2 Chart Theme

In the *Chart Colors* section the colors for the different charts can be defined. By specifying a color for `Main Color`, a main color can be defined. Based on the Main Color, the remaining colors are calculated automatically. However, the colors can also be defined individually.



In the following table you can see the default usage of the **Color** and **Contrast** defined for the color scheme:

> With **Contrast** you can switch between black and white.

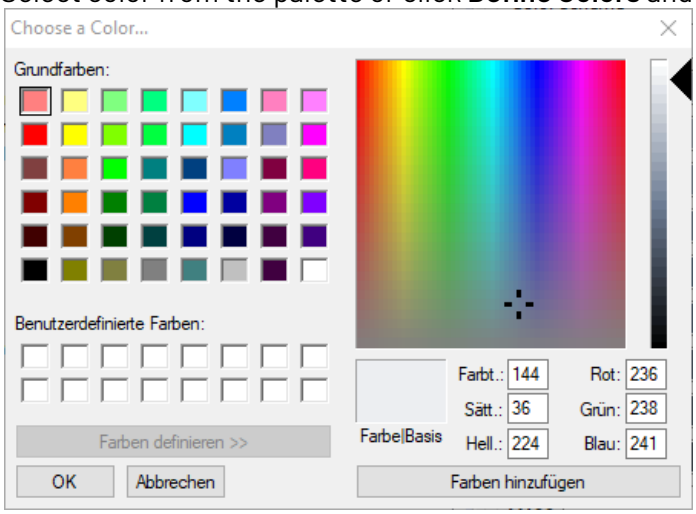
Color Scheme	Color	Contrast

`01 - 12`	• `Line`
	  • line chart
	• `Bar`
	  • bar chart
	• `Pie`
	  • pie/donut chart
		• `Line`
		  • label
		• `Bar`
		  • label
		• `Pie`
		  • label

`GaugeAxisMin`	`Gauge` • start range of the gauge chart	
`GaugeAxisMiddle`	`Gauge` • middle range of the gauge chart	
`GaugeAxisMax`	`Gauge` • end range of the gauge chart	

3.2.3 Defining colors

1. Open the *Theme Editor* by double-clicking the `<project name>.theme` file.
2. Click **Color** next to the color you want to change.
3. Select color from the palette or click **Define Colors** and define a color individually.

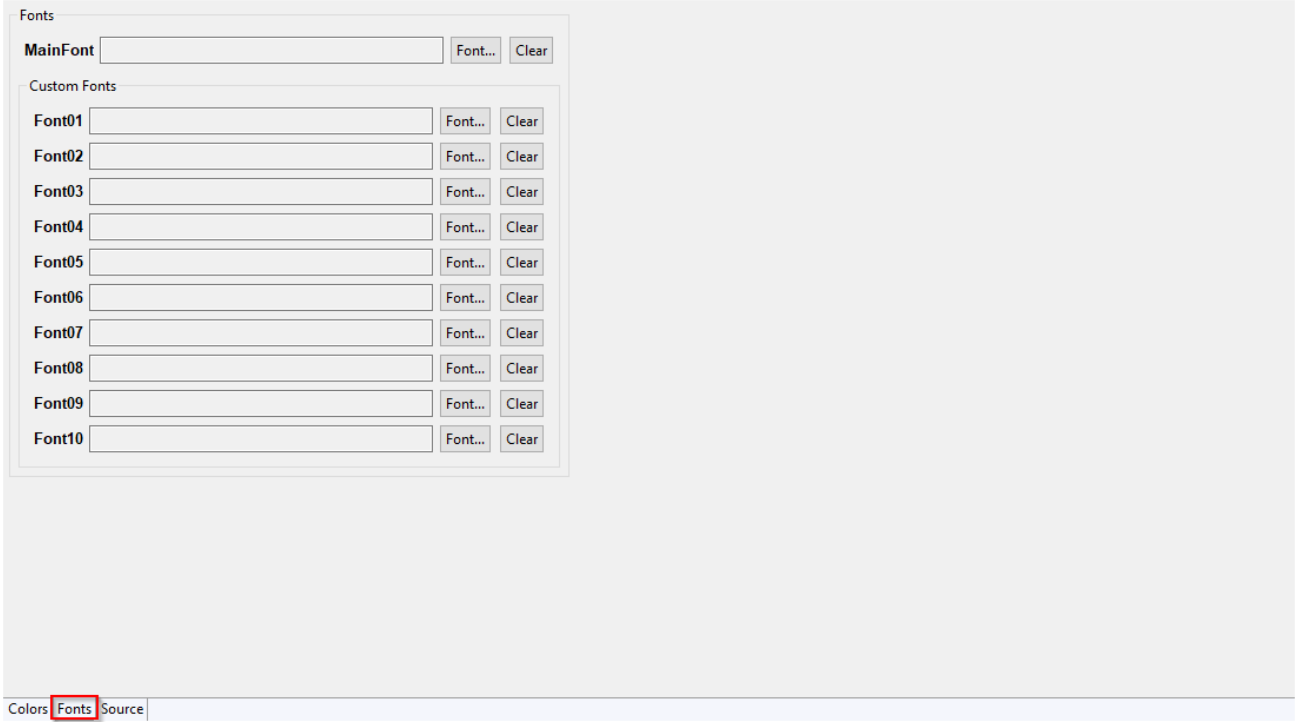


4. Click **Add Color** and add the specified color to the custom colors.
5. Select the color in **Custom Colors** and click **OK**.
   The color has now been applied.

> For documentation on the ReST interface, see http://localhost:8080/X4/X4Api/.

3.3 Fonts

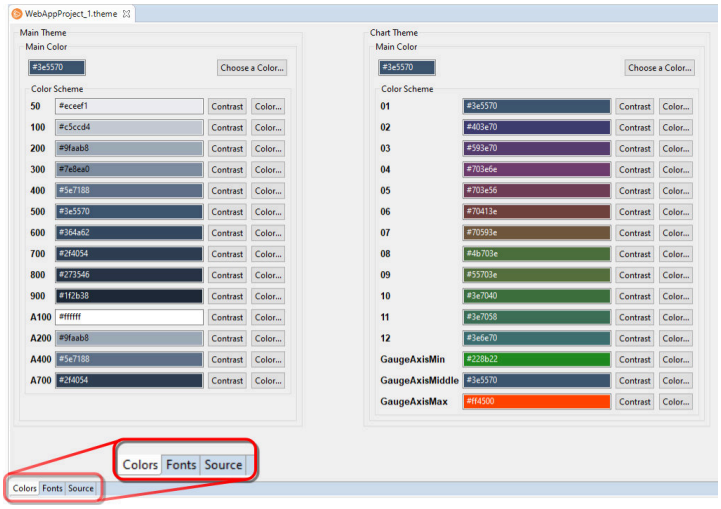
The main font of the Web App is defined using **MainFont**. Additional fonts can be defined using Custom Fonts. Each font has a unique name. The font name can be defined in attributes to apply the font to an element.



3.3.1 Defining fonts

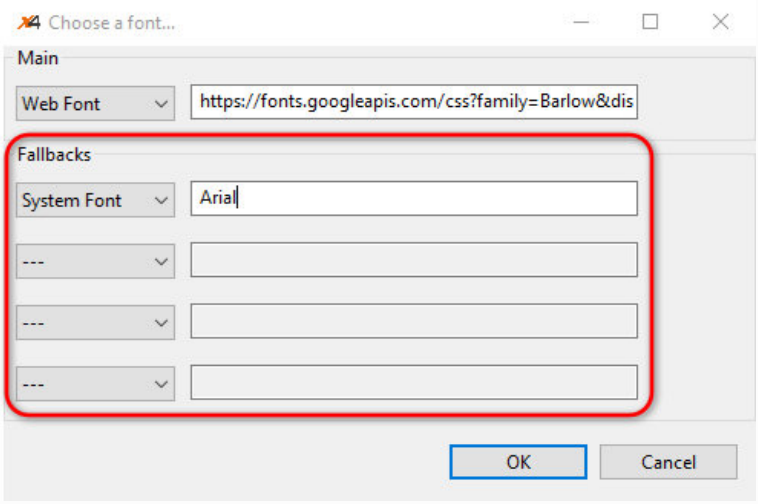
Custom fonts can be defined for a Web App. Fonts are defined using the Theme Editor.

1. Open the *Theme Editor* by double-clicking the `<project name>.theme` file.
2. In the bottom section, switch the view to **Fonts**.



3. **MainFont** sets the font that will be used as the main font of the Web App. Next to **MainFont** click on **Font...** to define the font.
4. Under **Main** select where the font is located.
   Possible values:
   - `Resource Font`: Font is stored as a file in the Web App.

Files of type `eot`, `ttf`, `woff` and `woff2` can be used. The different file types are not supported by all browsers. Up to four substitute fonts can be stored in the *Fonts Editor* in case of problems with the display of a font.



- `System Font`: Font is a system font of the operating system, e.g. `Arial`.
- `Web Font`: font is retrieved via a URL, e.g.
  `https://fonts.googleapis.com/css?family=Barlow&display=swap`
- `Generic`: Font is a standard font of the used browser, e.g. `sans-serif`.

5. If necessary, define substitute fonts under **Fallbacks**.
6. Click **OK** to confirm.
7. Save `<project name>.theme` file.

Documentation of the ReST interface can be found at http://localhost:8080/X4/X4Api/.

4 Configuration

The behavior of the web applications created with Web Apps is defined in the `.wac` configuration. General settings of the Web App, for example the logo or the login behavior, are defined in the `.wac` file.

The root element of the `.wac` file is `<Configuration>`. Within the root element, the `General`, `Access` and `Status` elements control the appearance and behavior of the Web App. When a `.wac` file is declared, the file is automatically validated. In this way, possible error potentials can already be reduced during the creation of web applications.

4.1 Create Configuration

1. Right-click a Web App project in the *X4 Designer*.
2. Select **New > Configuration** in the context menu.

A Configuration `.wac` is created and opened.

4.2 Managing access to Web Apps

Access to a Web App is managed using the role from Keycloak. In Keycloak, a role can be assigned to each user. The assigned roles can be used to control access to a Web App.

In the Configuration `.wac`, you can define for the Web App which role has to be assigned to a user so that the user can access the Web App.

For more information, see section Declaring Configuration.

4.3 Declaring Configuration

The root element of the `.wac` file is `<Configuration>`. The element contains all configurations for the Web App. Attributes are used to control the appearance and behavior of the Web App.

The `<Configuration>` element can contain the following elements:

Possible elements

Element	Description
`General`	Contains configuration definitions about the general appearance of the Web App.
`Access`	Contains configuration definitions for accessing the Web App.

Element	Description
`Status`	Contains configuration definitions for the internal status values of the Web App.
`MapAPIKey`	Contains the API key if requested by the card provider. ⓘ No API key is required to use Openstreetmap. If an API key is entered when Openstreetmap is used as the map provider, the API key is ignored. ⚠ When using Google Maps as a map provider, the following APIs must be enabled: • Maps JS API • Locations (for search and route calculation) • Directions (for route calculation)
`Logout`	Contains configuration definitions for logout.
`Notifications`	Contains configuration definitions for the info banner.
`HTTPSecurityHeader`	Contains configuration definitions for the HTTP security header.
`HeaderIncludes`	Contains configuration definitions for the header. For example, you can insert your own Javascript code in the header.

ⓘ For documentation on the ReST interface, see http://localhost:8080/X4/X4Api/.

✓ For more information, see the section

4.3.1 <General>

The `General` element contains configuration definitions of the general appearance of the Web App, such as the favicon, the display name or whether headers and footers are displayed.

Possible attributes

Attribute	Description
`displayName`	*Required*. Display name of the web app. Displayed in the browser tab or title bar. **Possible values:** Any string

Attribute	Description
`enableOverlayBlurredBackground`	Specify whether the background should be blurred or clear when displaying overlay structure elements. **Possible values:** • `true` (default): For overlays, the part of the Web App that is still visible in the background is blurred. • `false`: For overlays, the part of the Web App that is still visible in the background is displayed clearly.
`faviconUrl`	Path to the image file that will be used as the Web App's favicon. The file must be contained in the `Resources` folder directly inside the Web App project. **Possible values:** String (URI)
`logoUrl`	Path to the image file or specification of a Material Icon that will be used as the Web App logo. • The image file must be stored in the `Resources` folder directly inside the Web App project. Path specification relative to the `Resources` folder. • The Material Icon must be specified with the prefix `icon`, e.g. `icon:<MaterialIconName>`. **Possible values:** • String (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons. The logo is always displayed with a height of 45 pixels. The image file used as the logo is automatically scaled.

Attribute	Description
`logoColor`	Defines the color of the logo in the application. ⚠ This setting overwrites the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash before the color value or a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*
`showFooter`	Defines whether the footer of the web application is displayed. **Possible values:** `true` (default) / `false` ⚠ If there are actions in the displayed component, the footer is displayed in the mobile view independently of the value of the attribute.
`showHeader`	Defines whether the header of the web application is displayed. **Possible values:** `true` (default) / `false`
`imprintLink`	Links external pages as imprint.
`dprLink`	Links external pages as data protection statement.
`enableCookieConsent`	Defines whether the cookie consent pop-up is displayed. **Possible values:** `true` (default) / `false`

✓ For more information, visit the section

4.3.2 <Access>

The `Access` element contains configuration definitions for accessing the Web App.

Possible attributes

Source	Description
`accessRight`	Defines which role can access the module. **Possible values:** Name of a previously defined role ⚠ The roles are managed in the Keycloak Administration Console.

Source	Description
`authorizationFlow`	Defines which authorization flow is used. If you do not want to require a login, use `Public Access`. ⚠ To use the single sign-on functionality, you must use the `Authorization Code Flow`. **Possible values:** • `Public Access` ⚠ Users who access the Web App without login are recognized as `ANONYMOUS_USER`. They cannot be granted special rights. • `Ressource Owner Password Flow` ⚠ If a Web App uses the `Resource Owner Password Flow` authorization flow, a user with a temporary password cannot log in to that Web App. If you want to use temporary passwords, use the `Authorization Code Flow`. The `Resource Owner Password Flow` authorization flow will be removed in the next version. It is not recommended to use the `Authorization Code Flow` due to security reasons." • `Authorization Code Flow` ⚠ Keycloak must be configured according to the selected authorization flow. ✓ For more information, please visit the official Auth0® Inc. website. • https://auth0.com/docs/authorization/flows/resource-owner-password-flow • https://auth0.com/docs/authorization/flows/authorization-code-flow
`realm`	Defines which realm from Keycloak will be used. ⚠ The realm must be specified if the `Authorization Code Flow` or `Resource Owner Password Flow` is used and a realm other than the default should be used.

Source	Description
`client`	Defines which client from Keycloak will be used ⚠ The realm must be specified if the `Authorization Code Flow` or `Resource Owner Password Flow` is used and a client other than the default should be used.
`clientSecret`	Client Secret of the defined client, which can be read from Keycloak. ⚠ The realm must be specified if the `Authorization Code Flow` or `Resource Owner Password Flow` is used and a client secret other than the default should be used.
`managementUser`	User name of the user controlling the Keycloak ReST API. ⚠ The user data specification is required if the `Resource Owner Password Flow` authorization flow is used.
`managementUserPassword`	Password of the user controlling the Keycloak ReST API. ⚠ The user data specification is required if the `Resource Owner Password Flow` authorization flow is used.

✓ For more information, see the section

4.3.3 <Status>

The `Status` element contains configuration definitions for the internal status values of the Web App.

Possible attributes:

Attribute	Description
`active`	Specification whether the project is activated. **Possible values:** `true` / `false` (default) ⚠ If the project is not active, the web application cannot be displayed in the browser.

Attribute	Description
`localOnly`	Specifies whether the project is to be deployed locally only. **Possible values:** `true` / `false` (default) ⚠ If this option is enabled, the web application can be displayed in the browser only in the local system.
`dataFormatVersion`	Specifies which data format is used. **Possible values:** • `1.0` • `2.0` (default) ⚠ The value of the attribute cannot be changed during runtime.

4.3.3.1 dataFormatVersion

Technical Processes in the X4 BPMS can output data in various formats. In order to use the data in X4 Web Apps, the data format in which the data is output by the Technical Process must be defined in the Web App Configuration.

The data format used is defined in the `dataFormatVersion` attribute.

Sample

This example shows the structure of the data format version `1.0`.

```
<Object property="Container">
  <String value="test" property="Text" />
</Object>
```

This example shows the structure of the data format version `2.0`.

```
<Container>
  <Text>test</Text>
</Container>
```

✓ For more information, see the section

4.3.4 <Logout>

The `Logout` element contains configuration definitions for logout.

Possible attributes

Attribute	Description
`redirectUrl`	Redirecting users to the specified URL after logout. Uniform Resource Locator (URL) with any valid protocol (`http://`, `https://`, etc.) **Possible values:** Any valid URL (e.g. `https://www.softproject.de/`)

For more information, see the section

4.3.5 <Notifications>

The `Notifications` element creates an info banner in the Web App.

The content of the info banner is automatically reloaded every 30 minutes. To manually reload the info banner content, refresh your X4 repository.

Possible attributes:

Attribute	Description
`infoBannerProcess`	*Required*. Path to the `.wrf` file that provides the data. The Technical Process must be contained in the `Services/Processes` folder. **Possible values:** String

4.3.5.1 Technical Process

In order to display an info banner, a Technical Process must provide the data in a specified XML structure.

```
<Alerts>
    <Alert>
        <Background></Background>
        <Foreground></Foreground>
        <Icon></Icon>
        <Message></Message>
    </Alert>
</Alerts>
```

Element	Description
`Background`	Defines the background color. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`Foreground`	Defines the font color. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`Icon`	Path to the image file or specification of a Material Icon that will be used as the component's icon. ⓘ • The material icon must be specified with the prefix `icon`, e.g. `icon:<MaterialIconName>`. **Possible values:** • String (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* • Any URL, e.g. `http://mysite.com/logo.png`

Element	Description
`Message`	Message of the info banner. **Possible values:** • Any string

Example

```
<Alerts>
    <Alert>
        <Background>A700</Background>
        <Foreground>900</Foreground>
        <Icon>icon:3d_rotation</Icon>
        <Message>Upcoming maintenance</Message>
    </Alert>
</Alerts>
```

4.3.6 <HTTPSecurityHeader>

The `HTTPSecurityHeader` element creates an HTTP Security Header in the Web App.

Element	Description	Possible values
`HTTPSecurityHeader`	The HTTP Security Header is an HTTP Response Header. The HTTP Security Header is used to protect the Web App against unauthorized attacks.	The element contains the following elements: • `strictTransportSecurity`
`strictTransportSecurity`	The HTTP Strict Transport Security Response Header informs browsers that the website should only be accessed via HTTPS and that all future access attempts via HTTP should be automatically converted to HTTPS.	

4.3.6.1 HTTPSecurityHeader

Possible attributes

Attribute	Description	Possible values
`XFrameOptions`	The X-Frame options in the HTTP Response Header can be used to determine whether a calling browser is allowed to embed the target page in a <frame>, <iframe>, <embed>, or <object>.	• `DENY`: The target page is not allowed to be embedded. • `SAMEORIGIN`: The target page is allowed to be embedded.

4.3.6.2 strictTransportSecurity

Possible attributes

Attribute	Description	Possible values
`enabled`	Sets the status of the HTTP Strict Transport Security Response Header.	• `true`: The Strict Transport Security Response Header is enabled. • `false`: Strict Transport Security Response Header is disabled.
`includeSubDomains`	Specifies whether the HTTP Strict Transport Security Response Header settings are also applied to the subdomains of the Web App.	• `true`: The settings are also applied to the subdomains. • `false`: The settings are not applied to the subdomains.
`maxAge`	The time in seconds for the browser to remember that the Web App may only be accessed via HTTPS.	• Any integer
`preLoad`	Specifies whether the HTTP Strict Transport Security Response Header is preloaded.	• `true`: The HTTP Strict Transport Security Response Header is preloaded • `false`: The HTTP Strict Transport Security Response Header is not preloaded

4.3.7 <HeaderIncludes>

The `HeaderIncludes` element contains configuration definitions for the HTML Head element. For example, you can insert your own JavaScript code in the Head element.

Element	Description	Possible values
`HeaderIncludes`		The element contains the following elements: • `Include`

Element	Description	Possible values
`Include`	Inserts content from a file into the HTML head as meta elements.	

4.3.7.1 Include

Possible attributes

Attribute	Description	Possible values
`path`	Path to a file in the project ⚠ The file must be located in the **Resources** directory.	• Any string (e.g. *`File.txt`*)

✓ For more information, see the section

5 Localization

Web applications that are created with Web Apps can be made available in multiple languages.At the beginning of the `.wad` file, you must specify the languages in which the web application is to be available.

For each language in which the web application is to be available a translation file has to be created in the `Translations` folder. This language file assigns a variable to a name in a language. The name of the language file consists of the name of the language (usually a two-character language abbreviation) and the file extension `translation`, for example, `de.translation` (german language file), `en.translation` (english language file), or `es.translation` (spanish language file).

Note

The value specified for `name` must match the file name of the language file with file extension. The value of `displayName` is displayed during language selection.

```
<?xml version="1.0" encoding="UTF-8" ?>
<WebApp ...>
    <Translations>
        <Translation displayName="Deutsch" name="de.translation"/>
        <Translation displayName="English" name="en.translation"/>
        ...
    </Translations>
</WebApp>
```

5.1 Creating a New Language File

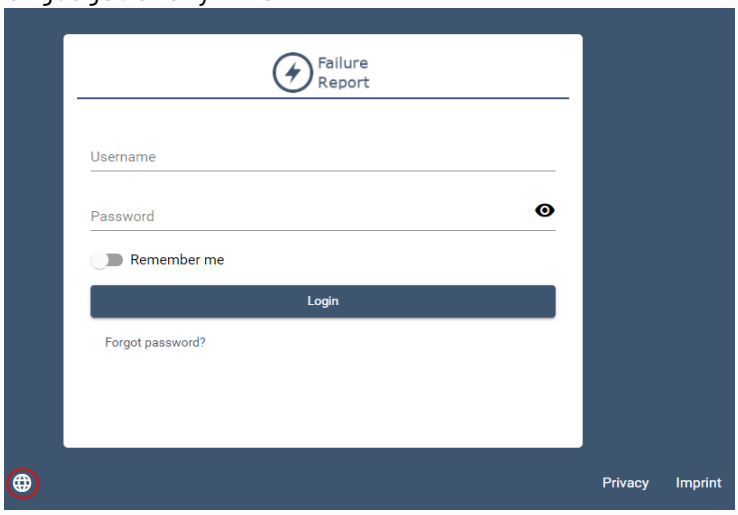
1. Right-click on the `Translations` folder.
   The context menu opens.
2. Click on **New** > **Translation**.
   The dialog for creating a new language file opens.
3. Enter the name of the language file in the **File name** field.
4. Click on **Finish**.

The language file was created. The language file is filled with predefined values in German and contains each possible key. The language file is structured as follows:

```
loginLabel = user name
passwordLabel = password
emailLabel = mail
...
```

The language file consists of a key (e.g. `loginlabel`) and a corresponding name (e.g. `user name`). Many elements support more than one language. The corresponding attributes have to be marked with a $ sign and the key, for example `loginLabel="$LoginLabel"`.

If several languages are available for a Web App, the user can switch between the different languages at any time.



5.2 Translation Keys

The following translation keys are currently provided in German by default and can be translated into as many languages as required:

```
loginLabel = Benutzername
passwordLabel = Passwort
emailLabel = E-Mail
loginButtonLabel = Anmelden
logoutButtonLabel = Abmelden
forgotPasswordLabel = Passwort vergessen?
changePasswordLabel = Passwort ändern
rememberMeLabel = Angemeldet bleiben
wrongPasswordLabel = Der eingegebene Benutzername oder das Passwort sind falsch.
changeButtonLabel = Ändern
required = Pflichtfeld
saveBtn = Speichern
deleteBtn = Löschen
newBtn = Neu
cancelBtn = Abbrechen
backButton=Zurück
notFoundToStart = Zur Startseite
notFoundText = Seite nicht gefunden
notFoundHeader = 404
componentNameNotDeclared = Der Komponentenname in der Aktion ist nicht deklariert.
forbiddenHeader = 403
forbiddenText = Sie haben keine Berechtigung für den Zugriff. Wenden Sie sich an den
Administrator.
dataReloaded = Der ausgewählte Datensatz wurde im Hintergrund geändert. Die Seite
wird neu geladen.
changePasswordButtonOk = Speichern
changePasswordButtonCancel = Abbrechen
changePasswordInvalid = Überprüfen Sie Ihr Passwort
changePasswordSuccess = Ihr Passwort wurde geändert
changePasswordWrong = Passwort entspricht nicht den Passwortrichtlinien
changePasswordReused = Passwort wurde bereits verwendet
changePasswordDoNotMatch = Passwörter stimmen nicht überein
changePasswordNewPassword = Neues Passwort
changePasswordOldPassword = Altes Passwort
changePasswordRepeat = Wiederholen
forgotPasswordButtonOk = Senden
forgotPasswordButtonCancel = Zurück
forgotPasswordRestoreField = E-Mail
greetLabel = Hallo,
errorHeader = Fehler
passwordNotChanged = Das Passwort wurde nicht geändert. Bitte überprüfen Sie Ihre
Eingabe.
requestError = Bei der Anfrage ist leider ein Fehler aufgetreten. Bitte versuchen Sie
es zu einem späteren Zeitpunkt nochmals.
passwordEmailSent = Wir haben eine E-Mail mit weiteren Anweisungen verschickt.
leavePageWithUnsavedDataHeader = Möchten Sie diese Seite wirklich verlassen?
leavePageWithUnsavedDataText = Beim Verlassen der Seite werden Ihre Änderungen
verworfen.
leavePageWithUnsavedDataOkButton = Seite verlassen
leavePageWithUnsavedDataCancelButton = Auf Seite bleiben
uploadBtn = Hochladen
downloadBtn = Herunterladen
multipleNavigationHeader = Navigationsfehler
multipleNavigationText = Es kann nur ein Navigationspfad definiert werden.
```

```
next=Vor
prev=Zurück
today=Heute
monthView=Monat
weekView=Woche
dayView=Tag
monday=Montag
tuesday=Dienstag
wednesday=Mittwoch
thursday=Donnerstag
friday=Freitag
saturday=Samstag
sunday=Sonntag
mondayShort=Mo
tuesdayShort=Di
wednesdayShort=Mi
thursdayShort=Do
fridayShort=Fr
saturdayShort=Sa
sundayShort=So
january=Januar
february=Februar
march=März
april=April
may=Mai
june=Juni
july=Juli
august=August
september=September
october=Oktober
november=November
december=Dezember
start=Start
end=Ende
location=Ort
description=Beschreibung
allDay=Ganztägig
start=Start
destination=Ziel
search=Suche
routingErrorTitle=Fehler
routingError=Die Route konnte nicht berechnet werden.
languagesLabel=Sprachauswahl
scrollingDesktopNonMacWarning=Verwenden Sie Strg + Scrollen zum Zoomen der Karte
scrollingDesktopMacWarning=Verwenden Sie \u2318 + Scrollen zum Zoomen der Karte
scrollingMobileWarning=Verwenden Sie zwei Finger zum Bewegen der Karte
registerLabel = Registrieren
titleLabel = Ihre Privatsphäre ist uns wichtig
defaultCookieConsentText = Einige Cookies sind für die einwandfreie Nutzung der
Webseite unerlässlich, während andere uns helfen, Ihr Erlebnis auf dieser Webseite zu
optimieren und zu verbessern.
dprLabel = Datenschutz
imprintLabel = Impressum
externalMediaCookiesTitle = Externe Medien
```

```
externalMediaCookiesDescription = Inhalte von Videoplattformen und Social-Media-
Plattformen sind standardmä\u00DFig blockiert. Wenn ExternalMedia-Cookies akzeptiert
werden, ist der Zugriff auf diese möglich.
preferencesLabel = Vorlieben
necessaryCookiesBtn = Nur notwendige Cookies zulassen
allowSelectionBtn = Auswahl zulassen
preferencesTitle = Cookie-Einstellungen
preferencesText = Cookies sind kleine Textdateien, die von Webseites verwendet werden
können, um Interaktionen zu analysieren und zu messen und Ihr Erlebnis auf dieser
Webseite zu optimieren und zu verbessern. Diese Webseite verwendet verschiedene Arten
von Cookies. Einige Cookies sind für den Betrieb dieser Webseite unerlässlich; für
alle anderen Arten von Cookies benötigen wir Ihre Zustimmung. Einige Cookies werden
von Diensten Dritter gesetzt, die auf unseren Webseites erscheinen.
allowAllCookiesBtn = Alle Cookies zulassen
essentialCookiesTitle = Essentielle
essentialCookiesDescription = Essentielle Cookies sind erforderlich, um grundlegende
Funktionen zu nutzen und um das reibungslose Funktionieren der Website zu
gewährleisten.
sixMonthText = 6 Monate
youtubePurposeText = Wird verwendet, um YouTube-Inhalte zu entsperren.
bingPurposeText = Wird verwendet, um Bing-Inhalte zu entsperren.
showCookiesDetailsLabel = Details zu Cookies anzeigen
acceptLabel = Akzeptieren
nameLabel = Name
providerLabel = Anbieter
purposeLabel = Aufgabe
hostLabel = Host(s)
cookieNameLabel = Cookie Name
expiryLabel = Cookie Ablauf
waitTitle = Bitte warten
waitText = Sie werden in Kürze weitergeleitet.
waitButton = Zur Anmeldung
```

6 Login page

⚠ If the `authorizationFlow=Authorization Code Flow` attribute is defined in the Configuration `.wac`, the login page of the authentication provider Keycloak is used.

The login page is displayed as soon as a user has to log in when opening the X4 Web App. The login mask is the central component of the login page.

1	Logo. Defined in the Configuration.
2	Password field. The appearance changes depending on the value of the `passwordFieldType` attribute.
3	Slider for the Remember-me function (`rememberMe`)
4	Forgot password option (`forgotPassword`)

You can customize the login mask or create a customized login page.

To customize the login mask on the login page, a .login file must be created in the Web App project. The .login file contains the <Login> element. The attributes of the element can be used to configure the login mask.

To create an individual login mask, a detail component must be declared within the <Login> element. All layouts and controls of the X4 Web Apps can be used in the detail component.

6.1 Create login page

The login page is named after the Web App project and has the .login file extension. Only one .login file can be created within a Web App project.

The `<Login>` element creates the default login mask. The attributes of the element can be used to configure the login mask.

To create an individual login mask, a Detail Component must be declared within the <Login> element. All layouts and controls of the X4 Web Apps can be used in the detail component.

> ⚠ The detail component overrides the default login mask. If you declare a detail component, you must define an individual login mask in the detail component.

1. Right-click on the Web App project.
2. Click **New > Login**.
   The file for the login page is created.

6.1.1 <Login>

> ⚠ The `<Login>` element can contain the following element:
>
> - `DetailComponent`
>
> The attributes set in the `DetailComponent` are ignored and have no effect on the login page.

The following attributes can be defined for the `<Login>` element:

Attribute	Description
`backgroundImageUrl`	Defines an image as background. ⚠ The image file must be located in the `Resources` folder directly inside the Web App project.
`changePassword`	Creates the **Change Password** button in the X4 Web App user menu.
`loginMaskSizeDesktop`	Defines the size of the login mask when the X4 Web App is displayed on desktop computers. **Possible values:** • `Third`: The login screen takes up one third of the login page. • `Half`: The login screen takes up half of the login page. • `Full`: The login screen takes up the entire login page.

Attribute	Description
`loginMaskSizeTablet`	Defines the size of the login mask when the X4 Web App is displayed on tablets. **Possible values:** • `Third`: The login screen takes up one third of the login page. • `Half`: The login screen takes up half of the login page. • `Full`: The login screen takes up the entire login page.

6.2 Create Forgot Password button

To create a **Forgot Password** button on the login page, a Button control must be added to the `<Login>` element in the `.login` file.

A Forgot Password action must then be inserted in the Button control.

6.2.1 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Login
    xmlns="http://softproject.de/webapp/1.0">
    <DetailComponent>
        <FlowLayout>
            <Button>
                <ForgotPasswordAction displayName="$forgotPasswordLabel"/>
            </Button>
        </FlowLayout>
    </DetailComponent>
</Login>
```

7 Modules

Web applications created with X4 Web Apps consist of modules that contain Components. Attributes can be defined for the single modules in the same way as for the web application itself:

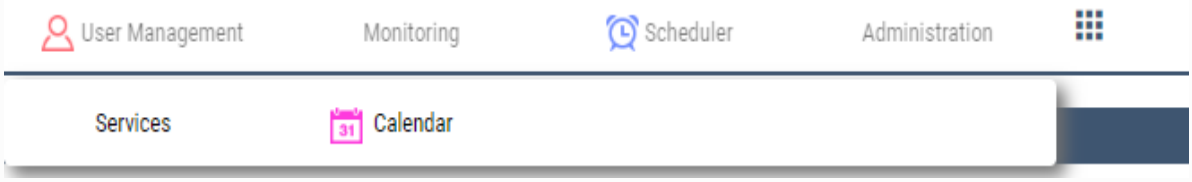
Attribute	Description
`accessRight`	Defines which role can access the module. **Possible values:** Name of a previously defined role ⚠ The roles are managed in the Keycloak Administration Console.
`default`	Defines whether the module should be loaded and opened first. ⓘ Only one module may have the value *`default="true"`*. **Possible values:** `true` / `false`
`displayName`	Display name of the module. Displayed in the browser tab. **Possible values:** Any string
`hideOnAccessDenied`	Defines whether users/groups that do not have access to the module can see the module. **Possible values:** `true` / `false` ⓘ For more information, see Zugriffsrechte für Module definieren.
`iconColor`	Defines the color of the icon in the application. ⚠ This setting overwrites the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash before the color value or a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*

Attribute	Description
`iconUrl`	Path to a graphic file or specification of a Material Icon that will be used as the module's icon. • The graphic file must be contained in the `Resources` folder in the Web App project. Path specification relative to the `Resources` folder. • The Material Icon must be defined with the prefix `icon`, e.g. `icon:<MaterialIconName>`. **Possible values:** • String (URI), e. g. *`clock.png`* • `icon:<MaterialIconName>` , e. g. *`icon:extension`* With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons.
`path`	Path to the module. Displayed in the address bar of the browser. **Possible values:** String of alphanumeric characters (no umlauts, dots, etc.)

Multiple modules can be defined for a web application. They are displayed in the header of the application.

Access rights can also be defined for the individual modules. How to do this is explained in the Zugriffsrechte für Module definieren section.

The first four modules are always displayed in the interface. If more than four modules have been defined, they can be displayed and selected via the submenu .



Example

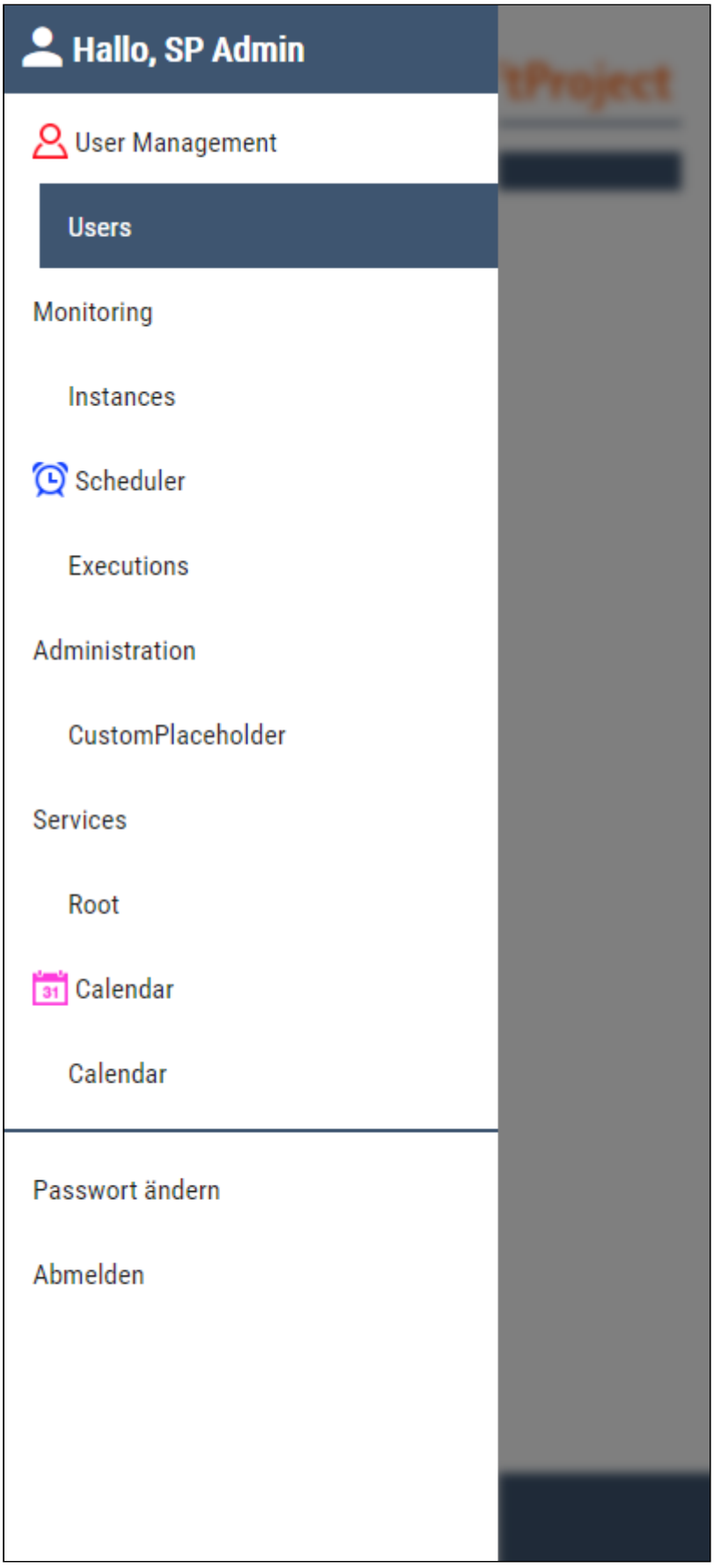
Example: Web app with multiple modules

```
<WebApp xmlns="http://softproject.de/webapp/1.0" path="Modules">
    <Modules>
        <Module displayName="User Management" path="UserManagement" iconUrl="users.pn
g" default="true" >
            <Components>
                <DetailComponent default="true" displayName="Users" path="Users">
                    <FlowLayout>
                        <Header value="User Management"/>
                    </FlowLayout>
                </DetailComponent>
            </Components>
        </Module>
        <Module displayName="Monitoring" path="Monitoring">
            ...
        </Module>
        <Module displayName="Scheduler" path="Scheduler" iconUrl="clock.png">
            ...
        </Module>
        <Module displayName="Administration" path="Administration">
            ...
        </Module>
        <Module displayName="Services" path="Services">
            ...
        </Module>
        <Module displayName="Calendar" path="Calendar" iconUrl="calendar.png">
            ...
        </Module>
    </Modules>
</WebApp>
```

The example above leads to the following result:

Desktop view

Mobile view


Hallo, SP Admin
User Management
Users
Monitoring
Instances
Scheduler
Executions
Administration
CustomPlaceholder
Services
Root
Calendar
Calendar
Passwort ändern
Abmelden

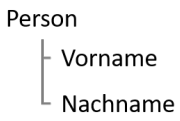
8 Data model and properties definition

Properties can be defined for each component. Properties define the data model to be used in the component. For List Components or Grid Components (standalone or within a Master/Detail Component), information about the data model in which the data is provided by a Technical Process is mandatory. Within one Master/Detail component, data is shared between the different components (see Properties in Master/Detail).

Properties are defined at the beginning of the component declaration within `Properties`:

```
<DetailComponent>
    <Properties>
        <Property name="Person" type="Complex">
            <Property name="Vorname" type="String"/>
            <Property name="Nachname" type="String"/>
        </Property>
    </Properties>
    <FlowLayout>
        <Label value="#Person.Vorname"/>
        <Label value="#Person.Nachname"/>
    </FlowLayout>
</DetailComponent>
```

The data model defined here corresponds to the following scheme:



Possible attributes:

Attribute	Description
`displayName`	Display name of the property. Used in the column headers of Master/Detail Components, List Components and Grid Components. **Possible values:** Any string
`name`	Name of the property. Required. **Possible values:** Any string of alphanumeric characters (`a-z`, `A-Z`, `0-9`) ⚠ Note that the property names `Component`, `OrderBy`, `Where`, `FileId`, `FileName`, `FileType`, `FileData`, `To`, `From` must not be used because they are preset and therefore validated.

Attribute	Description
`readOnly`	Restricts access to read-only. **Possible values:** *`true`* / *`false`*

Attribute	Description
`type`	Name of the property. Required. **Possible values:** • `Base64`: Base64 encoded data • `Boolean`: Boolean values • `Color`: Hexadecimal color value • `Complex`: Object contains other objects (nested properties). Used for list entries, for example. • `Date`: Date specification in the ISO format `YYYY-MM-DD` (e.g. *`2020-01-18`*) • `DateTime`: Date and time specifications in the format `YYYY-MM-DD'T'hh:mm:ss.fff` (e.g. *`2019-01-01T01:00:00.000Z`*) ⓘ The date and time format must conform to ISO 8601. ⓘ If no time zone is specified for a time entry, then the time zone of the server is used. On the server side, this time zone is then added. ⓘ If a value with property type `DateTime` is used in a List component, the specification of date and time is displayed in different ways depending on the existing values: ◦ If all values have the value `0` for seconds (`2020-01-01T00:00:`**`00`**`.000Z`), no seconds are displayed in the whole column. ◦ If at least one value has a value different from `0`, seconds are displayed in the whole column. ⚠ The processing of the property type `DateTime` has changed with version 6.3. For all previous entries of the property type `DateTime`, as of version 6.3 not only the date but also the time is displayed! • `Decimal`: Decimals • `Image`: Graphic (Base64, resource in the folder *Resources* or external URL) ⚠ If the graphic is stored as Base64 encoded string, the attribute `mediaType="image/*"` must be defined. If the image is stored as an external URL, then the attribute `type="Image"` must also be specified. • `Video`: Video (Base64, resource in the folder *Resources* or external URL)

Attribute	Description
	• `Integer`: Integers • `List`: Contains list elements. Used for ComboBox and ListView. • `String`: String • `URL`: URL ⚠ The XML data provided by the Technical Process have to contain the `type` and `externLinkTarget` attributes.

8.1 Data from the Technical Process

Data can be provided by a Technical Process, for example to generate lists dynamically. This data must correspond to the format defined by the properties. The data is bound to the UI elements using Data Binding. Within the XML file provided by a Technical Process, each element corresponds to a property. The value contained by the element is the value of the property.

```
<Ok>
    <Person>
        <Vorname>Max</Vorname>
        <Nachname>Mustermann</Nachname>
    </Person>
</Ok>
```

The XML data provided by the Technical Process has either `<Ok>` or `<OkList>` as root element. `<Ok>` is used for Detail Components, `<OkList>` for List Components and Grid Components.

8.1.1 URL

⚠ The XML data provided by the Technical Process have to contain the `type` and `externLinkTarget` attributes.

Attribute	Description
`displayName`	Display name that will be displayed instead of the record. **Possible values:** Any string
`type`	Defines whether the URL is an internal or external link. **Possible values:** • `internal` • `external`

Attribute	Description
`externLinkTarget`	Defines in which window the URL data type is opened. **Possible values:** • `same`: The URL will be opened in the same window. • `new`: The URL will be opened in a new window.

8.2 Example for a simple properties definition

The following example shows a property named *`Example`* of type *`String`*.

```
<DetailComponent>
    <Properties>
        <Property name="Example" type="String" />
    </Properties>
    <FlowLayout>
        ...
    </FlowLayout>
</DetailComponent>
```

8.3 Example for a complex property definition

Properties can be nested. A property that contains one or more other properties is always of type *`Complex`*.

The following example shows a complex property:

Example for a complex property definition

```
<DetailComponent>
    <Properties>
        <Property name="Example" type="String" />
        <Property name="Example2" type="Complex">
            <Property name="Internal" type="String"/>
            <Property name="Visible" type="Boolean"/>
        </Property>
    </Properties>
    <FlowLayout>
        <Label value="#Example"/>
        <Checkbox checked="#Example2.Visible" displayName="Visible"/>
        <Label value="#Example2.Internal" visible="#Example2.Visible"/>
    </FlowLayout>
</DetailComponent>
```

Example for a XML file with complex properties

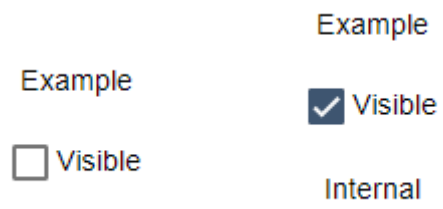
```
<Ok>
    <Example>Example</Example>
    <Example2>
        <Internal>Internal</Internal>
        <Visible>false</Visible>
    </Example2>
</Ok>
```

Example data binding

```
<DetailComponent process="fillDetail.wrf">
    <Properties>

        ...

    </Properties>
    <FlowLayout>
        <Label value="#Example" />
        <Checkbox checked="#Example2.Visible" displayName="Visible" />
        <Label value="#Example2.Internal" visible="#Example2.Visible" />
    </FlowLayout>
</DetailComponent>
```

Result

The result of this example is a detail component that displays data from `detailData.xml`:



8.4 Validation

Properties support validations. These allow to define rules that must apply to the values entered for properties.

To define validation rules, the `Validation` element must be used within the `Property` element. The `Validation` element can contain any number of `Validation` elements with validation rules and messages.

```
<Property name="test" type="Integer">
    <Validations>
        <Validation>
            <Value operator="eq" expected="0" />
            <Message>The entered value is not correct.</Message>
        </Validation>
    </Validations>
</Property>
```

The above example shows a validation rule. This rule checks if the value specified for the property `test` is `0`. The element `Message` is used to output a corresponding message.

> Validations affect if the save action is active. If the validation is not passed, the save action within the component is not active. Therefore, components that contain invalid entries cannot be saved.

8.4.1 Possible validation types

The following validation types can be used as child elements within `Validation`:

Element	Description
`Contains`	Checks if a specific value is contained in a property value. The attribute `value` specifies the corresponding value.
`Email`	*No attributes required.* Checks if the property value is a valid email address.
`EndsWith`	Checks if the property value ends with a specific value. The attribute `postfix` specifies the corresponding value.
`Max`	Defines a maximum value for the property value. The attribute `value` specifies the corresponding value.
`MaxLength`	Defines a maximum string length for the property value. The attribute `value` specifies the corresponding value.
`Message`	Defines the message to be output as result of the validation.
`Min`	Defines a minimum value for the property value. The attribute `value` specifies the corresponding value.
`MinLength`	Defines a minimum string length for the property value. The attribute `value` specifies the corresponding value.
`Regex`	Checks if the property value matches a specific pattern. The attribute `pattern` specifies the corresponding pattern.
`Required`	*No attributes required.* Checks if the property value is not empty.

Element	Description
`StartsWith`	Checks if the property value starts with a specific value. The attribute `prefix` specifies the corresponding value.
`Value`	Checks using an operator (attribute `operator`) if the entered value corresponds to the expected value (`expected`).

8.4.2 Possible operators

Several operators are available for validation rules:

Operator	Description
`eq`	equal
`gt`	greater than
`gt_eq`	*greater than or equal*
`lt`	*less than*
`lt_eq`	*less than or equal*
`neq`	*not equal*

The attribute `expected` is the so-called control value. This value is used to apply the validation.

Example: `<Value operator="gt" expected="0" />` means that the value of a property is checked to see if it is greater than 0.

8.4.3 Example logical operations

The following example shows how validations can be logical linked. If several rules are defined within a `Validation` element, they are linked with AND. If several `Validation` elements including rules are contained within a `Validations` element, then these are linked with OR.

Example validation

```
<Property name="test" type="Integer">
    <Validations>
        <Validation>
            <Value operator="eq" expected="0" />
        </Validation>
        <Validation>
            <Value operator="gt_eq" expected="10" />
            <Value operator="lt_eq" expected="15" />
        </Validation>
    </Validations>
</Property>
```

In this example, the validation is passed, if the value of the property `test` equals `0` OR simultaneously is greater or equal `10` AND less or equal `15`.

8.4.4 Example <Message>

A message can be defined within a Message element. This message is displayed in the input control to which the property to be validated is bound.

Please note the following specifications!

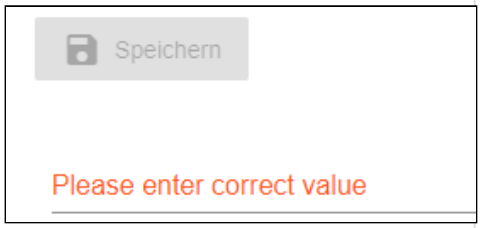
- If the Message element is defined directly within the Validations element, the message is always displayed.
- If no Message element is defined within the Validations elements, the messages defined in the appropriate Validation element will be displayed.
- If no Message element is defined within the Validations element, all Validation elements must contain a message.

- The message is defined directly within the Validations element:

Example: Message is specified directly within the validation element

```
<Property name="test" type="Integer">
    <Validations>
        <Message>Please enter correct value</Message>
        <Validation>
            <Email/>
        </Validation>
    </Validations>
</Property>
```

The above example results in the following messages:



Validation fails because no input was made.

Validation fails because no valid email address has been entered.

Validation is successful. No message is displayed.

- For each validation rule a message is defined:

```
<Validations>
<!-- <Message>Too long</Message> -->
    <Validation>
        <Message>End with fin and contains z</Message>
        <EndsWith postfix="fin" />
        <Contains value="z" />
    </Validation>
    <Validation>
        <Message>Too long</Message>
        <MaxLength value="10" />
    </Validation>
</Validations>
```

9 Components

Web application consist of different components, each of which is defined in a separate file. The components are then referenced within `Components` via the `<ComponentReference>` element.

> ⚠ Inheritance of properties via navigation is not possible.

There are different types of components:

Component	Tag	Description
Detail Component	`DetailComponent`	Displays text
List component	`ListComponent`	Displays data in the form of dynamically generated lists
Calendar Component	`CalendarComponent`	Displays a calendar
Master/Detail component	`MasterDetailComponent`	Can consist of the structural elements `Master`, `Detail` and `Overlay`. A dynamically generated list is displayed in the `master` area. Depending on the selected entry in this list, more detailed information on this entry is displayed in the `detail` structure element.
ExternalWeb component	`ExternalWebComponent`	Embeds an external website into a web app
Grid component	`GridComponent`	Displays data in a tabular format

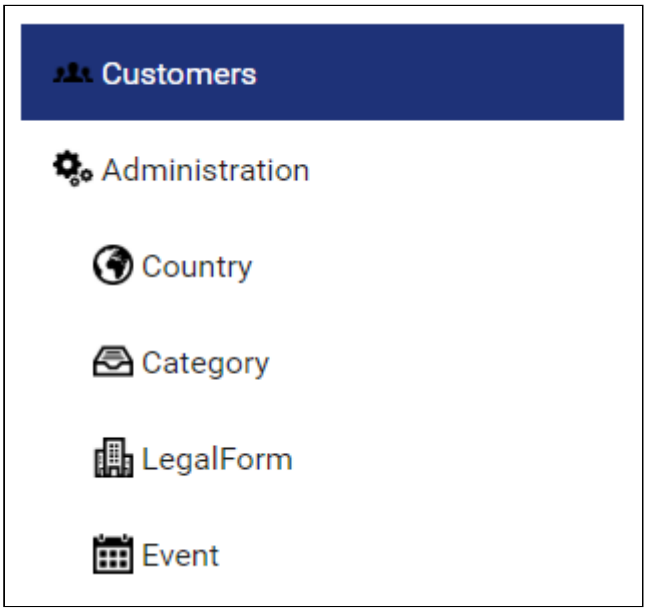
More detailed differences between the components are explained in the respective chapters.

```
<WebApp xmlns="http://softproject.de/webapp/1.0" path="BeispielProjekt">
    <Modules>
        <Module path="Module" displayName="My Module">
            <Components>
                <!-- Bereich für die Komponenten -->
                <ComponentReference/>
            </Components>
        </Module>
    </Modules>
</WebApp>
```

Components can be nested within each other.

```
<WebApp xmlns="http://softproject.de/webapp/1.0">
    <Modules>
        <Module displayName="My Module" path="Module">
            <Components>
                <ComponentReference default="true" displayName="$Customers" iconUrl="
logo.png" name="Home"
                path="Home" source="Customers/Customers.masterdetail"/>
                <ComponentReference displayName="$Administration" iconUrl="Gear.png"
path="Administration"
                source="Administration/Administration.masterdetail">
                    <Components>
                        <ComponentReference displayName="$Country" iconUrl="Country.p
ng" name="Countries"
                        path="Countries" source="Administration/Country/
Country.masterdetail"/>
                        ...
                        <ComponentReference displayName="$Event" iconUrl="Event.png"
name="Event"
                        path="Event" source="Administration/Event/Event.masterdetail/
>
                    </Components>
                </ComponentReference>
            </Components>
        </Module>
    </Modules>
</WebApp>
```

When components are nested, they are displayed as sub-items in the menu:



When nesting components, make sure that a `Components` element is first created within the outer component. `Components` contains the subordinate components.

9.1 Attributes for components

Components can have the following attributes:

Attribut	Description
`fontFamily`	Defines the font family within the component. The property is inherited by all controls and actions of the component. **Possible values:** Font code from the font list, e.g. *`Font04`*
`fontSize`	Defines the font size within the component. The property is inherited by all controls and actions of the component. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in point, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`*. • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Defines the width of the individual characters. The property is inherited by all controls and actions of the component. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the slant of the font. The property is inherited by all controls and actions of the component. **Possible values:** • `italic`: italic font • `normal`: normal font (default) • `oblique`: oblique font style (calculated)

Attribut	Description
`fontWeight`	Defines the font weight. The property is inherited by all controls and actions of the component. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`iconColor`	Defines the color of the component's icon. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash before the color value or a shortened notation of the color value! • Colour code from the colour palette of the web app (cf. Theming), e.g. *`A200`*

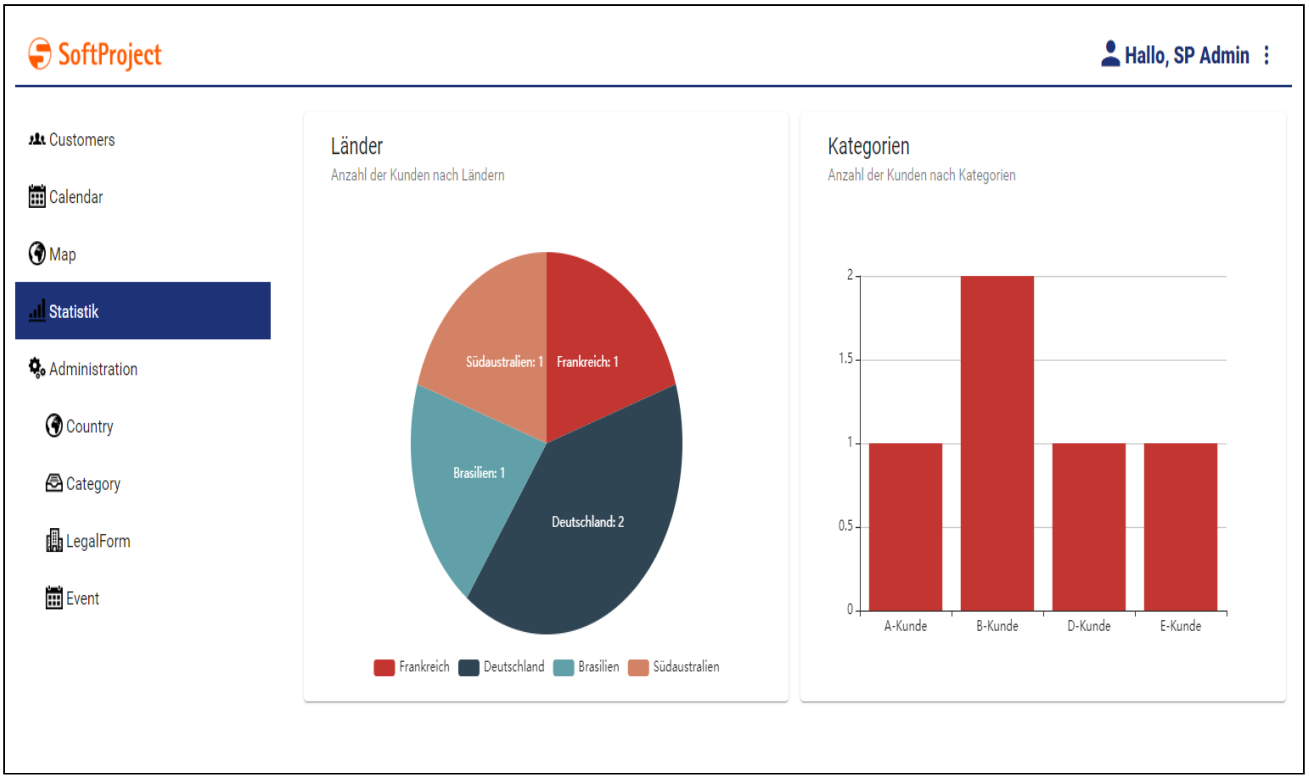
Attribut	Description
`iconURL`	Path to a graphic file or specification of a material icon that is used as the icon of the component. • The graphic file must be contained in the `Resources` folder directly within the web app project. Path specification relative to the `Resources` folder. • The material icon must be specified with the prefix `icon`, e.g. `icon:<MaterialIconName>`. **Possible values:** • Character string (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* With `Ctrl+space` you get an overview of the available icons. Another overview is also available online. The selection may differ from the actual material icons available.
`process`	*Required*. Path to the `.wrf` file that provides the data for the component. The Technical Process must be contained in the `Services/Processes` folder. Path relative to the `Services/Processes` folder. **Possible values:** String (URI)
`title`	Title of the component. Displayed in the application as a header. **Possible values:** Any character string
`titleBackground`	Defines the title background colour. **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` Do not use a hash (#) before the colour value and do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`

Attribut	Description
`titleForeground`	Defines a colour for the title foreground. **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` ⚠ Do not use a hash before the colour value and do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`

ⓘ Components must contain a layout.

9.2 Detail Component

Detail components display content page-based. This can be plain text, but also check boxes or static or dynamic lists, for example.

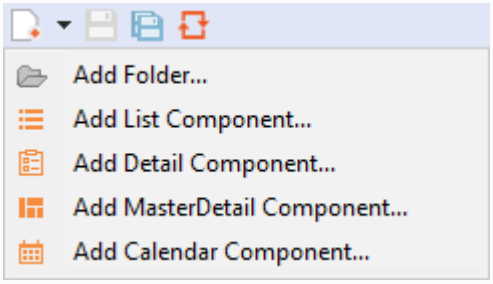


9.2.1 Defining a Detail Component

A detail component is defined within an own definition file `<DetailComponentName>.detail`. The definition file is created within the Repository folder **Components**.

How to create a new definition file:

1. Click the menu **New > Add *<corresponding component>***.
   Alternatively, the file can also be created via the context menu **New > Add *<corresponding component>***.
2. Select the desired definition file.



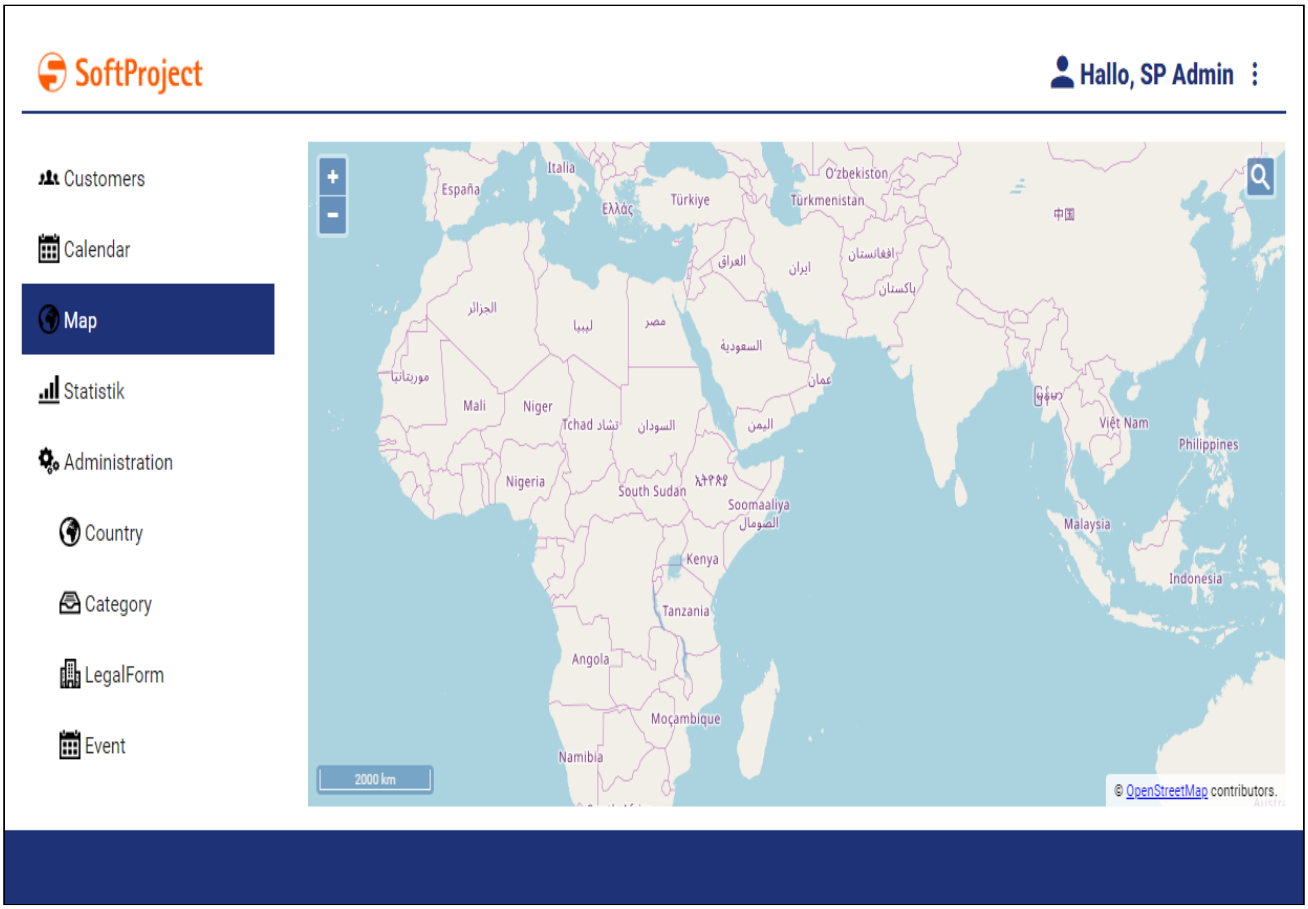
3. Enter a name for the definition file in **File name**.
4. Click **Finish**.
   A new definition file with a predefined structure is created.

The detail component is created via `<DetailComponent>` and can contain the standard attributes for Components.

A layout must be declared for a detail component.

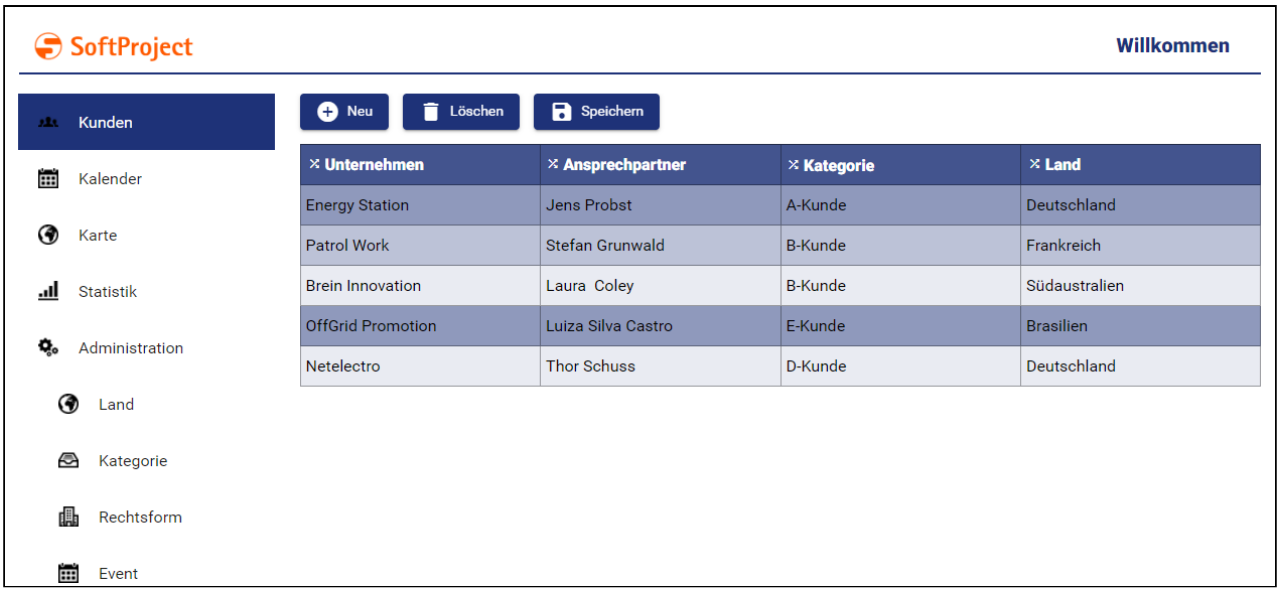
9.2.2 Example for a Detail Component

```
<DetailComponent process="/WebApp/Map/LoadMapData.wrf" xmlns="http://softproject.de/
webapp/1.0">
    <Properties>
        <Property name="Place" type="List">
            <Property name="Company" type="String" />
            <Property name="Address" type="String" />
        </Property>
    </Properties>
    <FlowLayout>
        <Map vendor="OpenStreetMap">
            <Search />
            <Markers data="#Place" iconUrl="Location.png" name="#Company"
searchString="#Address" />
        </Map>
    </FlowLayout>
</DetailComponent>
```



9.3 List component

A list component displays dynamically generated lists. The component requires data and information on how this data is to be displayed.

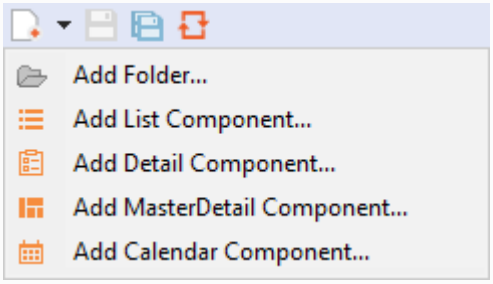


9.3.1 Defining List component

A List component is declared within its own definition file `<ListName>.list`. This is created in the **Components** project folder.

How to create a new definition file:

1. Click the menu **New > Add *<corresponding component>***.
   Alternatively, the file can also be created via the context menu **New > Add *<corresponding component>***.
2. Select the desired definition file.



3. Enter a name for the definition file in **File name**.
4. Click **Finish**.
   A new definition file with a predefined structure is created.

The List component is created via `<ListComponent>` and can have the following additional attributes besides the default attributes for Components.

Attribute	Description
`pageSize`	Defines the number of elements which are loaded into the list at once (see Paging). **Possible values:** Integer greater than `0`
`rowBackground`	Defines the background color of the rows in the list. **Possible values:** • Color value from the theme (e.g. `A50`, `A400` or `09` see Theming) • Hexadecimal color value ⚠ If a # character is to be used in a data binding expression, the character must be escaped. A backslash (\) is used as escaping character. • Data binding expression (Color)

Attribute	Description
`rowForeground`	Defines a color for the foreground (texts etc.). **Possible values:** • Color value from the theme (e.g. `A50`, `A400` or `09` see Theming) • Hexadecimal color value ⚠ If a # character is to be used in a data binding expression, the character must be escaped. A backslash (\) is used as escaping character. • Data binding expression (Color)
`layout`	Defines the layout mode of the list. **Possible values:** • `fixed`: (default) The column `width` of all columns is identical as long as no column width is defined. If a column width is defined, this column width will be respected. • `optimized`: The column `width` is dynamically calculated based on the table content and the optionally set attributes to display all content on the available space. ⓘ If line breaks are prevented, this can create vertical scrollbars if needed.
`textOverflow`	Defines what should happen if overflowing texts cannot be displayed. **Possible values:** • *`ellipsis`* : Show with `...` that the text is not finished. Text break is prevented. • *`hidden`* : (default when `layout="optimized"`) Break off text, paying attention to whole words. Text break is prevented. • *`wordBreak`* : (default when `layout="fixed"`) Break within word. • `allow`: Break off text between words

`borderColor`	Defines the border color of a list component.
	
	Possible values
	
	• Do not use a hash before the color value.
	  • Do not use abbreviated notation of the color value.
	  • Hexadecimal color value, e.g. `ff5a00`
	• Color code from the color palette of the Web App (see Theming), e.g. `A200`

The data to be displayed must be provided by a Technical Process and correspond to a specific data model. The data model delivered by the Technical Process must be defined in the Properties. To link the data that is delivered by the Technical Process with the representation within the web

application, the corresponding `Property` must be defined and stored in the column definition using Data Binding.

> ⓘ If a property is to be used in the list component, it must be defined as *`Complex`* type. This property can contain further properties.

9.3.2 Column definition

The list columns must be defined within the list component. Columns are defined using `Column` or `ImageColumn` for columns that contain images. All columns must be defined within `Columns`.

```
<ListComponent process="list.wrf">
    <Properties>
        ...
    </Properties>
    <Columns>
        <Column .../>
        <Column .../>
        <ImageColumn .../>
    </Columns>
</ListComponent>
```

Possible attributes for `<Column>`:

Attribute	Description
`displayName`	Title of the column. • Translatable • Data binding possible **Possible values:** Any string
`horizontalAlign`	Defines the horizontal alignment of the content within the column. **Possible values:** • `left`: Content left-aligned • `center`: Content centered • `right`: Content right-aligned

Attribute	Description
`mobilePosition`	Defines the position at which the column is displayed in the mobile view. A maximum of four values can be displayed in the mobile view. **Possible values:** • `topLeft` • `topRight` • `bottomLeft` • `bottomRight` ⓘ If this attribute is not set, the column is not displayed in the mobile view.
`sortable`	Defines if the column can be sorted according to its content. More information on Sorting. **Possible values:** `true` / `false`
`value`	Defines the content of the column using data binding. **Possible values:** String (data binding) ⓘ If no attribute `displayName` is set to the corresponding property, the data binding expression is used as column header.
`width`	Defines the column width. **Possible values:** • Integer • `auto`: The width adapts to the content. Can only be used in layout mode `optimized`. ⓘ The unit in which the column width is specified is set with the `widthUnits` attribute.

Attribute	Description
`widthUnits`	Defines the unit in which the column `width` is defined. **Possible values:** • `pixels`: The column width is specified in pixels. • `percents`: The column width is specified in percents.
`textOverflow`	Defines what should happen if overflowing texts cannot be displayed. **Possible values:** • *`ellipsis`* : Show with `...` that the text is not finished. Text break is prevented. • *`hidden`* : (default when `layout="optimized"`) Break off text, paying attention to whole words. Text break is prevented. • *`wordBreak`* : (default when `layout="fixed"`) Break within word. • `allow`: Break off text between words ⓘ The attribute in `<Column>` has a higher priority and overrides the attribute in `<ListComponent>`.

Possible attributes for `<ImageColumn>`:

Attribute	Description
`displayName`	Title of the column. • Translatable • Data binding possible **Possible values:** Any string
`horizontalAlign`	Horizontal alignment of the graphic in the column **Possible values:** • `left`: Graphic left-aligned • `center`: Graphic centered • `right`: Graphic right-aligned
`iconColor`	Defines the color of the image. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash in front of the color value or a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*

Attribute	Description
`image`	Graphic that is displayed in the column. **Possible values:** Data binding expression (Image)
`imageHeight`	Height of the graphic **Possible values:** Integer ⓘ The unit in which the height is specified is set with the `imageUnits` attribute.
`imageUnits`	Unit for the width and height of the graphic. **Possible values:** • `pixels` • `percents` (default)
`imageWidth`	Width of the graphic **Possible values:** Integer ⓘ The unit in which the width is specified is set with the `imageUnits` attribute.
`mobilePosition`	Defines the position at which the column is displayed in the mobile view. A maximum of four values can be displayed in the mobile view. **Possible values:** • `topLeft` • `topRight` • `bottomLeft` • `bottomRight` ⓘ If this attribute is not set, the column is not displayed in the mobile view.

9.3.3 Periodic reloading

The `ReloadTimer` element can be used to reload the List component periodically.

⚠ Periodic reloading of data can cause a high load and should only be used in rare use cases.

Possible attributes for `<ReloadTimer>`:

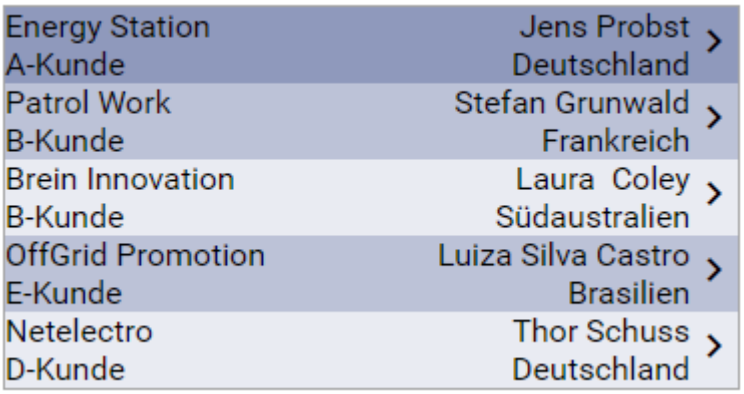
Attribute	Description
`refreshTime`	Time until reload in seconds • Data binding (integer) possible **Possible values:** any integer
`itemIdentifier`	*Optional*; Unique key for the data represented in the rows. • The attribute can be used to retrieve rows during reloading and improve the reloading process. • If the attribute is not defined, a string comparison on a record is used to identify the row. **Possible values**: Data Binding to a defined property of the grid component

9.3.4 Mobile view of the list component

In the mobile view, lists are displayed in compressed form with a maximum of four columns. The four columns are arranged in the four corners of a rectangle; the arrangement can be defined with the attribute `mobilePosition`. Other columns are not displayed in the mobile view.

Unternehmen	Ansprechpartner	Kategorie	Land
Energy Station	Jens Probst	A-Kunde	Deutschland
Patrol Work	Stefan Grunwald	B-Kunde	Frankreich
Brein Innovation	Laura Coley	B-Kunde	Südaustralien
OffGrid Promotion	Luiza Silva Castro	E-Kunde	Brasilien
Netelectro	Thor Schuss	D-Kunde	Deutschland

List in the desktop view



List in the mobile view

9.3.5 "Select" Action within a List component

The **Select** action is created via the element `<SelectAction/>` within a `<Actions>` and is triggered as soon as an entry has been selected in the List component.

In addition to the standard attributes of the Select action, the following attributes are available:

Attribute	Description
`disableAutoselect`	*Only when used within a Master/Detail component.* Defines if the first entry of a list is already selected when opening the component. **Possible values:** *`true`* / *`false`*

9.3.6 Example list component

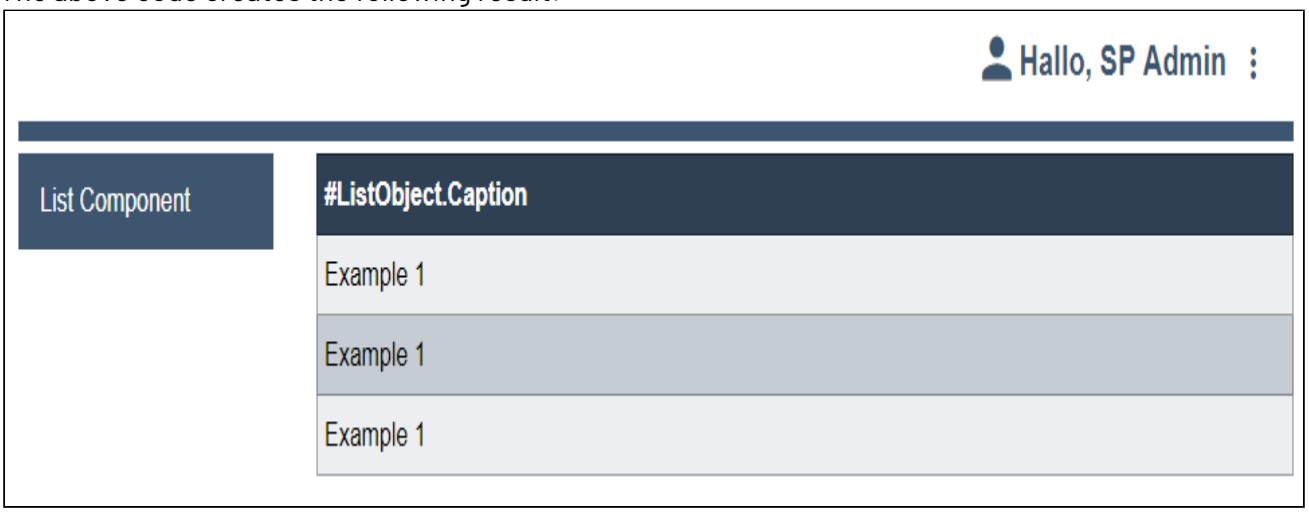
The following example shows the declaration of a list component.

```
<ListComponent process="list.wrf">
    <Properties>
        <Property name="ListObject" type="Complex">
            <Property name="Caption" type="String" />
        </Property>
    </Properties>
    <Columns>
        <Column value="#ListObject.Caption" />
    </Columns>
</ListComponent>
```

The component processes data in the following format.

```
<?xml version="1.0" encoding="UTF-8" ?>
<OkList size="30">
    <ListObject>
        <Caption>Example 1</Caption>
    </ListObject>
    <ListObject>
        <Caption>Example 1</Caption>
    </ListObject>
    <ListObject>
        <Caption>Example 1</Caption>
    </ListObject>
</OkList>
```

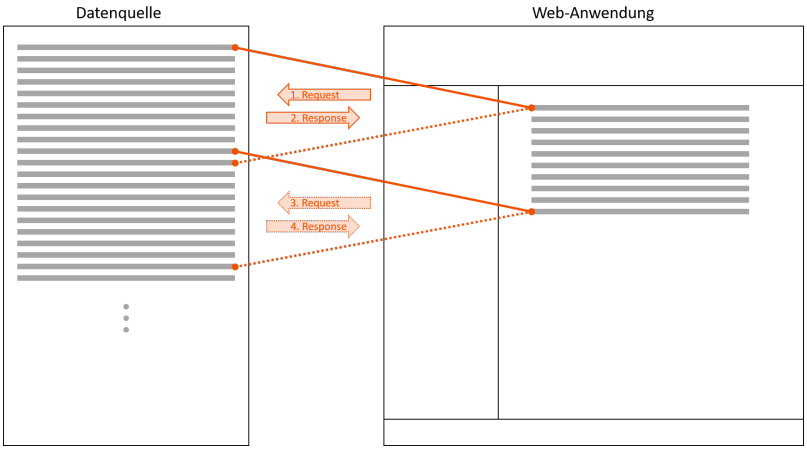
The above code creates the following result:



9.3.7 Paging

Web applications that are created with X4 Activities Web App automatically support paging in the master/detail and list components. Therefore, the process that provides data for these components must also support paging an process information on how many objects are to be read from which area.

With paging, data is only loaded in subsets from the data source step by step. This avoids large data streams that are very difficult to process by the data source or the browser. Unlike conventional Web applications, Web applications created with X4 Activities Web Apps make it possible to reload the new data while scrolling. The user does not have to explicitly turn to the next page.



9.3.7.1 Sample request

The component that is linked with a Technical Process sends paging requests that can be processed. `offset` is used to define from which object the readout is to start. `limit` specifies how many objects are to be queried.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Search offset="0" limit="50">
    <OrderBy />
    <Where />
</Search>
```

Attribute	Description
`offset`	Defines the offset of the data query, i.e. from which data set the data source is to be read. **Possible values:** Integer
`limit`	Number of data sets to be included in the response. **Possible values:** Integer greater than `0` ⚠ The value of this attribute is automatically stored by the `pageSize` attribute of the List component!

9.3.7.2 Sample response

The response that is returned by the Technical Process must contain information on how many objects are contained in the list (attribute `size`) in addition to the requested objects, so that it is clear if further objects need to be queried.

```
<OkList size="2">
    <List>
        <Description>Value</Description>
        <Id>0</Id>
    </List>
    <List>
        <Description>Value</Description>
        <Id>1</Id>
    </List>
</OkList>
```

Attribute	Description
size	The total number of records in the data source. **Possible values:** Integer

9.3.7.3 Example for paging

The following example contains a list component with a Technical Process and a properties definition.

> The following example is only an example to demonstrate the principle of paging, it should not be considered as best practice.

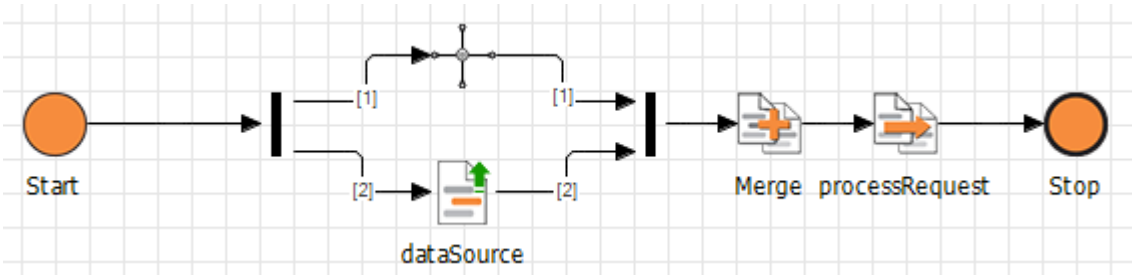
```
<ListComponent name="Paging" path="Paging" displayName="Paging" process="fillListProc
ess.wrf" default="true">
    <Properties>
        <Property name="List" type="Complex">
            <Property name="Description" type="String" displayName="Description"/>
            <Property name="Id" type="Integer" displayName="Id"/>
        </Property>
    </Properties>
    <Columns>
        <Column value="#List.Id" />
        <Column value="#List.Description" />
    </Columns>
</ListComponent>
```

In this example there is an XML file as data source containing 67 entries:

dataSource.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<List>
    <Description>Value</Description>
    <Id>0</Id>
    ...
    <Description>Value</Description>
    <Id>66</Id>
</List>
```

To process the request and get only the entries that are needed, a Technical Process must be designed:



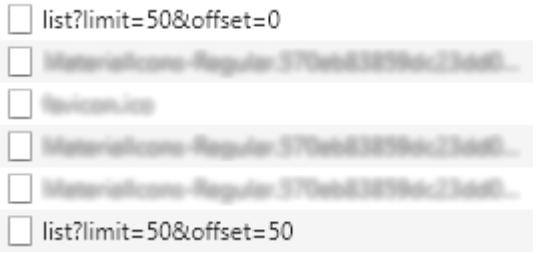
First, the data source and the request are merged in this Technical Process so that they can then be processed in a mapping.

The mapping looks like this:

processRequest.xsl

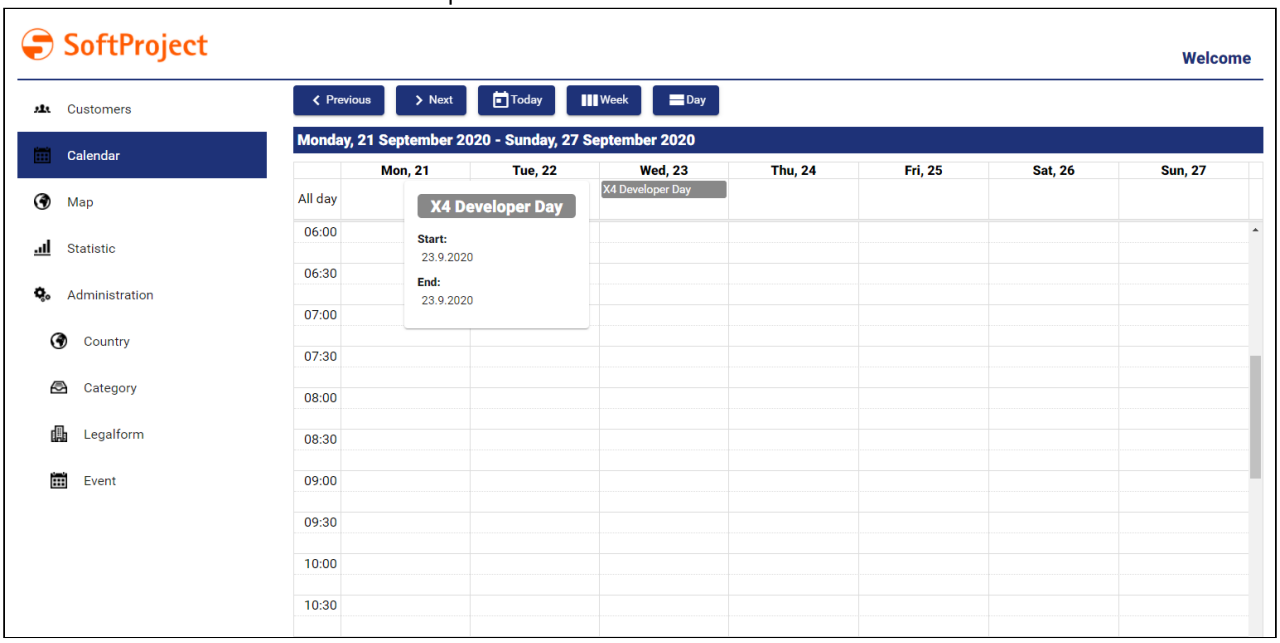
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output media-type="text/xml" method="xml"/>
  <xsl:template match="/">
    <OkList>
      <xsl:attribute name="size" select="count(Merge/List/Id)"/>
      <xsl:for-each select="Merge/List/Id">
        <xsl:if test="number(Id)>=/Merge/Search/@offset and number(Id)< /Merge/
Search/@offset+/Merge/Search/@limit">
          <List>
            <Description>
              <xsl:value-of select="Description"/>
            </Description>
            <Id>
              <xsl:value-of select="Id"/>
            </Id>
          </List>
        </xsl:if>
      </xsl:for-each>
    </OkList>
  </xsl:template>
</xsl:stylesheet>
```

The mapping processes the `offset` and `limit` information from the query (`/Merge/Search/@offset` bzw. `/Merge/Search/@limit`) and sorts out the entries that do not meet these conditions. A data model with `<OkList>` as root element is generated for the entries that fulfill the conditions. This data model can be read by the Web App. The total number of entries in the list is added to `<OkList>` with the `size` attribute. If you scroll now in the list component, the component knows that there are more entries than are displayed. These entries are loaded with a new query with a different offset.



9.4 Calendar Component

A calendar component displays a calendar and events. A calendar component can be used stand-alone or within a master/detail component.

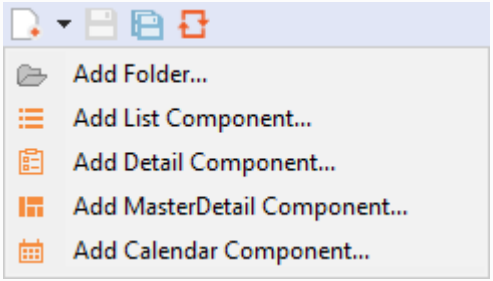


9.4.1 Defining a Calendar Component

A calendar component is defined within an own definition file `<CalendarName>.calendar`. The definition file is created within the Repository folder **Components**.

How to create a new definition file:

1. Click the menu **New > Add *<corresponding component>***.
   Alternatively, the file can also be created via the context menu **New > Add *<corresponding component>***.
2. Select the desired definition file.



3. Enter a name for the definition file in **File name**.
4. Click **Finish**.
   A new definition file with a predefined structure is created.

The calendar component is created via `<CalendarComponent>` and can contain the standard attributes for Components.

9.4.2 Calendar settings: Views

The calendar settings determine which views of the calendar can be displayed by the user and which view is the default view. The calendar settings are made using attributes of the `<CalendarSettings>` element.

```
<CalendarComponent ...>
    ...
    <CalendarSettings defaultView="Day" ...>
 </CalendarComponent>
```

Possible attributes:

Attribute	Description
`dayView`	Enables and disables the day view. **Possible values:** `true` (default) / `false`
`dayViewSlot`	Defines the time scale in the day view. **Possible values:** • `00:15`: 15 minutes as smallest unit • `00:30`: 30 minutes as smallest unit • `01:00`: 1 hour as smallest unit
`defaultView`	Defines the default view of the calendar. **Possible values:** • `Month`: Month view (default) • `Week`: Week view • `Day`: Day view ⓘ A view must be enabled to be defined as default view.
`monthView`	Enables and disables the month view. **Possible values:** `true` (default) / `false`
`weekView`	Enables and disables the week view. **Possible values:** `true` (default) / `false`

Attribute	Description
`weekViewSlot`	Defines the time scale in the week view. **Possible values:** • `00:15`: 15 minutes as smallest unit • `00:30`: 30 minutes as smallest unit • `01:00`: 1 hour as smallest unit

The calendar views (`monthView`, `weekView`, `dayView`) are linked with buttons that allow the user to select the respective view. If a calendar view is disabled, the corresponding button is hidden.

9.4.3 Calendar settings: Localization

A calendar component can be localized. A German user interface is provided by default. The `<Localization>` element within `<CalendarSettings>` can be used to define any translation or custom key for translation files.

```
<CalendarComponent ...>
    ...
    <CalendarSettings ...>
        <Localization march="March" ... />
    </CalendarSettings>
 </CalendarComponent>
```

If a translation file is created for the corresponding Web app, all translation keys are available.

Available translations:

Key	Value
`next`	Next
`prev`	Back
`today`	Today
`monthView`	Month
`weekView`	Week
`dayView`	Day
`monday`	Monday
`tuesday`	Tuesday
`wednesday`	Wednesday
`thursday`	Thursday
`friday`	Friday

Key	Value
saturday	Saturday
sunday	Sunday
mondayShort	Mo
tuesdayShort	Tu
wednesdayShort	We
thursdayShort	Th
fridayShort	Fr
saturdayShort	Sa
sundayShort	Su
january	January
february	February
march	March
april	April
may	May
june	June
july	July
august	August
september	September
october	October
november	November
december	December
start	Start
end	End
location	Location
description	Description
allDay	All Day

9.4.4 Events

Events are defined with the element `<EventConfiguration/>`. The element is used to bind values from a list to events in the calendar. The list that contains the events is delivered by a Technical Process to the Calendar component.

```
<CalendarComponent ...>
    ...
    <CalendarSettings ...>
        <Localization .../>
    </CalendarSetting>
    <EventConfiguration/>
 </CalendarComponent>
```

Possible attributes:

Attribute	Description
`allDay`	Defines if the event takes place all day. **Possible values:** `true` / `false` or data binding expression (Boolean)
`background`	Defines the color with which the event is displayed. **Possible values:** HTML color code e.g. `#FF0000` or data binding expression (Color)
`begin`	Defines the start time of the event. **Possible values:** Data binding expression (DateTime)
`description`	Defines the event description. **Possible values:** Any string or data binding expression (string)
`detailNavigationLinkName`	Name of the button opening the detail view, if the attribute `externalNavigationLink` is not set. **Possible values:** • Any string • `empty`: The default value **Detail view** is displayed ⓘ If the calendar is placed within the master component of a master/detail component, the detail view cannot be accessed on mobile devices since it is opened by an external navigation link. If the event does not have an external navigation link, you can navigate to the detailed view using a button. The attribute `detailNavigationLinkName` specifies the name of this button. The button is only displayed in the mobile view, if the calendar component is located within a master/detail component.

Attribute	Description
`end`	Defines the end time of the event. **Possible values:** Data binding expression (DateTime)
`externalNavigationLink`	Link to be opened when the event is clicked. **Possible values:** Hyperlink (string) or data binding expression (string)
`externalNavigationLinkName`	Text that is displayed for an external link instead of the URL. **Possible values:** String ⓘ If the `externalNavigationLinkName` attribute is not defined for an external link, the URL that is defined for the `externalNavigationLink` attribute is displayed.
`externalNavigationLinkTarget`	Defines whether the link (`externalNavigationLink`) is opened within the same tab or a new tab. **Possible values:** • `same`: The link is opened within the same tab • `new`: The link is opened in a new tab
`fontFamily`	Defines the font family. **Possible values:** Font code from the font palette, e.g. `Font04`
`fontSize`	Defines the font size. **Possible values:** • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Keywords, e.g. *`small`* or *`larger`*

Attribute	Description
`fontStretch`	Defines the width of the single characters. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font inclination. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)
`fontWeight`	Defines the font weight. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines a color for the event name. **Possible values:** • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`location`	Defines the event location. **Possible values:** Any string or data binding expression (string)
`name`	Defines the name of the event. **Possible values:** Any string or data binding expression (string)

9.4.5 External Links in Events

Events can contain external links. They are specified within the element `<EventConfiguration/>` using a `<SelectAction>`.

> The element `<SelectAction/>` can have the standard attributes for Actions and the attributes, which are specific for the Aktion "Auswählen".

```
<EventConfiguration>
    <SelectAction externalLink="#event.link"/>
</EventConfiguration>
```

9.4.6 Data for the calendar component

The calendar component needs data in a list as input to display events. This data is provided by a Technical Process and must comply with the following scheme.

```
<CalendarComponent displayName="Calendar" path="calendar" process="events.wrf">
    <Properties>
        <Property name="List" type="Complex">
            <Property name="Name" type="String"/>
            <Property name="AllDay" type="Boolean"/>
            <Property name="Begin" type="DateTime"/>
            <Property name="End" type="DateTime"/>
            <Property name="Color" type="Color"/>
            <Property name="Link" type="String"/>
        </Property>
    </Properties>
    <EventConfiguration allDay="#List.AllDay" name="#List.Name" begin="#List.Begin"
end="#List.End" color="#List.Color" externalNavigationLink="#List.Link"/>
</CalendarComponent>
```

> The data structure must be specified in the properties of the component!

9.4.7 Example

The definition of a calendar component can look like this.

Example CalendarComponent

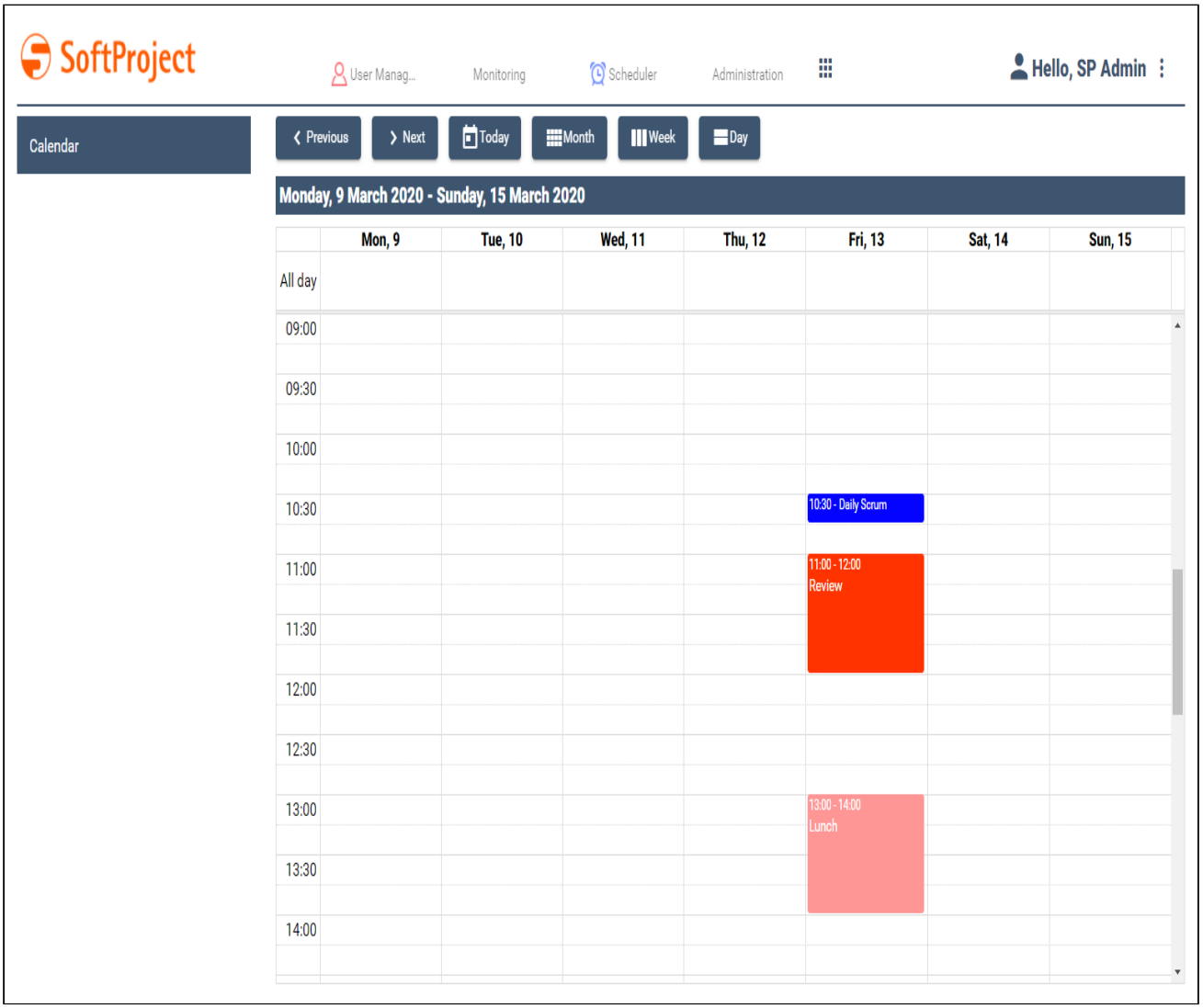
```
<CalendarComponent
    xmlns="http://softproject.de/webapp/1.0" process="calendar.wrf">
    <Properties>
        <Property name="event" type="Complex">
            <Property name="id" type="String"/>
            <Property name="name" type="String"/>
            <Property name="allDay" type="Boolean"/>
            <Property name="begin" type="DateTime"/>
            <Property name="end" type="DateTime"/>
            <Property name="color" type="Color"/>
            <Property name="link" type="String"/>
        </Property>
    </Properties>
    <CalendarSettings defaultView="Week"/>
    <EventConfiguration allDay="#event.allDay" background="#event.color" begin="#even
t.begin" end="#event.end" name="#event.name">
        <SelectAction externalLink="#event.link"/>
    </EventConfiguration>
</CalendarComponent>
```

The input is provided by the `calendar.wrf` Technical Process and looks like this:

Example Input Events.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<OkList>
    <event>
        <id>1</id>
        <name>Daily Scrum</name>
        <allDay>false</allDay>
        <begin>2020-03-13T09:30:00Z</begin>
        <end>2020-03-13T09:45:00Z</end>
        <color>0303FF</color>
        <link>https://www.scrum.org/resources/what-is-a-daily-scrum</link>
    </event>
    <event>
        <id>2</id>
        <name>Review</name>
        <allDay>false</allDay>
        <begin>2020-03-13T10:00:00Z</begin>
        <end>2020-03-13T11:00:00Z</end>
        <color>FF3300</color>
        <link></link>
    </event>
    <event>
        <id>3</id>
        <name>Lunch</name>
        <allDay>false</allDay>
        <begin>2020-03-13T12:00:00Z</begin>
        <end>2020-03-13T13:00:00Z</end>
        <color>FF9696</color>
        <link></link>
    </event>
</OkList>
```

The following result is output:



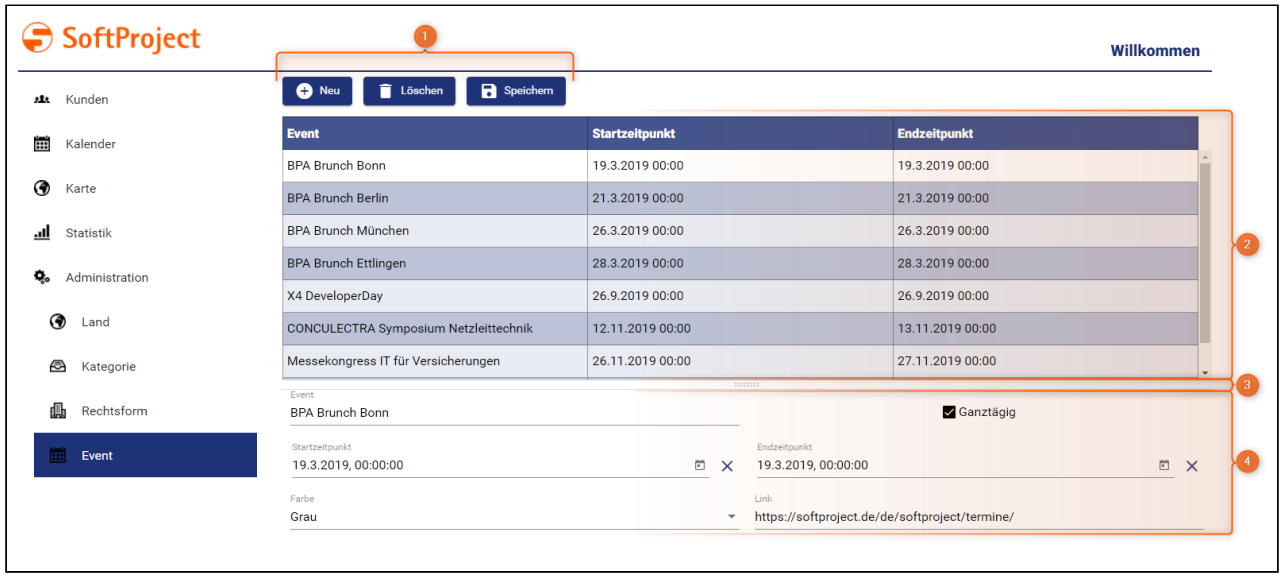
9.5 Master/Detail component

The master/detail component consists of Actions and structural elements Master, Detail and Overlay.

Element	Description
`Master`	Contains the component that is displayed at the top of the screen.
`Detail`	Contains the component that is displayed if a certain condition is fulfilled. By default, content is displayed in this area if a defined list entry is selected within the `Master` element.
`Overlay`	Displays pop-ups. Components that are declared accordingly are displayed in the pop-up on the desktop and as a separate page on mobile devices.

The structure elements Master, Detail and Overlay can contain a reference to a Detail or a List components, see Referencing Components in the Web App definition. In the `Detail` and `Overlay` element, the Master/Detail component can contain multiple referenced Components. This possibility

can be used by using actions with defined `componentName` attribute and declared components with defined `name` attribute.



1	Actions
2	Master structural element
3	Detail structural element

For more information on Components, Actions, properties and Data Binding, see the corresponding sections.

9.5.1 Properties in Master/Detail

For each component, a Technical Process for loading the data can be defined. The data model can be reused.

The properties of the Master/Detail component can be defined both in the Master/Detail component itself and in the subordinate components within the structure elements. A common data model for the entire Master/Detail component is created from these property definitions. This also means that conflicts between Properties within a Master/Detail component (e.g. same name but different data type) are not permitted and lead to a validation error.

All components within a Mster/Detail component therefore work with the same data (for List components related to the selected entry). For overlays, a temporary copy of the data is created, which is synchronized back after successful processing.

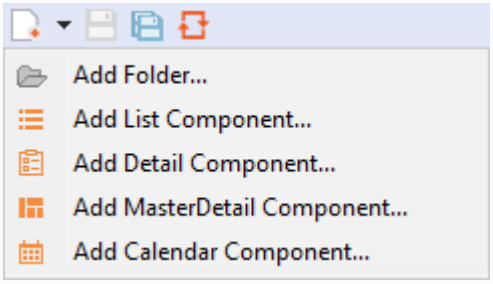
Components included via `ComponentReference` can only access Properties that are also defined within this component. The components work nevertheless on the same data object, have however a limited view of the Properties, which they actually need. Especially when calling processes, this avoids unnecessary data transfer.

9.5.2 Defining Master/Detail component

A Master/Detail component is declared within its own definition file `<MasterDetailComponentName>.masterdetail`. This is created in the **Components** project folder.

How to create a new definition file:

1. Click the menu **New > Add *<corresponding component>***.
   Alternatively, the file can also be created via the context menu **New > Add *<corresponding component>***.
2. Select the desired definition file.



3. Enter a name for the definition file in **File name**.
4. Click **Finish**.
   A new definition file with a predefined structure is created.

The Master/Detail component is created via `<MasterDetailComponent>` and can have the standard attributes for Components.

Attribute	Description
`defaultSeparatorPosition`	Default horizontal position of the separator in percent **Possible values:** • `30 - 70`

The following example shows the basic structure of a Master/Detail component.

```
<MasterDetailComponent xmlns="http://softproject.de/webapp/1.0">
    <Actions>
        <NewAction componentName="CustomerDetail" />
    </Actions>
    <Master>
        <ComponentReference source="Customer/Customer.list" name="AllCustomers" />
    </Master>
    <Detail>
        <TabGroup>
            <ComponentReference source="Customer/Tabs/Company.detail" displayName="$C
ompany" name="CustomerDetail" />
            <ComponentReference source="Customer/Tabs/ContactPerson.detail"
displayName="$ContactPerson" />
            <ComponentReference source="Customer/Tabs/Event.list" displayName="Events
" name="CustomerEvents" />
        </TabGroup>
    </Detail>
    <Overlay>
        <ComponentReference source="Customer/Overlay/AddEvent.list" name="AddToEvent"
/>
    </Overlay>
</MasterDetailComponent>
```

Explanation:

- Within `Actions` the available actions are defined.
- The `Master` element contains a reference to a List component. Depending on the selected list entry, further information is displayed in the detail view.
- The `Detail` element defines which information is displayed for a list item. In the above example, the detail information is displayed in a Tab Group.

9.5.3 Hide and display components

Areas of a Master/Detail component (Master structure element, Detail structure element and Overlay) can be hidden and displayed dynamically via data binding. This can be controlled via the `visible` attribute:

Component with dynamic hide/display

```
<CalendarComponent displayName="CalendarComp with visible attribute" visible="#visibi
lity">
```

Component reference with dynamic hide/display

```
<ComponentReference source="detail_1.detail" displayName="CompRef with visible
attribute" visible="#visibility"/>
```

For more information, see the section Navigation within a Web Application.

9.5.4 Master Structural Element

The `Master` structural element defines which information is to be displayed in the upper part of the screen using which component. The `Master` structural element controls which data set displays additional information in the `Detail` structural element.

If the user has not yet selected a list entry in the `Master` structural element, the first entry is automatically selected in the desktop view and the corresponding information is displayed in the `Detail` structural element.

The `Master` structural element is defined first within the master/detail component. The `Master` structural element must contain a list component, which is referenced here.

Sample Definition of a Master Structural Element

```
<MasterDetailComponent  xmlns="http://softproject.de/webapp/1.0">
    <Actions/>
    <Master>
    <!-- Reference to a List component-->
        <ComponentReference source="Customer/Customer.list" name="AllCustomers" />
    </Master>
    <Detail/>
    <Overlay/>
</MasterDetailComponent>
```

The Technical Process processing the data of the master/detail component is specified in the `process` attribute of the list component. The Technical Process must be contained in the `Services/Processes` folder. The path specification is relative to the `Services/Processes` folder.

9.5.5 Detail Structural Element

The `Detail` structural element displays additional information on the data set that is selected in the master structural element.

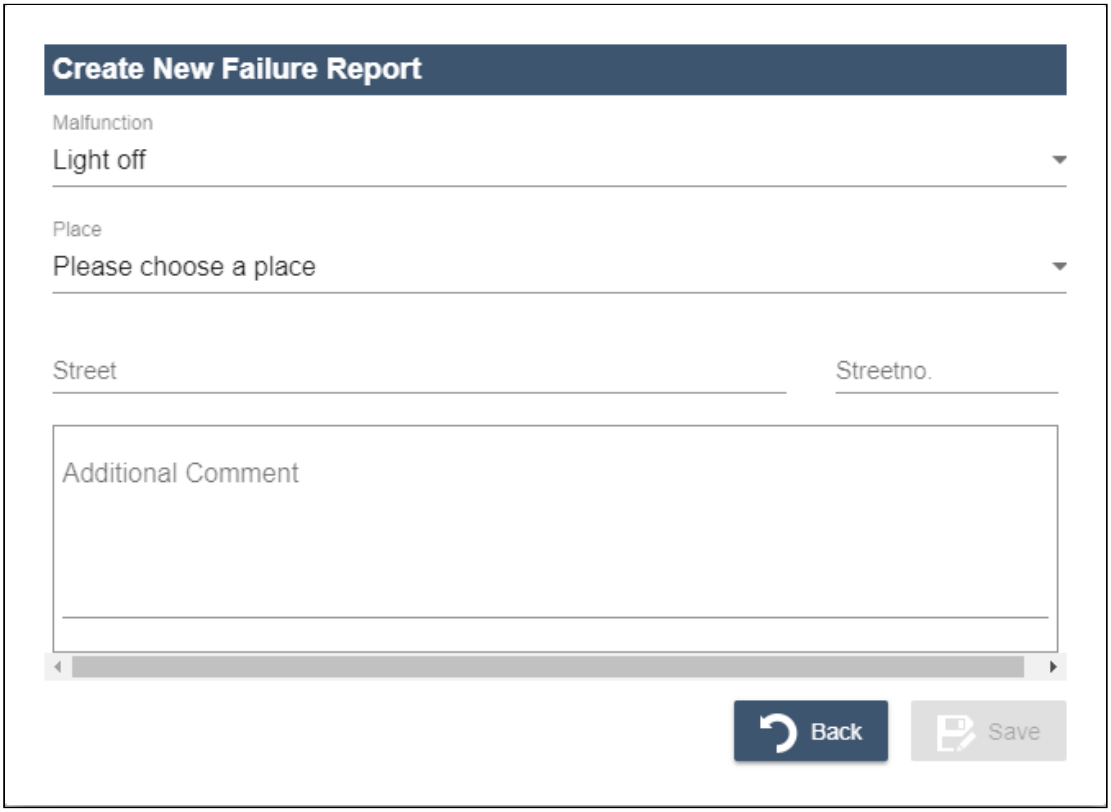
The `Detail` structural element is defined within `MasterDetailComponent` after the `Master` structural element. The components which are to be displayed are referenced here.

Sample Definition of a Detail Structural Element

```
<MasterDetailComponent xmlns="http://softproject.de/webapp/1.0">
    <Actions/>
    <Master/>
    <Detail>
    <!-- Reference to one or several Detail or List Components -->
        <ComponentReference source="Administration/Country/Country.detail" name="Coun
tryDetail" displayName="$General"/>
    </Detail>
    <Overlay/>
</MasterDetailComponent>
```

9.5.6 Overlay Structural Element

Within the MasterDetailComponent any number of overlay structural elements can be created, which can be defined with the following attributesThe content of the `Overlay` structural element is displayed as pop-up in the web application and lies above other elements. The `Overlay` structural element contains one more component that can display both static and dynamic content.



The `Overlay` structural element is defined within `MasterDetailComponent` after the structural elements `Master` and `Detail`. The `Overlay` structural element is only displayed if it is called using an action.

Within the `MasterDetailComponent` any number of `Overlay` elements can be created, which can be defined with the following attributes.

Attribute	Description
`width`	Width of the overlay ⓘ The specification of the `width` attribute is optional. If the attribute is not specified, the default width is used. **Possible value**: • Integer *> 0* for pixel specifications • Integer between *0* and *100* for percentage specifications
`height`	Height of the overlay ⓘ The specification of the `height` attribute is optional. If the attribute is not specified, the default width is used. **Possible value**: • Integer *> 0* for pixel specifications • Integer between *0* and *100* for percentage specifications
`units`	Defines the unit that applies to size specifications. **Possible values:** • `pixels` • `percents` (default)

⚠
- If the attributes `width` and `height` are filled with invalid values when specifying a percentage, a validation error will be output.
- If the attributes `width` and `height` are not set or specified with `0`, overlays are displayed in the default size.

ⓘ The attribute `enableOverlayBlurredBackground` within the Web App definition (`.wad`) can be used to specify whether the part of the Web App that is still visible in the background should be blurred or not in the case of overlays.

Sample definition

```
<MasterDetailComponent  xmlns="http://softproject.de/webapp/1.0">
    <Actions/>
    <Master/>
    <Detail/>
    <Overlay width="100" height="100" units="percents">
    <!-- Reference to a component in which the Overlay element is defined-->
        <ComponentReference source="Customers/AddToEvent.list" name="AddToEvent"></
ComponentReference>
    </Overlay>
    <Overlay width="200" height="300" units="pixels">
        <ComponentReference source="Customers/AddToEvent.list" name="DifferentEvent">
</ComponentReference>
    </Overlay>
</MasterDetailComponent
```

9.6 ExternalWeb component

An ExternalWeb component embeds an external web page in a Web App. An ExternalWeb component is created with the `<ExternalWebComponent>` element.

- If the host of the web page prohibits embedding, for example, by including an appropriate HTTP response header, that web page cannot be embedded in a Web App.
- Note that embedded websites may store and use cookies on the end device.

The following attributes can be defined for the `<ExternalWebComponent>` element:

Attribute	Description
`displayName`	Display name of the component in the menu. • Translatable • Data Binding possible **Possible values:** Any string
`path`	*Required*. Unique URL of the component, visible in the browser address bar. **Possible values:** String of alphanumeric characters (no umlauts, dots, etc.)
`allowForms`	Allows embedded content to submit forms. **Possible values:** • `true` • `false` (default)

Attribute	Description
`allowModals`	Allows embedded content to display modal dialogs. **Possible values:** • `true` • `false` (default)
`allowPointerLock`	Allows embedded content to interpret mouse movements directly as an input method. **Possible values:** • `true` • `false` (default)
`allowPopups`	Allows embedded content to open a web page in a new window or tab. **Possible values:** • `true` • `false` (default)
`allowSameOrigin`	Allows embedded content to share memory if the origin of the embedded content is the same as the origin of the host web application. **Possible values:** • `true` • `false` (default)
`allowScripts`	Allows embedded content to execute JavaScript. **Possible values:** • `true` • `false` (default)
`allowTopNavigation`	Allows embedded content to open a web page in the same window or tab. **Possible values:** • `true` • `false` (default)
`process`	Repository path to a Technical Process **Possible values:** Any path in the *X4* repository without leading slash (e.g. *`Project/Folder/Process.wrf`*)

Attribute	Description
`url`	Uniform Resource Locator (URL) with any valid protocol (`http://`, `https://`, etc.) **Possible values:** Any valid URL (e.g. `https://www.softproject.de/`)
`name`	Unique name that references the component. **Possible values:** Any string
`default`	Determines whether the view on a component is selected as default. **Possible values:** `true` / `false` (maximum one component may have the attribute `default="true"`)
`showInMenu`	Determines whether the external web component is visible in the navigation bar. **Possible values:** `true` / `false`

Input

If the parameter `process` is set, an XML file with the URL as input is expected in the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok
    url="URL"/>
```

9.6.1 Example

9.6.1.1 URL in ExternalWeb

The following example in the `.wad` creates an ExternalWeb component:

```
<ExternalWebComponent
    path="ExternalWebUrl"
    displayName="SoftProject Website"
    url="https://www.softproject.de" />
```

9.6.1.2 URL via Technical Process

The following example in the `.wad` creates an ExternalWeb component:

```
<ExternalWebComponent
    path="ExternalWebUrl"
    displayName="SoftProject Website"
    process="provideURL.wrf" />
```

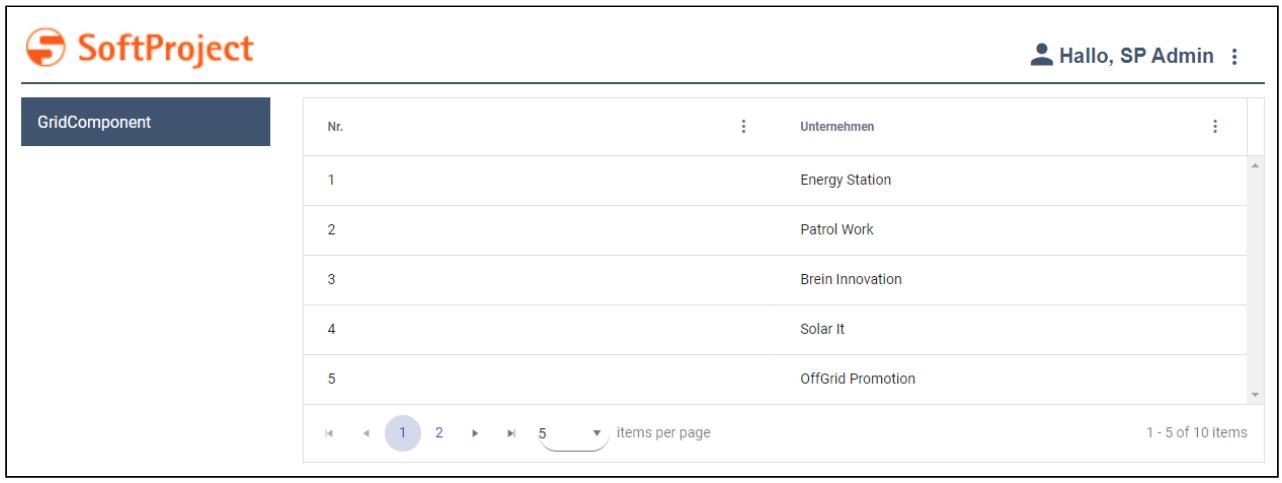
The `provideURL.wrf` Technical Process returns the following output, which sets the URL:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok
    url="https://www.softproject.de"/>
```

9.7 Grid component

A grid component is ideal for displaying data in a tabular display. In the grid settings, the behavior of the grid component can be defined by setting individual elements.

> The grid component will be optimized for being displayed on mobile devices in a future version of X4 BPMS.

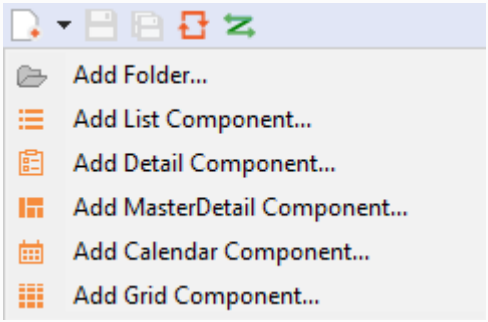


9.7.1 Defining Grid component

A grid component is declared within its own `<GridName>.grid` definition file. This is created in the **Components** project folder.

How to create a new definition file:

1. Click in the menu **New > Add Grid Component**
   . Alternatively, you can create the file using the **New > Add Grid Component** context menu.
2. Select the desired definition file.



3. In **File name** enter the name of the definition file.
4. Click **Finish**.
   A new definition file with a predefined structure is created.

The data to be displayed must be provided by a Technical Process and conform to a specific data model.

The data model provided by the Technical Process must be mapped in the Properties. To link the data provided by the Technical Process with the representation within the web application, the corresponding `Property` must be defined and stored via Data Binding in the GridColumn definition.

If a property is to be used in the grid component, it must be defined as a *`Complex`* type. This property can contain other properties.

The grid component is created via `<GridComponent>` and can have the following attributes:

Attribute	Description
`background`	Defines a background color. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`displayName`	Display name of the component in the menu. • Translatable • Data Binding possible **Possible values:** Any string
`fontFamily`	Defines the font family within the component. The property is inherited by all controls and actions of the component. **Possible values:** Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size within the component. The property is inherited by all controls and actions of the component. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g.*`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in point, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Keywords:*`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Defines the width of each character. The property is inherited by all controls and actions of the component. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the slant of the font. The property is inherited by all controls and actions of the component. **Possible values:** • `italic`: italic font • `normal`: normal font (default) • `oblique`: oblique font style (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. The property is inherited by all controls and actions of the component. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines the font color. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`headerBackground`	Defines the background color of the header ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`headerFontFamily`	Defines the font family of the header. **Possible values:** Font code from the font palette, e.g. *`Font04`*
`headerFontWeight`	Defines the font weight of the header. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`headerForeground`	Defines the font color of the header. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`iconColor`	Defines the color of the component's icon. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. *`A200`* A200. A200
`iconUrl`	Path to the image file or specification of a Material Icon that will be used as the component's icon. ⓘ • The image file must be contained in the `Resources` folder directly below the Web App project. Path specification relative to the `Resources` folder. • The material icon must be specified with the prefix `icon`, e.g. `icon:<MaterialIconName>`. **Possible values:** • String (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* ✓ With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons.
`itemsPerPage`	Defines the number of rows displayed per page. **Possible values:** • `5` (default) • `10` • `20` • `50` • `100`
`name`	Unique name that references the component. **Possible values:** Any string

Attribute	Description
`rowHighlightBackground`	Defines the background color of the selected row. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. Hexadecimal color value, e.g. `ff5a00` • Color code from the color palette of the Web App (see Theming), e.g. *`A200`* A200. A200
`rowHighlightForeground`	Defines the font color of the selected row. ⚠ This setting overrides the default color of the color scheme. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. Hexadecimal color value, e.g. `ff5a00` • Color code from the color palette of the Web App (see Theming), e.g. *`A200`* A200. A200
`process`	*Required*. Path to the `.wrf` file that provides the data for the component. The process must be contained in the `Services/Processes` folder. Path specification relative to the `Services/Processes` folder. **Possible values:** String (URI)
`title`	Title of the component. Displayed in the application as a header. **Possible values:** Any string

Attribute	Description
`titleBackground`	Defines a title background color. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`titleForeground`	Defines a color for the title foreground. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash before the color value. • Do not use abbreviated notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`visible`	Defines the visibility. • Data Binding (Boolean) possible. **Possible values:** Boolean or string for Data Binding

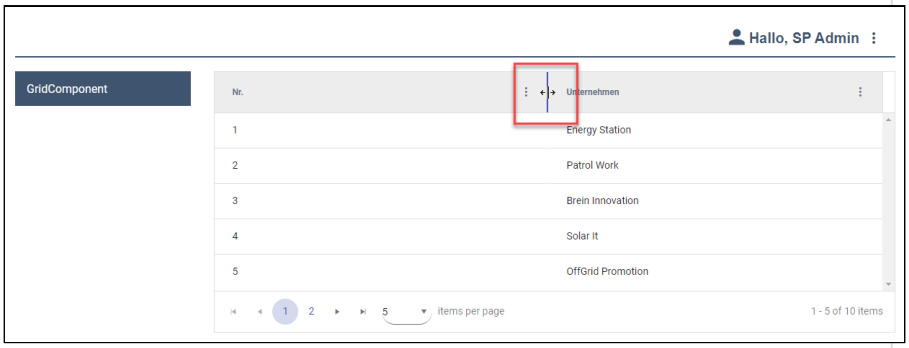
9.7.2 Sample of a grid component

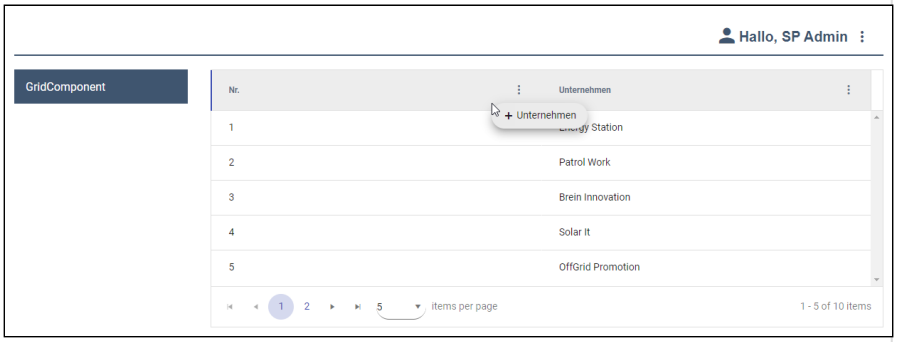
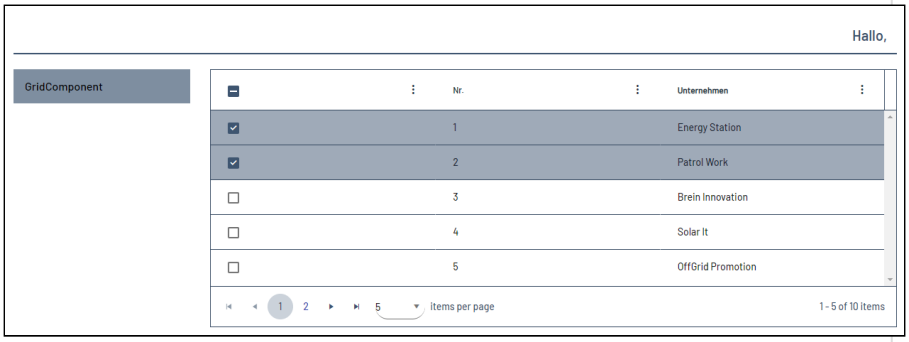
```
<?xml version="1.0" encoding="UTF-8"?>
<GridComponent xmlns="http://softproject.de/webapp/1.0" process="loadGrid.wrf" >
    <Properties>
        <Property name="list" type="Complex">
            <Property name="id" type="Integer" displayName="Nr." />
            <Property name="company" type="String" displayName="Unternehmen" />
        </Property>
    </Properties>
    <GridSettings>
        <Resize/>
        <Reorder/>
        <ColumnVisibility/>
        <Filtering/>
        <Multiselect checkboxSelection="true"/>
        <Autoselection/>
    </GridSettings>
    <GridColumns>
        <GridColumn value="#list.id" sortable="true"/>
        <GridColumn value="#list.company"/>
    </GridColumns>
</GridComponent>
```

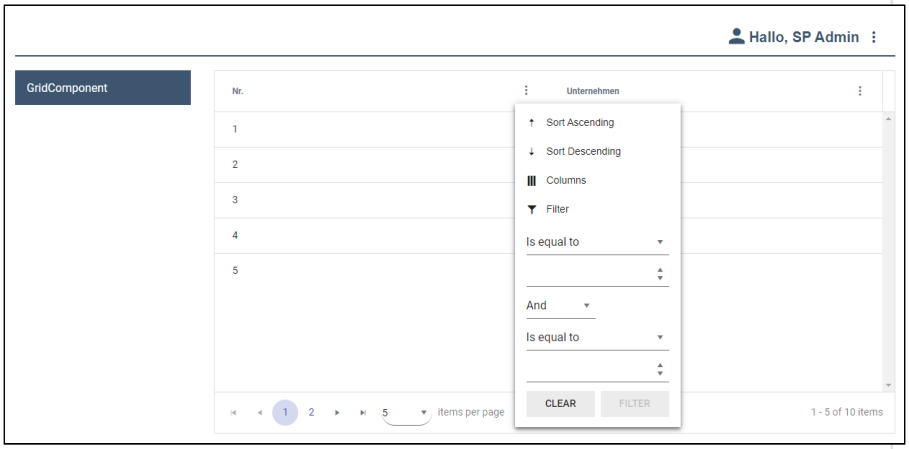
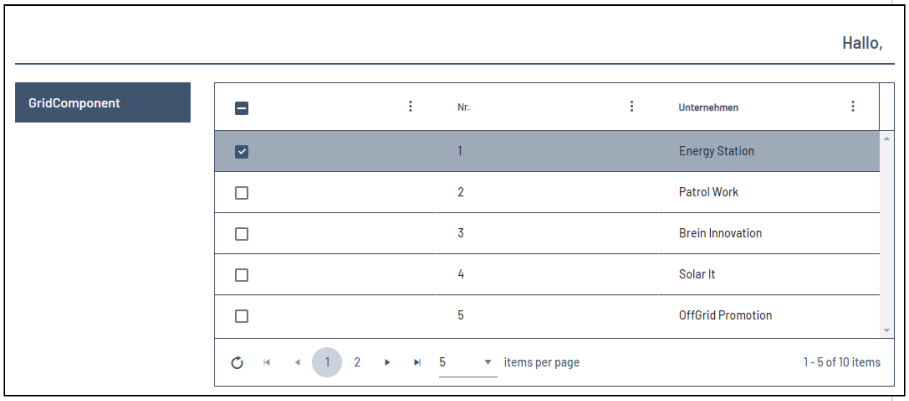
9.7.3 Grid settings

Within the `GridSettings` element, the elements listed below can be declared to define the behavior of the grid component:

Possible elements

Element	Description
`Resize`	The `Resize` element can be used to click on the transitions of the column headers and to change the width of the respective columns by holding down the mouse button. 

Element	Description
`Reorder`	With the `Reorder` element the column headers can be clicked and moved to another position by drag and drop. 
`Multiselect`	The `Multiselect` element allows multiple entries to be selected. When using the `Multiselect` element, the `checkboxSelection` attribute can be set and contain the following values: **Possible values:** `true` / `false` 
`ColumnVisibility`	The `ColumnVisibility` element can be used to show and hide individual columns.

Element	Description
`Filtering`	The `Filtering` element can be used to define conditions according to which filtering is to be performed. 
`Autoselection`	The `Autoselection` element defines that the first row of the table is automatically selected when the grid component is opened. 
`Editing`	The `Editing` element has a function only if the `editable="true"` attribute has been set in the `GridColumn` element. When using the `Editing` element, the `reloadOnSave` attribute can be set and contain the following values: **Possible values:** • `true`: When saving using a **Save Action**, the Grid Component is reloaded. • `false`: When saving using a `Save Action`, the Grid Component is not reloaded.

9.7.4 GridColumn definition

Within the grid component the columns have to be defined. The column definition is done via `GridColumn`. All columns must be defined within `GridColumns`.

Possible attributes:

Attribute	Description
`horizontalAlign`	Defines the horizontal alignment of the content within the column. **Possible values:** • `left`: content left aligned • `center`: Content centered • `right`: content right aligned
`format`	Defines the date format including minutes, seconds or milliseconds. **Möglche Werte:** • `Datetime with Minutes`: The date has the format hh:mm. (default) • `Datetime with Seconds`: The date has the format hh:mm:ss. • `Datetime with Milliseconds`: The date has the format hh:mm:ss.sss.
`sortable`	Defines whether sorting can be performed according to the contents of the column. More information about Sorting. **Possible values:** `true` / `false`
`editable`	Defines whether the contents of the column are editable. **Possible values:** `true` / `false` (default)
`value`	Defines the content of the column by data binding. **Possible values:** String (Data Binding) ⓘ If no attribute `displayName` is assigned for the corresponding property, then the data binding expression is used as the column header.

Attribute	Description
`width`	Defines the column width. **Possible values:** • Integer ⓘ The unit in which the column width is specified is defined with the `widthUnits` attribute.
`widthUnits`	Defines the unit in which the column width `width` is specified. **Possible values:** • `pixels`: The column width is specified in pixels. • `percents`: The column width is specified in percent.
`textOverflow`	Specifies what should happen if overflowing texts cannot be displayed. **Possible values:** • *`ellipsis`*: With `...` show that the text is not finished. A text break is prevented. • *`hidden`*: (default if `layout="optimized"`) Break text, watch for whole words. Text wrapping is prevented. • *`wordBreak`*: (default if `layout="fixed"`) Break within word. • `allow`: Break text between words The attribute in `<GridColumn>` has a higher priority and overrides the attribute in `<GridComponent>`.

9.7.5 Periodical reloading

The `ReloadTimer` element can be used to periodically reload the grid component.

⚠ Periodical reloading of data can cause a high load and should only be used in rare use cases.

Possible attributes:

Attribute	Description
`refreshTime`	Time until reload in seconds • Data Binding (Integer) possible **Possible values:** Any integer

Attribute	Description
`itemIdentifier`	*Optional*; Unique key for the data represented in the rows. • The attribute can be used to retrieve rows during reloading and improve the reloading process. • If the attribute is not specified, a string comparison on a record is used to identify the row. **Possible values:** Data Binding to a defined Property of the grid component

9.7.6 "Select" action within the grid component

The **Select** action is created via the `<SelectAction/>` element within `<Actions>` and is triggered as soon as an entry is selected from a grid component.

9.7.7 "Save" action within the grid component

If the action **Save** is defined, the Technical Process receives the data objects of the edited rows with a new index attribute containing the index of the row. The numbering starts from `0`.

The action Save is created with the element `<SaveAction/>` within `<Actions>`. The action **Save** saves entered data. The status depends on the validation status.

9.7.8 Paging

Web Apps created with X4 Web Apps automatically support paging in the grid component. Therefore, the processes that provide data for these components also have to support paging and process information about how many objects are to be read from which area.

9.7.8.1 Client-side paging

A high number of data sets to be displayed requires high performance for processing.
If more than 10,000 records are to be displayed, it is recommended to switch to server-side paging.

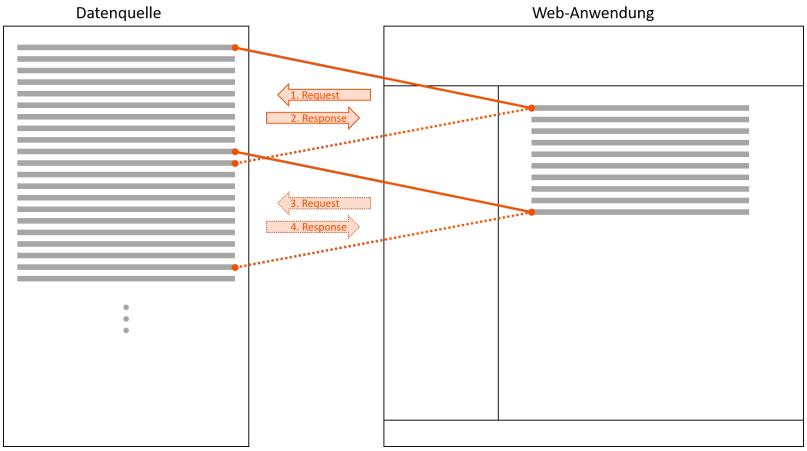
With client-side paging, the browser performs the paging. This corresponds to the default behavior of the grid component.

The data must be provided by the Technical Process in the following form:

```
<OkList>
    <List>
        ...
    </List>
</OkList>
```

9.7.8.2 Server-side paging

With paging, data is only loaded in subsets from the data source step by step. This avoids large data streams that are very difficult to process by the data source or the browser. Unlike conventional Web applications, Web applications created with X4 Activities Web Apps make it possible to reload the new data while scrolling. The user does not have to explicitly turn to the next page.



The response that is returned by the Technical Process must contain information on how many objects are contained in the list (attribute `size`) in addition to the requested objects, so that it is clear if further objects need to be queried.

The data must be provided by the Technical Process in the following form:

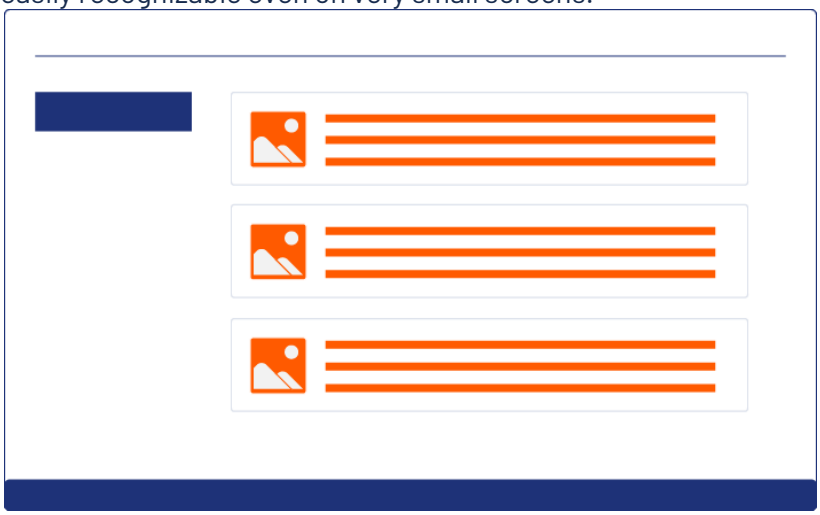
```
<OkList size="">
    <List>
        ...
    </List>
</OkList>
```

Attribute	Description
`size`	The total number of records in the data source. **Possible values:** Integer

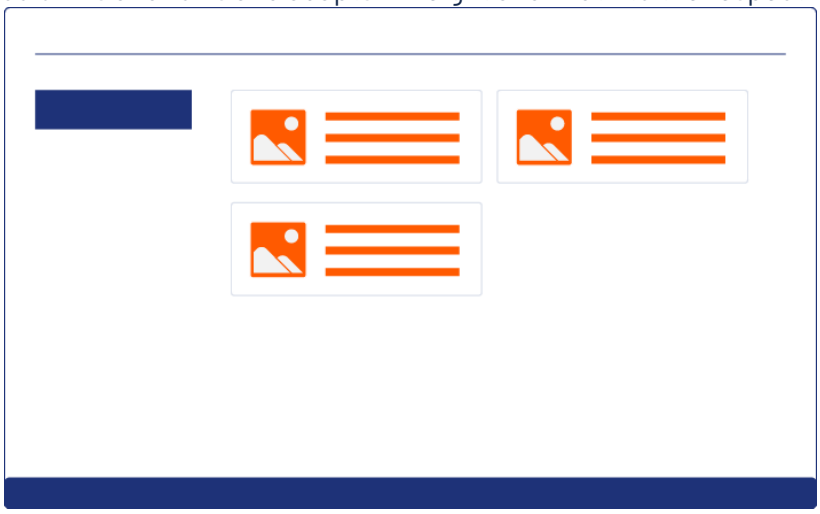
10 Layouts

Various layouts can be specified for the components within a web application:

- Flow Layout
  With a flow layout, the elements are arranged automatically. The individual elements flow freely across the page, similar to text. This layout is very suitable for web applications that should be easily recognizable even on very small screens.

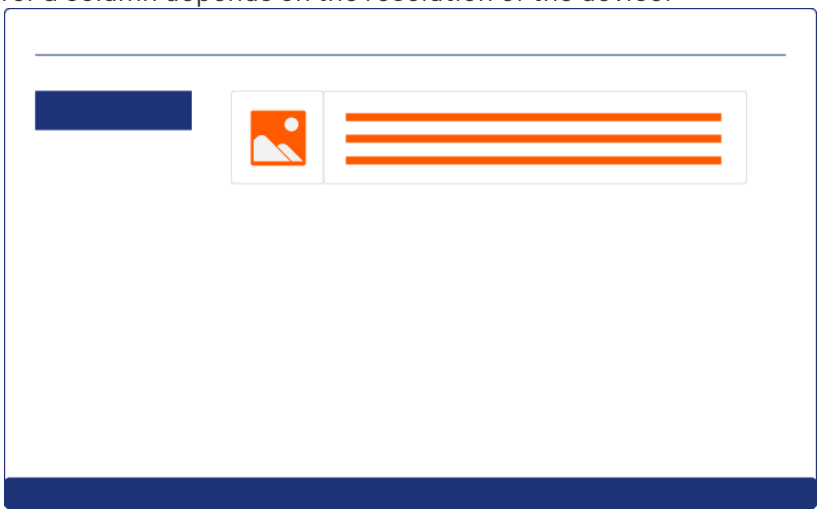


- Grid Layout
  When the grid layout is chosen for the elements of a Web App, the elements are aligned to a grid. This layout provides a very orderly appearance. The grid is defined by the number of columns and rows and adapts in height and width to the respective end device.



- Tab Layout

- Responsive Layout
  The Responsive Layout is ideal for web applications that are designed to be displayed on different devices. The Responsive Layout is divided into 12 virtual columns. The available space for a column depends on the resolution of the device.



- Box Layout
  The Box layout is a simple layout with only one content element. Height, width, border, margin and padding can only be defined for the box layout.

The default layout controls how the controls (text, radio buttons, checkboxes, images, etc.) are displayed. The different layouts can be nested within each other. Layouts can contain different Controls.

The layout is declared within a component:

```
<DetailComponent>
    <FlowLayout>
        <!-- Content -->
    </FlowLayout>
</DetailComponent>
```

10.1 Box Layout

The Box layout is a simple layout with only one content element. Height, width, border, margin and padding can only be defined for the box layout.

The `<BoxLayout>` element creates a box layout.

For further information please visit the section

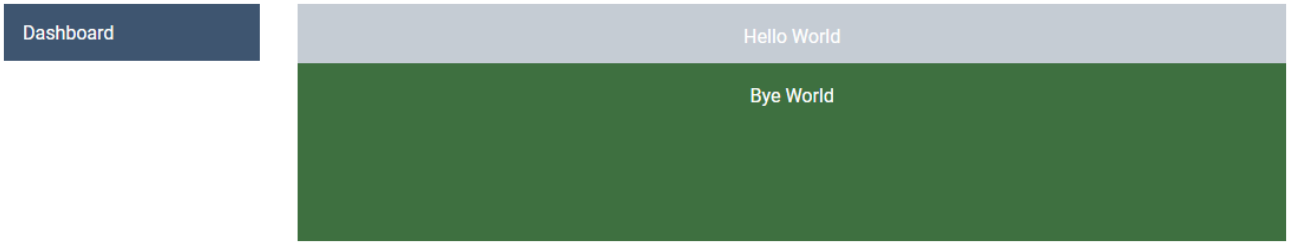
- <BoxLayout>

10.1.1 Examples

10.1.1.1 BoxLayout in combination with FlowLayout

```
<DetailComponent
    default="true"
    displayName="Dashboard"
    path="Dashboard">
    <BoxLayout
        background="50"
        foreground="A100">
        <FlowLayout>
            <BoxLayout
                background="100"
                height="50"
                units="pixels">
                <Label value="Hello World" />
            </BoxLayout>
            <BoxLayout
                background="10"
                height="150"
                units="pixels"
                horizontalAlign="center">
                <Label value="Bye World" />
            </BoxLayout>
        </FlowLayout>
    </BoxLayout>
</DetailComponent>
```

Desktop view



10.1.2 <BoxLayout>

The following attributes can be defined for the `<BoxLayout>` element:

Attribute	Description
`background`	Defines the background color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash (#) in front of the color value! • Do not use a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`border`	Defines if a border is displayed around the layout. **Possible values:** *`true`* / *`false`*
`border-left`	Defines whether a left border is displayed **Possible values:** *`true`* / *`false`*
`border-right`	Defines whether a right border is displayed **Possible values:** *`true`* / *`false`*
`border-top`	Defines whether a top border is displayed **Possible values:** *`true`* / *`false`*

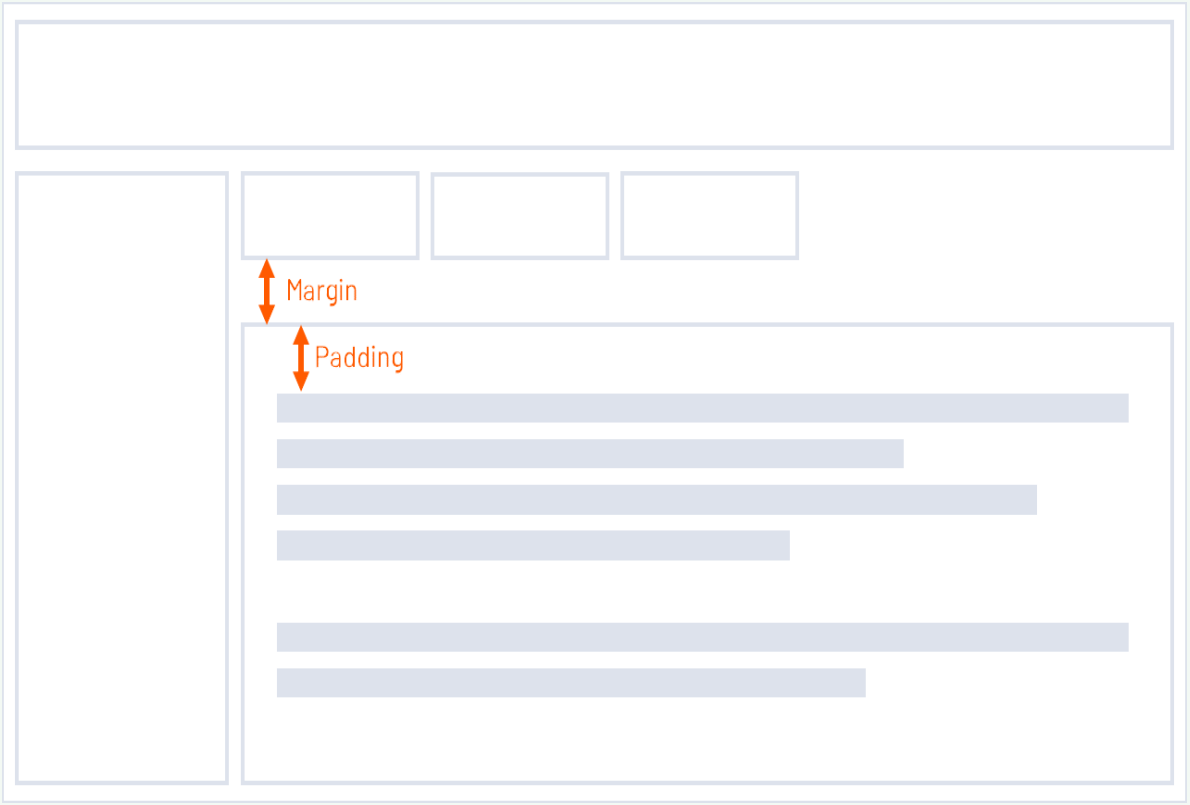
Attribute	Description
`border-bottom`	Defines whether a bottom border is displayed **Possible values:** *`true`* / *`false`*
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default width of the characters. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default weight of the characters. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash (#) in front of the color value! • Do not use a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`height`	Height of the layout **Possible values:** Integer

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`paddingBottom`	Lower inner spacing **Possible values:** Any numerical values
`paddingLeft`	Left inner spacing **Possible values:** Any numerical values
`paddingRight`	Right inner spacing **Possible values:** Any numerical values
`paddingTop`	Upper inner spacing **Possible values:** Any numerical values
`paddingUnits`	Units for the inner spacing **Possible values:** • `pixels` • `percents`
`marginBottom`	Lower outer spacing **Possible values:** Any numerical values
`marginLeft`	Left outer spacing **Possible values:** Any numerical values
`marginRight`	Right outer spacing **Possible values:** Any numerical values
`marginTop`	Upper outer spacing **Possible values:** Any numerical values

Attribute	Description
`marginUnits`	Units for the outer spacing **Possible values:** • `pixels` • `percents`
`textOverflow`	Defines what happens if the page is full. **Possible values:** • `ellipsis` (default): Show with `...` that the text is not finished • `hidden`: Break off text, paying attention to whole words • `wordBreak`: Break off within the word • `allow`: Break off text between words
`units`	Specifies the unit that applies to the `height` and `width` size specifications. **Possible values:** • `pixels` (default) • `percents`
`visible`	Defines if the layout is visible. • Data binding (boolean) possible. **Mögliche Werte:** Boolean or data binding expression
`width`	Defines the width of the layout. **Possible values:** Integer

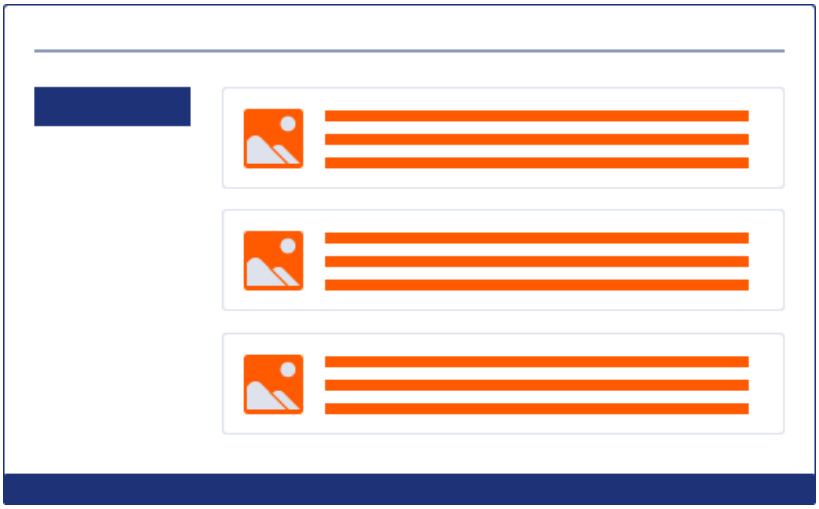
Padding is the internal spacing, which is the distance between the content (e.g. text) and the frame of the element.
Margin is the outer distance, which is the distance between the element and the neighboring elements.



10.2 Flow Layout

With a flow layout, the elements are arranged automatically. The individual elements flow freely across the page, similar to text. This layout is very suitable for web applications that should be easily recognizable even on very small screens.

The `<FlowLayout>` element creates a flow layout.

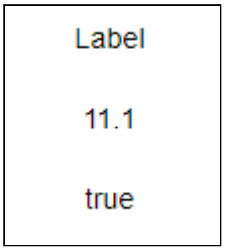


For further information please visit the section

- <FlowLayout>

10.2.1 Examples

```
<FlowLayout>
    <Label value="Label"/>
    <Label value="1"/>
    <Label value="true"/>
</FlowLayout>
```



10.2.2 <FlowLayout>

The following attributes can be defined for the `<FlowLayout>` element:

Attribute	Description
`direction`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`horizontal`* • *`vertical`* (default)
`background`	Defines the background color. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Activates the layout • Data binding (boolean) possible **Possible values:** `true`/`false` or data binding expression
`fontFamily`	Defines the font family. ⓘ • This attribute overrides the default font of the Web App for this layout. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this layout. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *20*; *20.8*; *.9* • Font size in pixels, e.g. *20px* • Font size in points, e.g. *18pt* • Font size compared to the font size of the parent element, e.g. *.8em* or *120* • Key words: *xx-Small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*, *smaller*, *larger*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App for this layout. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font style. This attribute overrides the default style of the characters of the Web App for this layout. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this layout. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines a color for the foreground (texts etc.) of the layout. ⓘ • This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`textOverflow`	Defines what happens if the layout is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`* (default): Break off text between words

Attribute	Description
`visible`	Defines if the layout is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

If `direction="vertical"`, the elements are arranged one above the other from top to bottom. A scroll bar appears if the elements are higher than the layout.

10.3 Grid Layout

When the grid layout is chosen for the elements of a web application, the elements are aligned to a grid. This layout provides a very orderly appearance. The grid is defined by the number of columns and rows and adapts in height and width to the respective end device.

The `<GridLayout>` element creates a grid layout. When displaying the grid layout on smaller screens (smartphones, tablets), the number of columns remains the same, the columns are just displayed narrower.

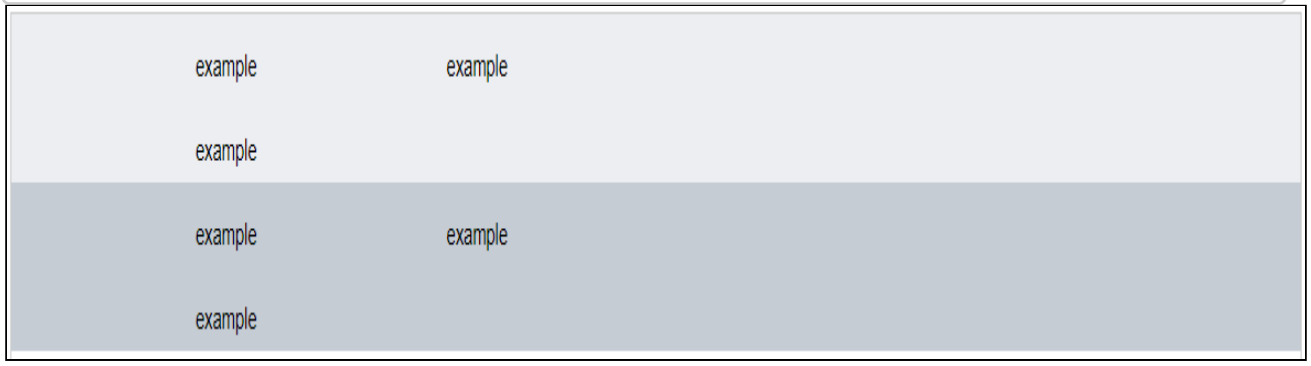
For further information please visit the section

- <GridLayout>
- <ColumnDefinitions>
- <Column>

10.3.1 Examples

Example Grid Layout

```
<GridLayout columns="2" alternatingRows="true" border="true">
	<ColumnDefinitions>
		<Column width="1" />
		<Column width="2" />
	</ColumnDefinitions>
	<FlowLayout>
		<Label value="example" />
		<Label value="example" />
	</FlowLayout>
	<Label value="example" />
	<FlowLayout>
		<Label value="example" />
		<Label value="example" />
	</FlowLayout>
	<Label value="example" />
</GridLayout>
```



10.3.2 <GridLayout>

The following attributes can be defined for the `<GridLayout>` element:

Attribute	Description
`alternatingRows`	Defines whether the rows have different, alternating background colors. **Possible values:** *true* / *false*
`columns`	Defines how many columns the layout grid consists of. **Possible values:** Integer (default: *3*)

Attribute	Description
`rows`	Defines how many lines the layout grid consists of. **Possible values:** Integer
`background`	Defines the background color. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Activates the layout • Data binding (boolean) possible **Possible values:** `true`/`false` or data binding expression
`fontFamily`	Defines the font family. ⓘ • This attribute overrides the default font of the Web App for this layout. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this layout. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *20*; *20.8*; *.9* • Font size in pixels, e.g. *20px* • Font size in points, e.g. *18pt* • Font size compared to the font size of the parent element, e.g. *.8em* or *120* • Key words: *xx-Small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*, *smaller*, *larger*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App for this layout. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font style. This attribute overrides the default style of the characters of the Web App for this layout. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this layout. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines a color for the foreground (texts etc.) of the layout. ⓘ • This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*

Attribute	Description
`textOverflow`	Defines what happens if the layout is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the layout is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

For further information please visit the section

- <ColumnDefinitions>
- <Column>

10.3.3 <ColumnDefinitions>

The `GridLayout` element can contain the `ColumnDefinitions` element. `ColumnDefinitions` can be used to define the width of each column in the grid layout:

Example ColumnDefinition

```
<GridLayout alternatingRows="false">
    <ColumnDefinitions>
        <Column width="2"/>
        <Column width="2"/>
        <Column width="1"/>
    </ColumnDefinitions>
    ...
</GridLayout>
```

If the width of the columns is defined using the column definition, then it is not necessary to specify a number of columns for the layout. The number of columns is derived from the column definition. In the above example, three columns are displayed because the `Column` element exists three times. The first two columns are twice as wide as the third column.

For further information please visit the section

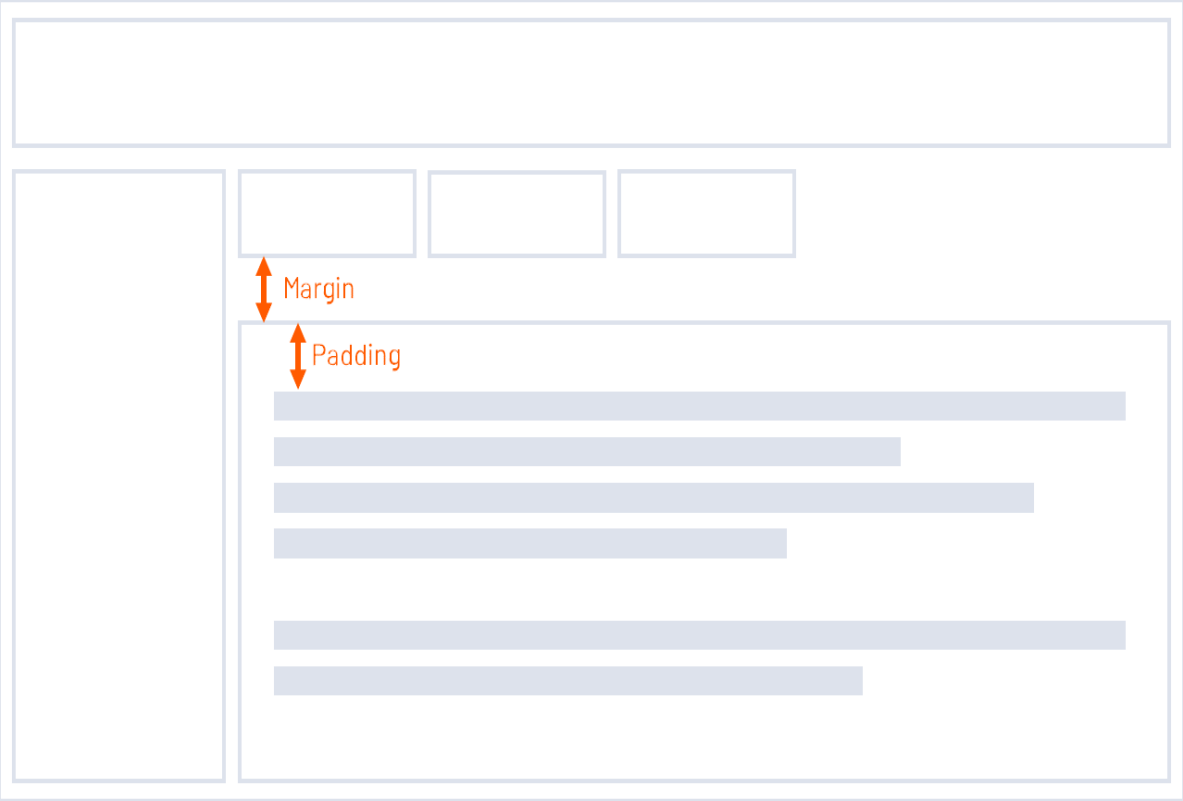
- <GridLayout>
- <Column>

10.3.4 <Column>

The following attributes can be defined for the `<Column>` element:

Attribute	Description
`horizontalAlign`	Positioning within the column **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`paddingBottom`	Lower inner spacing **Possible values:** Any numerical values
`paddingLeft`	Left inner spacing **Possible values:** Any numerical values
`paddingRight`	Right inner spacing **Possible values:** Any numerical values
`paddingTop`	Upper inner spacing **Possible values:** Any numerical values
`paddingUnits`	Units for the inner spacing **Possible values:** • `pixels` • `percents`

Padding is the internal spacing, which is the distance between the content (e.g. text) and the frame of the element.
Margin is the outer distance, which is the distance between the element and the neighboring elements.



For further information please visit the section

- <GridLayout>
- <ColumnDefinitions>

10.4 Tab Group

To use tabs within a Master/Detail-Component, the `<TabGroup>` element can be used in master, detail and overlay structure elements. The `<TabGroup>` element contains one or more components, each of them is displayed in a tab.

For further information please visit the section

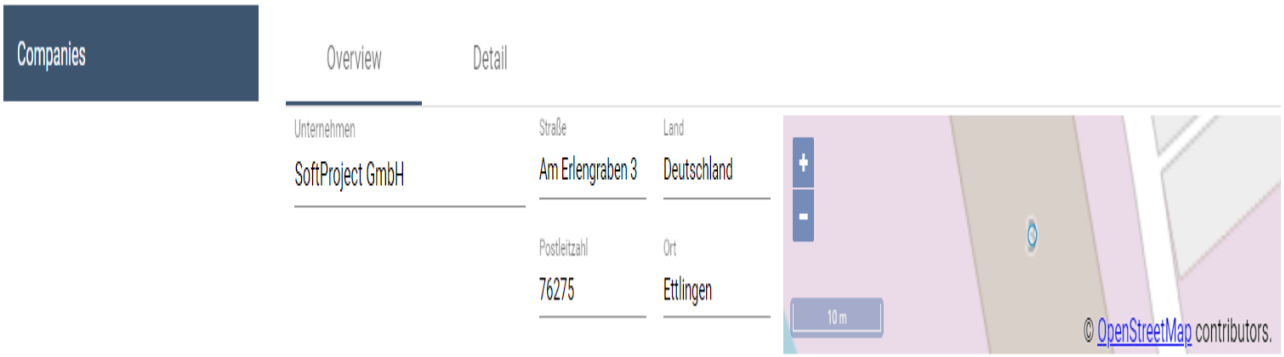
- <TabGroup>

10.4.1 Examples

```
<MasterDetailComponent
    default="true"
    path="Companies"
    displayName="Companies">
    <Master defaultComponentName="overview">
        <TabGroup>
            <DetailComponent
                displayName="Overview"
                name="overview">
                <ResponsiveLayout defaultCellSizeMobile="6">
                    <Cell cellSizeDesktop="3">
                        <FlowLayout>
                            <TextBox
                                displayName="Unternehmen"
                                value="SoftProject GmbH" />
                        </FlowLayout>
                    </Cell>
                    <Cell cellSizeDesktop="3">
                        <GridLayout columns="2">
                            <TextBox
                                displayName="Straße"
                                value="Am Erlengraben 3" />
                            <TextBox
                                displayName="Land"
                                value="Deutschland" />
                            <TextBox
                                displayName="Postleitzahl"
                                value="76275" />
                            <TextBox
                                displayName="Ort"
                                value="Ettlingen" />
                        </GridLayout>
                    </Cell>
                    <Cell
                        cellSizeDesktop="6"
                        cellSizeTablet="12">
                        <Map vendor="OpenStreetMap">
                            <SingleMarker
                                name="Marker"
                                searchString="Am Erlengraben 3, 76275 Ettlingen" />
                        </Map>
                    </Cell>
                </ResponsiveLayout>
            </DetailComponent>
            <DetailComponent
                displayName="Detail"
                name="detail">
                <FlowLayout>
                    <!-- Content -->
                </FlowLayout>
```

```
            </DetailComponent>
        </TabGroup>
    </Master>
    <Detail>
        <!-- Content -->
    </Detail>
</MasterDetailComponent>
```

10.4.1.1 Desktop view



10.4.2 <TabGroup>

The following attributes can be defined for the `<BoxLayout>` element:

Attribute	Description
`background`	Defines the background color. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`description`	Description text of the component. Displayed in each tab above the respective component. • Data binding possible **Possible values:** Any string

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this layout. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this layout. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* oder *`120%`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Defines the width of the single characters. This attribute overrides the default width of the characters for this layout. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`

Attribute	Description
`fontStyle`	Defines the font inclination. This attribute overrides the default inclination of the characters for this layout. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)
`fontWeight`	Defines the font weight. This attribute overrides the default weight of the characters for this layout. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines a color for the foreground (texts etc.) of the layout. • This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`title`	Title of the group • Data binding possible **Possible values:** Any string
`subtitle`	Subtitle of the group • Data binding possible **Possible values:** Any string

10.5 Tab layout

You can use the tab layout to arrange other layouts (flow layout or grid layout) on tabs. This layout is well suited for providing a lot of information that can be arranged thematically, such as forms.

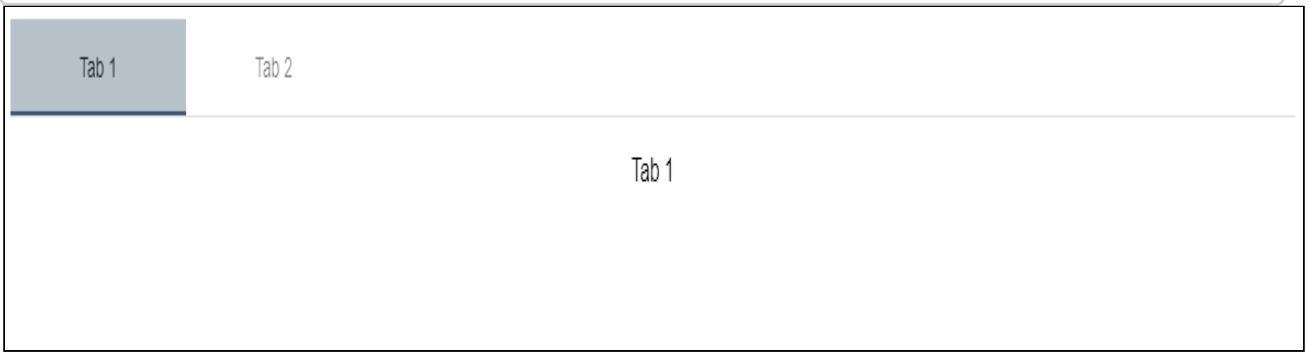
> The tab layout can be used both in components and within other layouts.

A tab layout is created with a `<TabLayout>` element. A tab layout must contain at least one `<Tab>` element.

> A `<Tab>` element must contain exactly one `<FlowLayout>` element or a `<GridLayout>` element.

10.5.1 Examples

```
<TabLayout title="Tab layout">
	<Tab displayName="Tab 1">
		<FlowLayout>
			<Label value="tab 1" />
		</FlowLayout>
	</Tab>
	<Tab displayName="Tab 2">
		<GridLayout>
			<Label value="tab 1" />
			<Label value="tab 2" />
			<Label value="tab 3" />
		</GridLayout>
	</Tab>
</TabLayout>
```



10.5.2 <TabLayout>

The following attributes can be defined for the `<TabLayout>` element:

Attribute	Description
`background`	Defines the background color. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`enabled`	Activates the layout • Data binding (boolean) possible **Possible values:** `true`/`false` or data binding expression
`fontFamily`	Defines the font family. ⚠ • This attribute overrides the default font of the Web App for this layout. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. ⚠ This attribute overrides the default font size of the Web App for this layout. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* oder *`120%`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⚠ This attribute overrides the default width of the characters of the Web App for this layout. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font style. ⚠ This attribute overrides the default style of the characters of the Web App for this layout. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. ⚠ This attribute overrides the default font weight of the Web App for this layout. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines a color for the foreground (texts etc.) of the layout. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*

Attribute	Description
`textOverflow`	Defines what happens if the layout is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`* (default): Break off text between words
`title`	Title of the tab layout is displayed as header. • Translatable • Data binding possible **Possible values:** Any string
`visible`	Defines if the layout is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

10.5.3 <Tab>

The following attributes can be defined for the `<Tab>` element:

Attribute	Description
`background`	Defines the background color. ⚠ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`displayName`	Displayed name for the tab • Translatable • Data binding possible **Possible values:** Any string

Attribute	Description
`enabled`	Activates the layout • Data binding (boolean) possible **Possible values:** `true`/`false` or data binding expression
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this layout. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this layout. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this layout. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this layout. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this layout. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines a color for the foreground (texts etc.) of the layout. ⓘ • This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*

Attribute	Description
`textOverflow`	Defines what happens if the layout is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the layout is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

10.6 Responsive Layout

The Responsive Layout is ideal for web applications that are designed to be displayed on different devices. The Responsive Layout is divided into 12 virtual columns. The available space for a column depends on the resolution of the device.

The `<ResponsiveLayout>` element creates a Responsive layout. At least one `<Cell>` element must be defined inside the ResponsiveLayout `<ResponsiveLayout>`. For each `<Cell>` element, you can define the number of columns that the element will take on the corresponding device. Additional layout types can be nested in the `<Cell>` element. In Responsive layout, the content within the `<Cell>` element is adjusted to the available screen space and wrapped if necessary.

For further information please visit the section

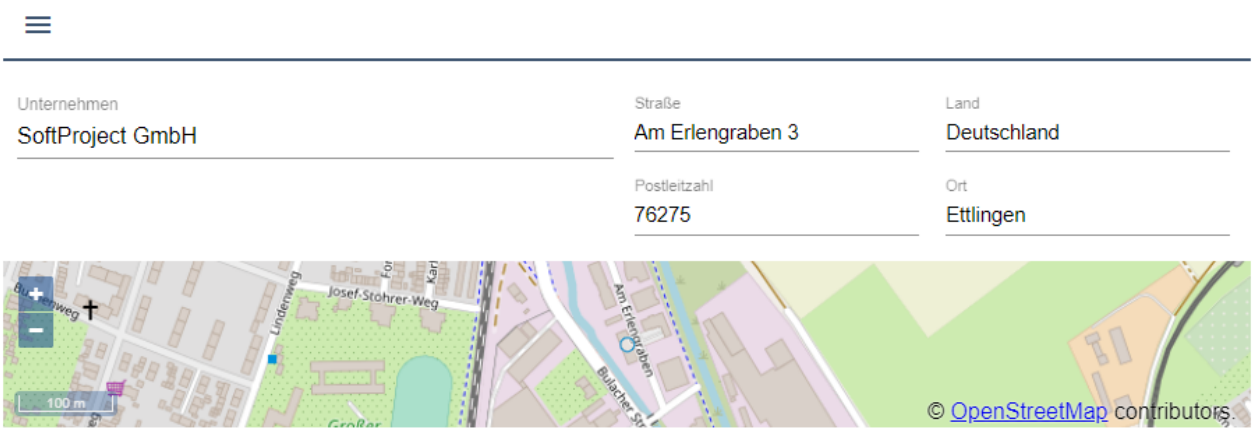
- <ResponsiveLayout>
- <Cell>

10.6.1 Examples

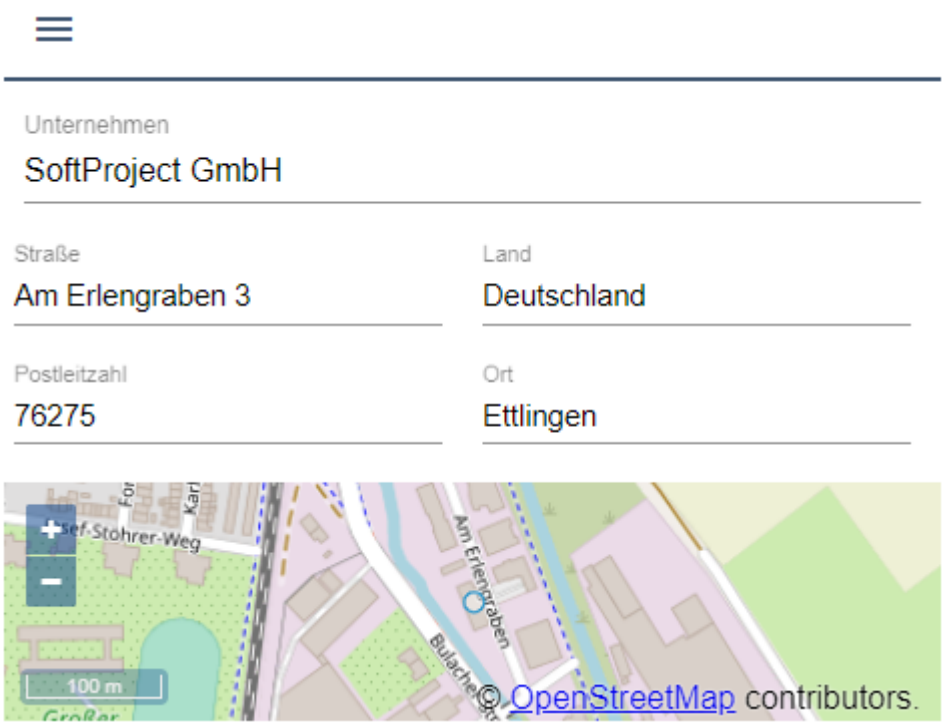
```
<DetailComponent
    default="true"
    displayName="Dashboard"
    path="Dashboard">
    <ResponsiveLayout defaultCellSizeTablet="6">
        <Cell cellSizeTablet="3">
            <FlowLayout>
                <TextBox
                    displayName="Unternehmen"
                    value="SoftProject GmbH" />
            </FlowLayout>
        </Cell>
        <Cell cellSizeDesktop="3">
            <GridLayout columns="2">
                <TextBox
                    displayName="Straße"
                    value="Am Erlengraben 3" />
                <TextBox
                    displayName="Land"
                    value="Deutschland" />
                <TextBox
                    displayName="Postleitzahl"
                    value="76275" />
                <TextBox
                    displayName="Ort"
                    value="Ettlingen" />
            </GridLayout>
        </Cell>
        <Cell
            cellSizeDesktop="6"
            cellSizeTablet="12">
            <Map vendor="OpenStreetMap">
                <SingleMarker
                    name="Marker"
                    searchString="Am Erlengraben 3, 76275 Ettlingen" />
            </Map>
        </Cell>
    </ResponsiveLayout>
</DetailComponent>
```

Desktop view

Tablet view



Mobile view



10.6.2 <ResponsiveLayout>

The following attributes can be defined for the `<ResponsiveLayout>` element:

Attribute	Description
`defaultCellSizeDesktop`	Defines the number of columns that the `<Cell>` element takes by default when displayed on desktop computers. **Possible values:** Integers between 1-12 (default: `4`) ⓘ The default defined here will be overridden when the `cellSizeDesktop` attribute is defined in the `<Cell>` child element.
`defaultCellSizeTablet`	Defines the number of columns that the `<Cell>` element takes by default when displayed on tablets. **Possible values:** Integers between 1-12 (default: `6`) ⓘ The default defined here will be overridden when the `cellSizeTablet` attribute is defined in the `<Cell>` child element.
`defaultCellSizeMobile`	Defines the number of columns that the `<Cell>` element takes by default when displayed on smartphones. **Possible values:** Integers between 1-12 (default: `12`) ⓘ The default defined here will be overridden when the `cellSizeMobile` attribute is defined in the `<Cell>` child element.
`background`	Defines the background color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash (#) in front of the color value! • Do not use a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. This attribute overrides the default font size of the Web App. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* oder *`120%`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`

Attribute	Description
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default width of the characters. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default weight of the characters. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash (#) in front of the color value! • Do not use a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • `ellipsis` (default): Show with `...` that the text is not finished • `hidden`: Break off text, paying attention to whole words • `wordBreak`: Break off within the word • `allow`: Break off text between words
`visible`	Defines if the layout is visible. • Data binding (boolean) possible. **Mögliche Werte:** Boolean or data binding expression

The `<ResponsiveLayout>` element has to contain the following elements:

- `Cell`

For further information please visit the section

- <Cell>

10.6.3 <Cell>

The `<Cell>` element is used to create a content area in the `<ResponsiveLayout>` element.

The following attributes can be defined for the `<Cell>` element:

Attribute	Description
`cellSizeDesktop`	Defines the number of columns that the `<Cell>` element takes when displayed on desktop computers. **Possible values:** Integers between 1-12 The value defined here overrides the`defaultCellSizeDesktop`attribute in the `<ResponsiveLayout>` parent element.

Attribute	Description
`cellSizeTablet`	Defines the number of columns that the `<Cell>` element takes when displayed on tablets. **Possible values:** Integers between 1-12 ⓘ The value defined here overrides the `defaultCellSizeTablet` attribute in the `<ResponsiveLayout>` parent element.
`cellSizeMobile`	Defines the number of columns that the `<Cell>` element takes when displayed on smartphones. **Possible values:** Integers between 1-12 ⓘ The value defined here overrides the `defaultCellSizeMobile` attribute in the `<ResponsiveLayout>` parent element.
`background`	Defines the background color. This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash (#) in front of the color value! • Do not use a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`fontFamily`	Defines the font family. ⓘ • This attribute overrides the default font of the Web App. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. ⓘ This attribute overrides the default font size of the Web App. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* oder *`120%`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default width of the characters. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default weight of the characters. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ • Do not use a hash (#) in front of the color value! • Do not use a shortened notation of the color value. • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`minHeight`	Minimum height of the layout **Possible values:** Integer

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • `ellipsis` (default): Show with `...` that the text is not finished • `hidden`: Break off text, paying attention to whole words • `wordBreak`: Break off within the word • `allow`: Break off text between words
`visible`	Defines if the layout is visible. • Data binding (boolean) possible. **Mögliche Werte:** Boolean or data binding expression

For further information please visit the section

- <ResponsiveLayout>

11 Controls

In this section, you learn what controls are and which attributes can be used for all controls. Controls can be used to display information in the web application and process user input.

11.1 Button

`<Button>` controls are used to create buttons. The button can be freely placed in the Web App. If you want to always display an action button at the top in front of the component content, create a button using Actions.

The `<Button>` element may contain the following actions:

- Action „New“ `<NewAction/>`
- Action „Save“ `<SaveAction/>`
- Action „Delete“ `<DeleteAction/>`
- Action „File upload“ `<UploadAction/>`
- Action „Download file“ `<DownloadAction/>`
- Custom Action `<CustomAction/>`
- Action "Reload" `<ReloadAction>`

The following additional attributes can be defined for the `<Button>` element:

Attribute	Description
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true`/`false` or data binding expression
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • `left` (default) • `center` • `right`

Attribute	Description
`textOverflow`	Defines what happens if the page is full. **Possible values:** • `ellipsis`: Show with ... that the text is not finished • `hidden`: Break off text, paying attention to whole words • `wordBreak`: Break off within the word • `allow` (default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *true* / *false* or data binding expression

11.1.1 Example

```
<DetailComponent default="true" displayName="Dashboard" path="Dashboard">
    <FlowLayout>
        <Header value="Welcome to my new Web App!"/>
        <TextBlock>Upload your file here</TextBlock>
        <Button horizontalAlign="center">
            <UploadAction displayName="Upload" iconUrl="icon:file_upload"/
>
        </Button>
    </FlowLayout>
</DetailComponent>
```

11.2 Charts

Charts ("Chart Controls") can be included in an X4 Web App.

Different types of charts can be included:

- Line chart
- Bar chart
- Pie/donut chart
- Gauge

All charts in a layout must be defined within a `<Chart>` element.

Attribute	Description
`title`	Chart title **Possible values:** Any string or Data Binding

Attribute	Description
`saveAsImage`	Enables the dowload of the charts as image files in SVG format. **Possible values:** • Any string: Defining the file name • TIMESTAMP: File name consists of a string "Chart_" and a timestamp
`minHeight`	Minimum height of the chart **Possible values:** Integer ⓘ If the `minHeight`Charts attribute is not explicitly set, charts with a height of 400 pixels are displayed.
`textOverflow`	Specifies what happens when the control is full. **Possible values:** • *`ellipsis`* : With `...` show that the text is not finished • *`hidden`* : Break text, care for whole words • *`wordBreak`* : Break within the word • *`allow`*(default): Break text between words

ⓘ The charts are optimized for size specifications in pixels. If an explicit definition of height and width of the diagram is needed, the `units` attribute should be set to *`pixels`* beforehand.

⚠ All charts must be filled via data binding!

11.2.1 Line Chart

A line chart can be used to show trends or changes over time. Data points are connected by straight line segments. A line chart can also display several series of numbers.

A line chart can contain the following elements:

- `<Line>`: Contains all elements that belong to the line chart.
- `<Axis>`: Defines the axis labels.
- `<LineData>`: Defines how the data is displayed. For each data set, i.e. each line in the line chart, a `<LineData>` element must be defined.
- `<Tooltip>`: Defines how the tooltip for the single values within the chart is displayed.

11.2.1.1 General Attributes

The following attributes are provided for all elements of a line chart:

`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

11.2.1.2 Line

`<Line>`: Contains all elements that belong to the line chart.

In addition to the general attributes, `<Line>` can have the following attributes:

Attribute	Description
`legend`	Defines if a legend is displayed. **Possible values:** `true` (default) / `false`

11.2.1.3 Axis

`<Axis>`: Child element of `<Line>`. Defines the axis labels.

In addition to the general attributes, `<Axis>` can have the following attributes:

Attribute	Description
`color`	Defines the axis color **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalLabel`	Defines the label of the x-axis. **Possible values:** Any string
`horizontalMax`	Maximum value of the x-axis **Possible values:** Integer
`horizontalMin`	Minimum value of the x-axis **Possible values:** Integer
`horizontalUnits`	Unit of the x-axis **Possible values:** Integer
`verticalLabel`	Defines the label of the y-axis. **Possible values:** Any string
`verticalMax`	Maximum value of the y-axis **Possible values:** Integer

Attribute	Description
`verticalMin`	Minimum value of the y-axis **Possible values:** Integer
`verticalUnits`	Unit of the y-axis **Possible values:** Integer

11.2.1.4 LineData

`<LineData>`: Child element of `<Line>`. Defines how the data is displayed. For each data set, i.e. each line in the line chart, a `<LineData>` element must be defined.

In addition to the general attributes, `<LineData>` can have the following attributes:

Attribute	Description
`color`	Defines the line color. **Possible values:** Hexadecimal color value or data binding expression (Color) ⓘ If the attribute `color` is not specified, the colors defined within the Theming Editor will be used.
`data`	Defines which data is displayed in the diagram. **Possible values:** Data Binding
`labels`	Defines if the data points are labeled with the respective value. **Possible values:** `true` / `false` (default)
`marker`	Defines how a data point is displayed. **Possible values:** • `circle`: Circle • `rect`: Rectangle • `roundRect`: Rectangle with rounded corners • `triangle`: Triangle • `diamond`: Diamond • `pin`: Pin • `arrow`: Arrow • `none`: No marker

Attribute	Description
`markersEnabled`	Defines if a data point is marked. **Possible values:** `true` / `false`
`name`	Name of the data set that is displayed in the legend. **Possible values:** Any string
`xValues`	Defines which data is displayed in the diagram on the x-axis. The specified string refers to a defined property within the data that is specified for `data`. **Possible values:** Data Binding
`yValues`	Defines which data is displayed in the diagram on the y-axis. The specified string refers to a defined property within the data that is specified for `data`. **Possible values:** Data Binding

11.2.1.5 Tooltip

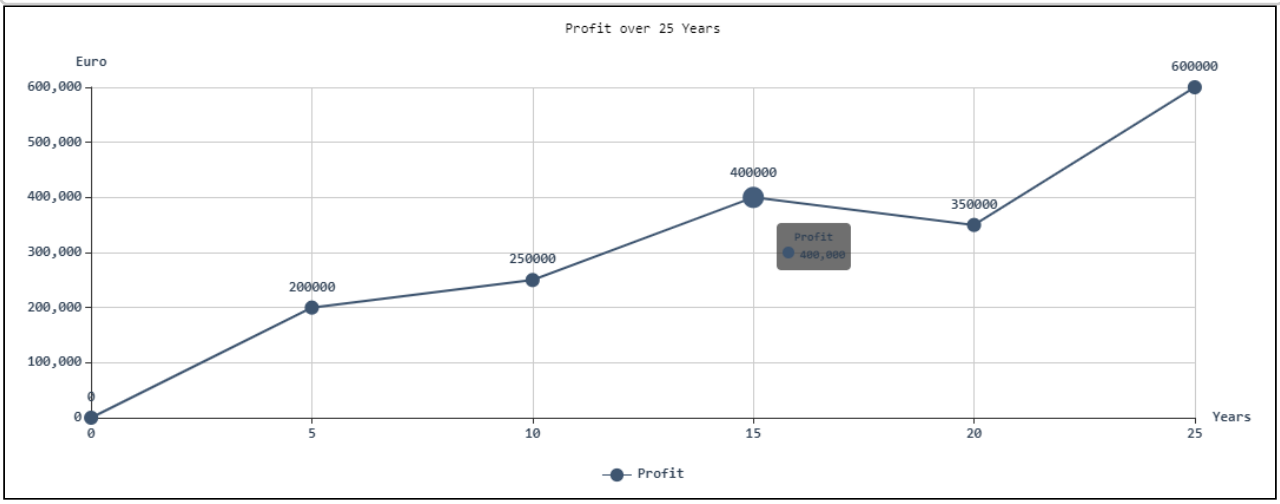
`<Tooltip>`: Child element of `<Line>`. Defines how the tooltip for the single values within the chart is displayed.

For `<Tooltip>` the general attributes (see above) can be defined.

11.2.1.6 Examples

Example 1:

```
<Properties>
    <Property name="DataSource" type="Complex">
        <Property name="Profit" type="List">
            <Property name="Years" type="Integer"/>
            <Property name="Money" type="Integer"/>
        </Property>
    </Property>
</Properties>
<FlowLayout>
    <Chart  fontFamily="Font01" title="Profit over 25 Years">
        <Line fontSize="" fontStyle="normal" fontFamily="Font02" fontWeight="Black"
foreground="A700" legend="true">
            <Tooltip fontFamily="Font01" fontSize="30"/>
            <Axis fontFamily="Font03" horizontalLabel="Years" verticalLabel="Euro"/>
            <LineData data="#DataSource.Profit" labels="true" name="Profit" xValues="
#Years" yValues="#Money"/>
        </Line>
    </Chart>
</FlowLayout>
```

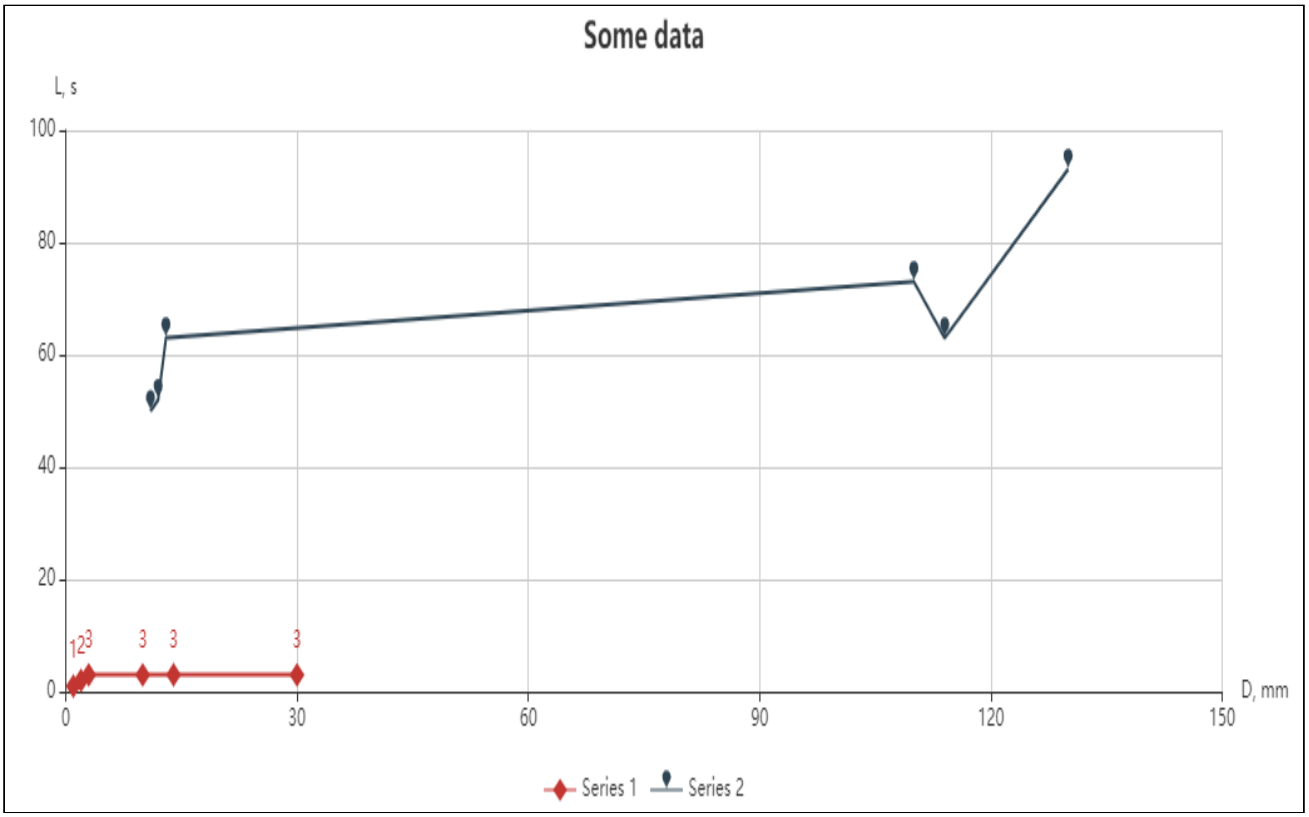


Example 2

```
<Property name="Datasource" type="Complex">
    <Property name="LineDataSet1" type="List">
        <Property name="xValue" type="Integer"/>
        <Property name="yValue" type="String"/>
    </Property>
    <Property name="LineDataSet2" type="List">
        <Property name="xValue" type="Integer"/>
        <Property name="yValue" type="String"/>
    </Property>
</Property>
    ...
<Chart title="Some data">
    <Line>
        <Axis horizontalLabel="D, mm" verticalLabel="L, s" />
        <LineData data="#Datasource.LineDataSet1" xValues="#xValue"
            yValues="#yValue" name="Series 1" marker="diamond" labels="true" />
        <LineData data="#Datasource.LineDataSet2" xValues="#xValue"
            yValues="#yValue" name="Series 2" marker="pin" />
    </Line>
</Chart>
```

The element `<LineData>` is used to display a line in the diagram. The displayed data itself is contained in the property `Datasource`. In the above example, the data is retrieved using the data binding expression

`#Datasource.LineDataSet1`. The above example leads to the following diagram:



11.2.2 Bar Chart

A bar chart is used to compare several variables with a single value. They are helpful to compare categories.

A bar chart can contain the following elements:

- `<Bar>`: Contains all element that belong to the bar chart. Defines how the diagram is displayed.
- `<BarData>`: Defines how the data is displayed. For each category in the bar chart a `<BarData>`-Element must be defined.
- `<Axis>`: Defines the axis labels.

11.2.2.1 General Attributes

The following attributes are provided for all elements of a bar chart:

`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

11.2.2.2 Bar

`<Bar>`: Contains all element that belong to the bar chart. Defines how the diagram is displayed.

In addition to the general attributes, `<Bar>` can have the following attributes:

Attribute	Description
`legend`	Defines if a legend is to displayed. **Possible values:** `true` (default) / `false`
`stacked`	Defines if the bar chart is displayed as stacked diagram. **Possible values:** `true` / `false` (default)

11.2.2.3 BarData

`<BarData>`: Child element of `<Bar>`. Defines how the data is displayed. For each category in the bar chart a `<BarData>`-Element must be defined.

In addition to the general attributes, `<BarData>` can have the following attributes:

Attribute	Description
`category`	Defines which category (`category`) the value is assigned to. **Possible values:** Data Binding
`color`	Bar color **Possible values**: Hexadecimal color value or data binding expression (Color) ⓘ If the attribute `color` is not specified, the colors defined within the Theming Editor will be used.
`data`	Defines which data object of the data source is visualized. **Possible values:** Data Binding
`name`	Defines the name of the data that is displayed in the legend. **Possible values:** Any string or Data Binding
`value`	Defines which value of the data object is visualized. **Possible values:** Data Binding

11.2.2.4 Axis

`<Axis>`: Child element of `<Bar>`. Defines the axis labels.

In addition to the general attributes, `<Axis>` can have the following attributes:

Attribute	Description
`color`	Defines the axis color **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

11.2.2.5 Tooltip

`<Tooltip>`: Child element of `<Bar>`. Defines how the tooltip for the single values within the chart is displayed.

For `<Tooltip>` the general attributes (see above) can be defined.

11.2.2.6 Empty data

If you reload additional data using a process, the already loaded chart is not removed. If you want to overwrite the already loaded chart with the new data, the attribute `empty="true"` must be declared in the data. The attribute empties the existing data.

Input 1

Beispiel 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok>
    <b2 empty="true"/>
    <b1>
        <a>1</a>
        <b>1</b>
    </b1>
</Ok>
```

Input 2

Beispiel 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok>
    <b1 empty="true"/>
    <b2>
        <a>1</a>
        <b>1</b>
    </b2>
</Ok>
```

Chart in the Component

Beispiel 1:

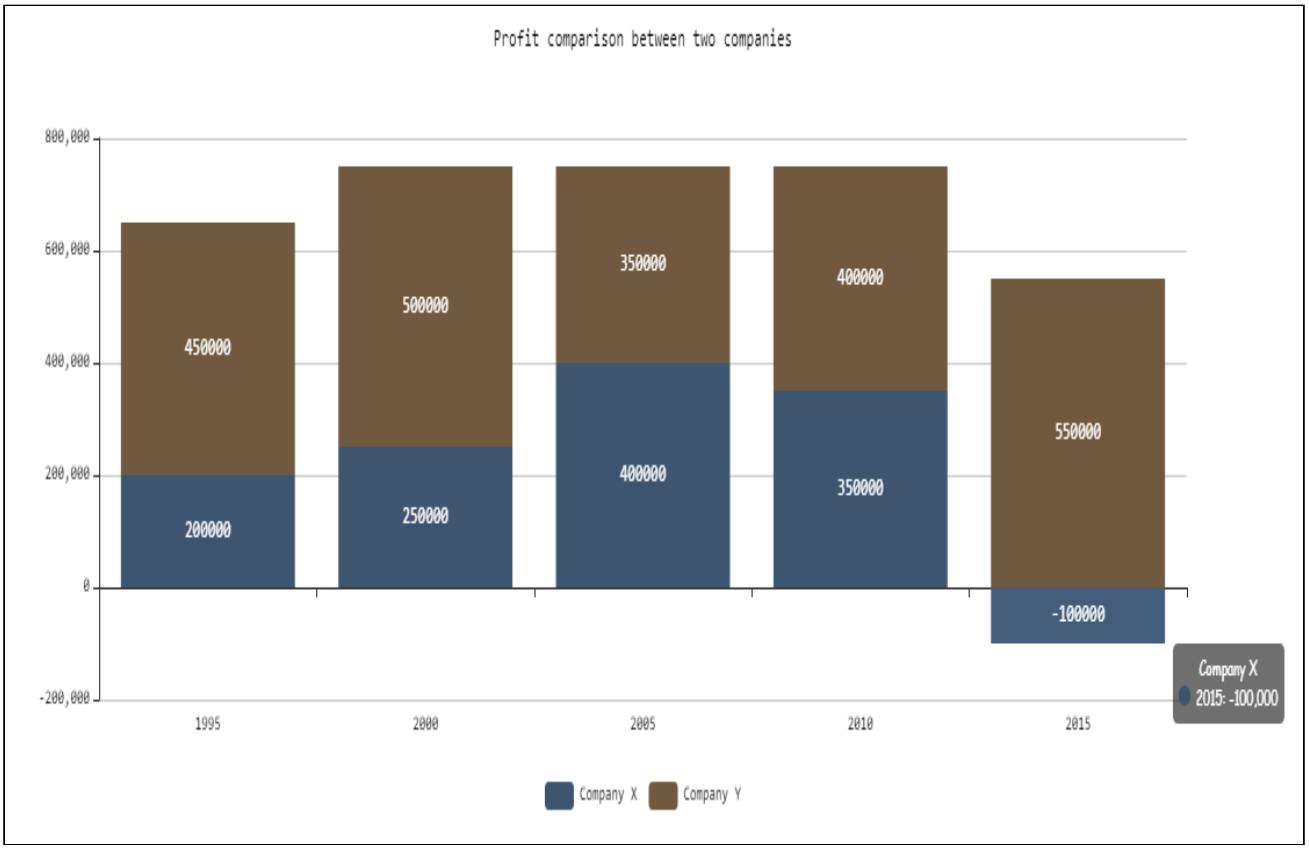
```
<Chart>
    <Bar>
        <BarData
            category="#a"
            data="#b1"
            value="#b" />
        <BarData
            category="#a"
            data="#b2"
            value="#b" />
    </Bar>
</Chart>
```

11.2.2.7 Examples

Example 1:

```
<Properties>
    <Property name="DataSource" type="Complex">
        <Property name="CompanyX" type="List">
            <Property name="Category" type="String"/>
            <Property name="Money" type="Integer"/>
        </Property>
        <Property name="CompanyY" type="List">
            <Property name="Category" type="String"/>
            <Property name="Money" type="Integer"/>
        </Property>
    </Property>
</Properties>
<FlowLayout>
    <Chart fontFamily="Font03" title="Profit comparison between two companies">
        <Bar stacked="true">
            <Tooltip fontFamily="Font02" fontSize="10"/>
            <Axis fontFamily="Font03" fontSize="10"/>
            <BarData category="#Category" data="#DataSource.CompanyX" fontFamily="Fon
t03" name="Company X" value="#Money"/>
            <BarData  category="#Category" data="#DataSource.CompanyY" fontFamily="Fo
nt03" name="Company Y" value="#Money"/>
        </Bar>
    </Chart>
</FlowLayout>
```

The above example leads to the following diagram:


Profit comparison between two companies
800,000
600,000
400,000
200,000
0
-200,000
450000
200000
500000
250000
350000
400000
400000
350000
550000
-100000
1995
2000
2005
2010
2015
Company X
2015: -100,000
Company X
Company Y

Example 2:

```
<Property name="Datasource" type="Complex">
    <Property name="Bar1" type="List">
        <Property name="category" type="String"></Property>
        <Property name="value" type="Integer"></Property>
    </Property>
    <Property name="Bar2" type="List">
        <Property name="category" type="String"></Property>
        <Property name="value" type="Integer"></Property>
    </Property>
    <Property name="Bar3" type="List">
        <Property name="category" type="String"></Property>
        <Property name="value" type="Integer"></Property>
    </Property>

    ...
    ...
    ...

<Chart title="Bar">
    <Bar>
        <BarData data="#Datasource.Bar1" category="#category" value="#value"
            name="Data 1" />
        <BarData data="#Datasource.Bar2" category="#category" value="#value"
            name="Data 2" />
        <BarData data="#Datasource.Bar3" category="#category" value="#value"
            name="Data 3" />
    </Bar>
</Chart>

<Chart title="Stacked Bar">
    <Bar stacked="true">
        <BarData data="#Datasource.Bar1" category="#category" value="#value"
            name="Data 1" />
        <BarData data="#Datasource.Bar2" category="#category" value="#value"
            name="Data 2" />
        <BarData data="#Datasource.Bar3" category="#category" value="#value"
            name="Data 3" />
    </Bar>
</Chart>
```

The element `BarData` is used to display a category in the diagram. The displayed data itself is contained in the property `Datasource`. The above example leads to the following diagrams:

11.2.3 Pie/Donut Chart

A pie chart allows to illustrate the share of individual data in the total.

A pie chart can contain the following elements:

- `<Pie>`: Defines how the diagram is to be displayed.
- `<Tooltip>`: Child element of `<Pie>`. Defines how the tooltip for the single values within the chart is displayed.

11.2.3.1 General Attributes

The following attributes are provided for all elements of a pie chart:

`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

11.2.3.2 Pie

`<Pie>`: Defines how the diagram is to be displayed.

In addition to the general attributes, `<Pie>` can have the following attributes:

Attribute	Description
`color`	Color of the single chart portion **Possible values**: Hexadecimal color value or data binding expression (Color) ⓘ If the attribute `color` is not specified, the colors defined within the Theming Editor will be used.
`data`	Defines which data is displayed in the diagram. The specified string refers to a defined property within the data source specified in the `<Chart>` element. **Possible values:** Data Binding
`donut`	Defines whether the diagram is to be displayed as a donut diagram. **Possible values:** `true` / `false` (default)
`legend`	Defines whether a legend is displayed. **Possible values:** `true` (default) / `false`
`name`	Defines the name assigned to the respective value. The specified string refers to a defined property within the data for `data`. **Possible values:** Data Binding
`normals`	Defines whether the values are to be labeled directly in the diagram. **Possible values:** `true` / `false` (default)
`value`	Defines which value of the data object is visualized. The specified string refers to a defined property within the data for `data`. **Possible values:** Data Binding

Attribute	Description
`labelColor`	Sets the font color of the label. ⚠ Overwrites the color of all labels of the chart. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash before the color value or a shortened notation of the color value. • Color code from the color palette of the web app (see Theming), e.g. *`A200`*
`labelWidth`	Sets the width of the caption in pixels. **Possible values:** • Any integer
`labelOverflow`	Specifies what should happen if the caption is longer than the specified caption width. **Possible values:** • `none`: Prevents text break (default) • `truncate`: With `...` show that the text is not finished • `break`: Break text between words • `breakAll`: Break within the word
`avoidLabelOverlap`	Specifies whether to prevent labels from overlapping. **Possible values:** • `True`: Overlapping of labels is prevented. • `False`: Overlapping of labels is not prevented.
`radius`	Sets the radius in pixels. **Possible values:** • Any integer
`radiusOuterDonut`	Sets the inner radius in pixels. **Possible values:** • Any integer
`centerHorizontal`	Sets the horizontal position of the chart. **Possible values:** • Any integer

Attribute	Description
centerVertical	Sets the vertical position of the chart. **Possible values:** • Any integer
tooltipPercentage	Determines whether a percentage is displayed in the caption and tooltip in addition to the value. **Possible values:** • True: Percentage is displayed. • False: Percentage is not displayed.

11.2.3.3 Tooltip

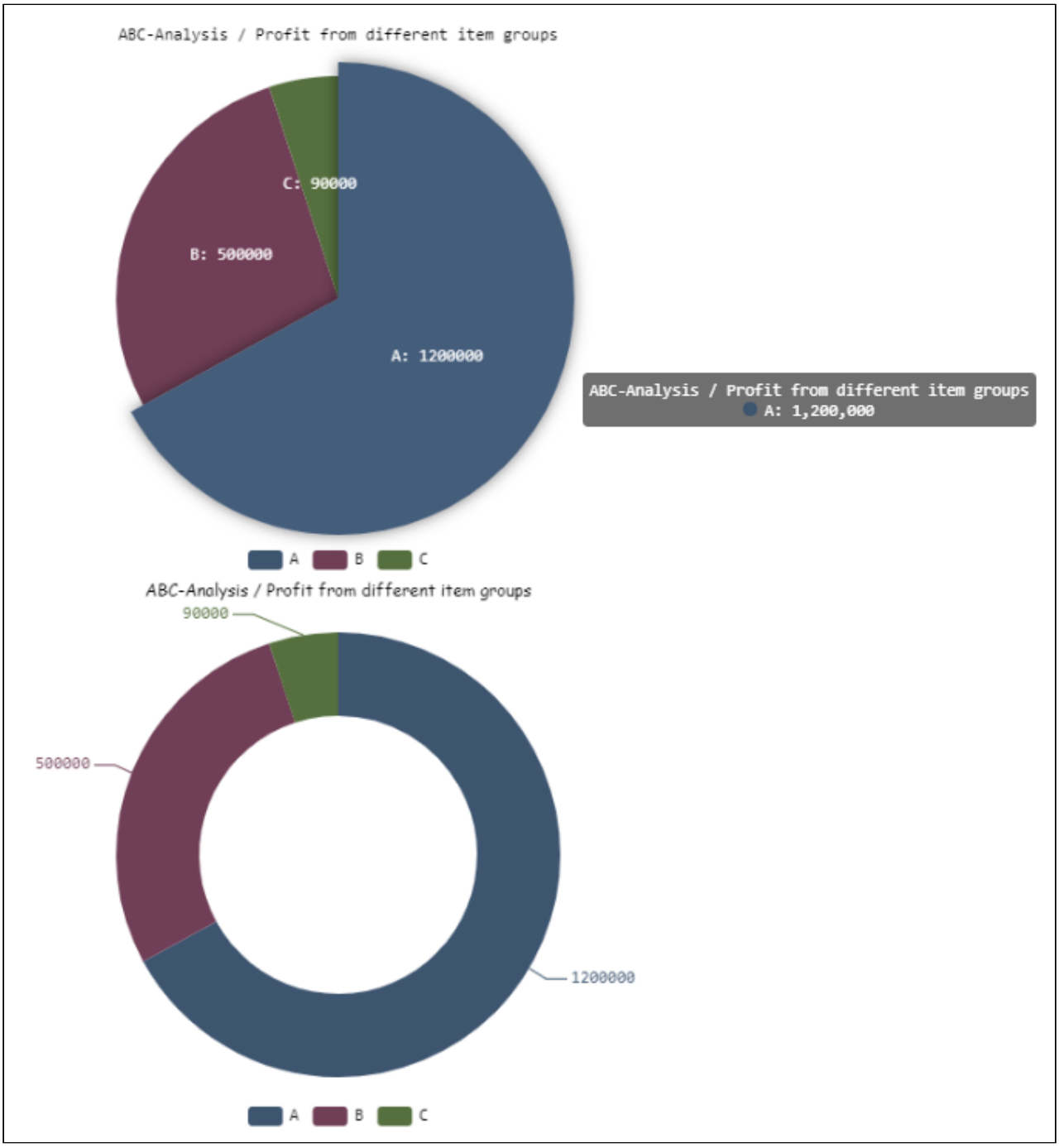
<Tooltip>: Child element of <Pie>. Defines how the tooltip for the single values within the chart is displayed.

For <Tooltip> the general attributes (see above) can be defined.

11.2.3.4 Example

```
<Properties>
    <Property name="DataSource" type="Complex">
        <Property name="ABC-Analysis" type="List">
            <Property name="ItemGroup" type="String"/>
            <Property name="Profit" type="Integer"/>
        </Property>
    </Property>
</Properties>
<FlowLayout>
    <Chart fontFamily="Font03" title="ABC-Analysis / Profit from different item
groups">
        <Pie data="#DataSource.ABC-Analysis" fontFamily="Font03" name="#ItemGroup"
normals="false" value="#Profit">
           <Tooltip fontFamily="Font03"/>
        </Pie>
    </Chart>
    <Chart fontFamily="Font02" title="ABC-Analysis / Profit from different item
groups">
        <Pie data="#DataSource.ABC-Analysis" donut="true" fontFamily="Font03" name="#
ItemGroup"  normals="true" value="#Profit">
            <Tooltip fontFamily="Font03"/>
        </Pie>
    </Chart>
</FlowLayout>
```

The above example leads to the following diagrams:



11.2.4 Gauge Chart

A gauge chart can be used to show states with different valuations. It is well suited to compare target/actual values, e.g. for key figures, customer satisfaction or quality measurements.

A gauge chart can contain the following elements:

- `<Gauge>`: Defines how the diagram is displayed.
- `<Tooltip>`: Defines how the tooltip for the single values within the chart is displayed.

11.2.4.1 General Attributes

The following attributes are provided for all elements of a gauge chart:

`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

11.2.4.2 Gauge

`<Gauge>`: Defines how the diagram is displayed.

In addition to the general attributes, `<Gauge>` can have the following attributes:

Attribute	Description
`axisMax`	Defines the end range of the gauge chart in percent. The color of the area is defined within the Theming Editor via the color property `GaugeAxisMax`. **Possible values:** Integer between `0` and `100` (default: `100`) **Example** The default values (`axisMin="20"`, `axisMiddle="80"`, `axisMax="100"`) define a gauge chart in which the range from 0 to 20 %, from 20 to 80 % and from 80 to 100 % is displayed in different colors.
`axisMiddle`	Defines the middle range of the gauge chart in percent. The color of the area is defined within the Theming Editor via the color property `GaugeAxisMiddle`. **Possible values:** Integer between `0` and `100` (default: `80`)
`axisMin`	Defines the initial range of the gauge chart in percent. The color of the area is defined within the Theming Editor via the color property `GaugeAxisMin`. **Possible values:** Integer between `0` and `100` (default: `20`)
`endAngle`	Defines the endpoint of the gauge chart. **Possible values:** Integer between `-360` and `360` (default: `45`) `endAngle` must always be smaller than `startAngle`.
`legend`	Defines if a legend is displayed. **Possible values:** `true` (default) / `false`
`max`	Defines the maximum value of the gauge chart. **Possible values**: Integer (default: `100`)
`min`	Defines the minimum value of the gauge chart. **Possible values:** Integer (default: `0`) The connection from `min` to `max` is always **clockwise**.

Attribute	Description
`startAngle`	Defines the start point of the gauge chart. **Possible values:** Integer between `-360` and `360` (default: `255`) For this setting, the following integers correspond to the following positions on a clock face: • `0`: 3 o'clock • `90`: 12 o'clock • `180`: 9 o'clock • `270`: 6 o'clock
`value`	Defines which value of the data object is visualized. **Possible values:** Any string or Data Binding

11.2.4.3 Tooltip

`<Tooltip>`: Child element of `<Gauge>`. Defines how the tooltip for the single values within the chart is displayed.

For `<Tooltip>` the general attributes (see above) can be defined.

11.2.4.4 Examples

Example 1:

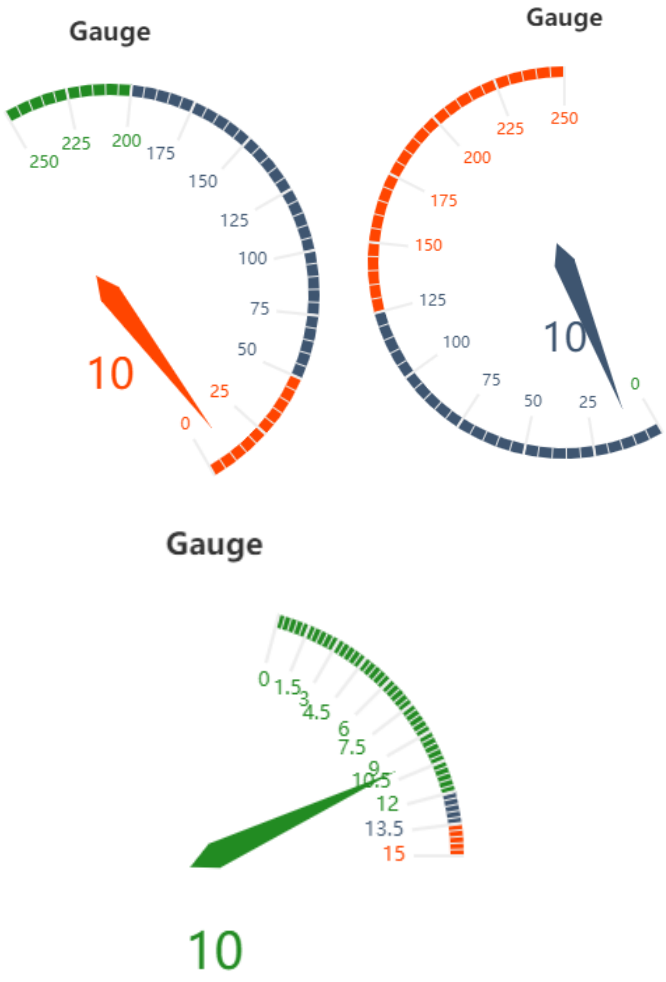
```
<Properties>
    <Property name="DataSource" type="Complex">
        <Property name="Satisfaction" type="Integer"/>
    </Property>
</Properties>
<FlowLayout>
    <Chart fontFamily="Font03" title="Employee Satisfaction">
        <Gauge  max="0" min="100" value="#DataSource.Satisfaction">
            <Tooltip fontFamily="Font02" fontSize="12"/>
        </Gauge>
    </Chart>
</FlowLayout>
```

The above example creates the following diagram:

Example 2:

```
<Property name="Datasource" type="Complex">
    <Property name="gauge" type="Integer"/>
    ...
    ...
    ...
</Property>
<Chart title="Gauge">
    <Gauge value="#Datasource.gauge" min="0" max="250" startAngle="-60" endAngle="-27
0" axisMin="0" axisMiddle="50" />
</Chart>
<Chart title="Gauge">
    <Gauge value="#Datasource.gauge" min="0" max="15" startAngle="75" axisMin="80"
axisMiddle="90" />
</Chart>
<Chart title="Gauge">
    <Gauge value="#Datasource.gauge" min="-1" max="250" startAngle="120" />
</Chart>
```

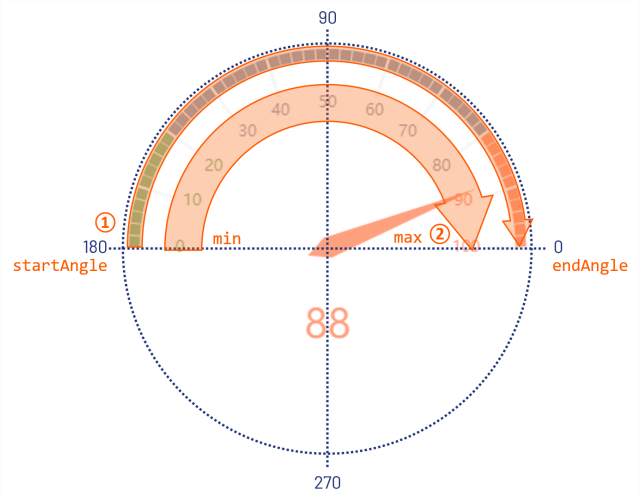
The above example creates the following diagrams:


Gauge
250
225
200
175
150
125
100
75
50
25
0
10
Gauge
250
225
200
175
150
125
100
75
50
25
0
10
Gauge
0
1.5
3
4.5
6
7.5
9
10.5
12
13.5
15
10

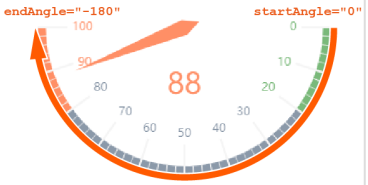
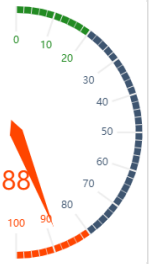
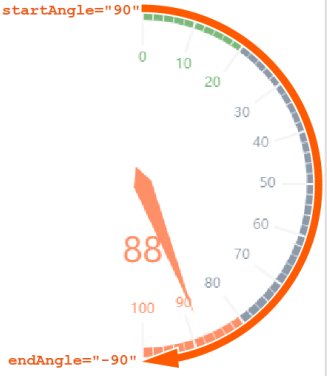
11.2.4.5 Functionality and further examples for gauge charts

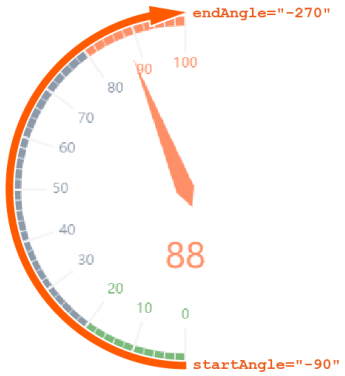
The following must be observed for gauge charts:

1. `endAngle` must be smaller than `startAngle`.
2. The connection from the start angle to the end angle is always **clockwise**.
3. The connection from `min` to `max` (scale) is always **clockwise**.



The following are further examples of gauge charts with explanations to illustrate how the start and end angles work.

Definition	Result	Explanation
`<Gauge startAngle="0" endAngle="-180" min="0" max="100" ...>`		
`<Gauge startAngle="90" endAngle="-90" min="0" max="100" ...>`		

Definition	Result	Explanation
`<Gauge startAngle="180" endAngle="0" min="0" max="100" ...>`		
`<Gauge startAngle="-90" endAngle="-270" min="0" max="100" ...>`		
`<Gauge startAngle="240" endAngle="-60" min="0" max="100" ...>`		

11.3 Checkbox

<Checkbox> controls are used to work with Boolean values, e.g. to get user input in "Yes/No" format.

Attribute	Description
`checked`	**mandatory** Indicates whether the checkbox is checked or unchecked by default. • Data binding (boolean) possible ⓘ The usage of Data Binding for the state of the checkbox is necessary for the checkbox to work properly and to store the initial value of `result` or `set`. **Possible values:** *`true`* / *`false`* or data binding expression
`displayName`	Label. Displayed next to the checkbox • Data binding possible **Possible values:** Any string
`sliderOnDesktop`	Defines whether the checkbox is displayed as a slider in the desktop version. **Possible values:** *`true`* / *`false`*
`background`	Defines a color for the background of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression ⓘ The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .
`fontFamily`	Defines the font family. ⓘ • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. ⓘ • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines a color for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

Checkboxes are displayed differently in the desktop and mobile version.

- Display in the desktop version:
- Display in the mobile version:

with the attribute `sliderOnDesktop` the display for the mobile version can also be displayed in the desktop version.

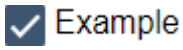
11.3.1 "Select" Action within a Checkbox

The **Select** action is created via the element `<SelectAction/>` within `<Checkbox>` and is triggered as soon as the checkbox has been checked or unchecked.

11.3.2 Example for <Checkbox>

```
<DetailComponent name="Example" path="Example" displayName="Example">
    <FlowLayout>
        <Checkbox checked="true" displayName="Example"/>
    </FlowLayout>
</DetailComponent>
```

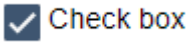
Above code creates the following Checkbox:



11.3.3 Example for <Checkbox> with a Technical Process

```
<DetailComponent path="Dashboard"
	displayName="Dashboard" default="true"
	process="/Processes/process_1.wrf">
	<Properties>
		<Property name="boolean" type="Boolean"></Property>
	</Properties>
	<FlowLayout>
		<Checkbox checked="#boolean" displayName="Check box" />
	</FlowLayout>
</DetailComponent>
```

Above example creates the following Checkbox:



11.4 ComboBox

<ComboBox> controls are used to create selection lists.

To trigger the Action "Select" when a ComboBox item is selected, a SelectAction action can be inserted within the ComboBox control.

In addition to the standard attributes for controls, the following attributes are available for the element <Video>:

Attribute	Description
displayName	Title of the ComboBox. Displayed small above the input/output field. • Translatable • Data binding possible **Possible values:** Any string
displayProperty	Has to be set if optionsList is set. Text that is displayed as option in the ComboBox. **Possible values:** String that corresponds to a defined property. ⚠ The specified string must correspond to a valid subproperty used in optionsList. If no displayProperty is specified, the first subproperty of the property specified for optionsList is automatically used.

Attribute	Description
`iconColorProperty`	Name of the property that contains the color information of the icon. ⓘ This attribute is only used for material icons. The color specification can be a hexadecimal color value, e.g. *`ff5a00`* or a color code from the color palette of the web app (see Theming), e.g. *`A200`*. • Data binding possible **Possible values:** String that corresponds to a defined property. ⓘ For relative bindings a hashtag # must be used as prefix for the property, e.g. *`#Country.Id`*.
`iconProperty`	Name of the property containing the URL of the icon resource. ⓘ The icon can be either a resource from the `Resources` folder, a Material icon or an icon embedded via an external URL. If the icon is included via an external URL, an additional attribute `type="external"` must be specified in the property. • Data binding possible **Possible values:** String that corresponds to a defined property. ⓘ For relative bindings a hashtag # must be used as prefix for the property, e.g. *`#Country.Id`*.
`optionsAutoWidth`	Enable automatic width for the option list **Possible values:** • `true` (default): The width of the option list automatically adjusts to the width of the option values • `false`: The width of the option list depends on the width of the ComboBox
`optionsList`	*If the option list is outsourced*: Source of the list. Can only be used together with `displayProperty`. • Controlled by data binding (list). **Possible values:** String (data binding)

Attribute	Description
`optionsMaxWidth`	Maximum width of the option list in percent or pixels, where percent refers to the screen width **Possible values:** Integer
`optionsUnits`	Defines the unit for the attribute `optionsMaxWidth` **Possible values:** • `percents` • `pixels` (default)
`value`	Value that is displayed and saved. • Data binding possible **Possible values:** Any string
`valueProperty`	Value that is processed technically. If this attribute is set, the property that is defined for this value must have the same name. If this attribute is not set, `value` is used. **Possible values:** Any string ⚠ The specified string must correspond to a valid subproperty used in `optionsList`. If no `valueProperty` is specified, the value specified for `displayProperty` is automatically used.
`background`	Defines a color for the background of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression ⓘ The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .
`fontFamily`	Defines the font family. ⓘ • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. ⓘ • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines a color for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

`ComboBox` contains at least one `<Option>` element that creates an option. The options of the pick list can either come from a Technical Process or be entered manually.

The `<Option>` element may have the following attributes:

Attribute	Description
`iconColor`	Color of the icon ⓘ This attribute is only used for Material Icons. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*

Attribute	Description
`iconUrl`	Path to the used icon ⓘ The icon can be either a resource from the `Resources` folder, a Material icon or an icon embedded via an external URL. **Possible values:** • String (URI), e.g. *`clock.png`* • Specification of a material icon according to the pattern `icon:<MaterialIconName>` , , e.g. *`icon:extension`* ✓ With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons. • External URL

The data for the outsourced options list must be available as XML data. The options must be contained in a list:

```
<example>
    <listElement>list element 1</listElement>
</example>
<example>
    <listElement>list element 2</listElement>
</example>
```

The property definition is carried out according to the XML data. Please not that an additional property `saveValue` is created, which is used as initial value and container for saving the user input.

```
<Property name="example" type="List">
    <Property name="listElement" type="String" />
</Property>
<Property name="saveValue" type="String" />
```

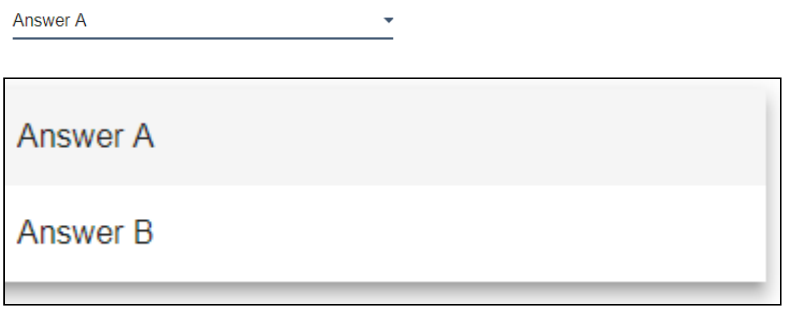
11.4.1 Examples

11.4.1.1 Example <ComboBox>

The following example shows the usage of the `<ComboBox>` tag.

```
<DetailComponent name="ExampleComboBox" path="ExampleComboBox" displayName="Example
ComboBox" process="ComboBox.wrf">
    <FlowLayout>
        <ComboBox value="#saveValue">
            <Option displayName="Answer A"/>
            <Option displayName="Answer B"/>
        </ComboBox>
    </FlowLayout>
</DetailComponent>
```

Above code creates the following ComboBox:



11.4.1.2 Example <ComboBox> with outsourced pick list

The following example shows the usage of the <ComboBox> element with an outsourced pick list.

***.wad file ComboBox**

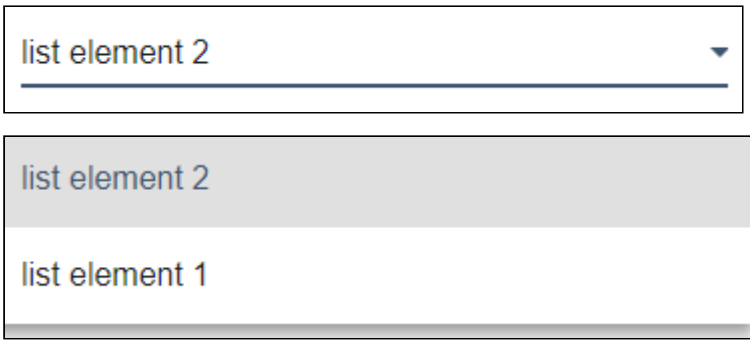
```
<DetailComponent name="ExampleComboBox" path="ExampleComboBox" displayName="Example
ComboBox" process="ComboBox.wrf">
    <Properties>
        <Property name="example" type="List">
            <Property name="listElement" type="String" />
        </Property>
        <Property name="saveValue" type="String" />
    </Properties>
    <FlowLayout>
        <ComboBox value="#saveValue" optionsList="#example" >
        </ComboBox>
    </FlowLayout>
</DetailComponent>
```

***.xml-Datei option list ComboBox**

```
<?xml version="1.0" encoding="UTF-8" ?>
<Ok>
    <example>
        <listElement>list element 1</listElement>
    </example>
    <example>
        <listElement>list element 2</listElement>
    </example>
</Ok>
```

ⓘ The XML data to be used for the options must be provided by a Technical Process (in the example `ComboBox.wrf`).

Above code creates the following ComboBox:



11.4.1.3 Example <ComboBox> with icon

The following example shows the usage of the `<ComboBox>` element with icons and an outsourced pick list.

The following options are provided as selection options via a Technical Process (here `Options.wrf`):

Options list

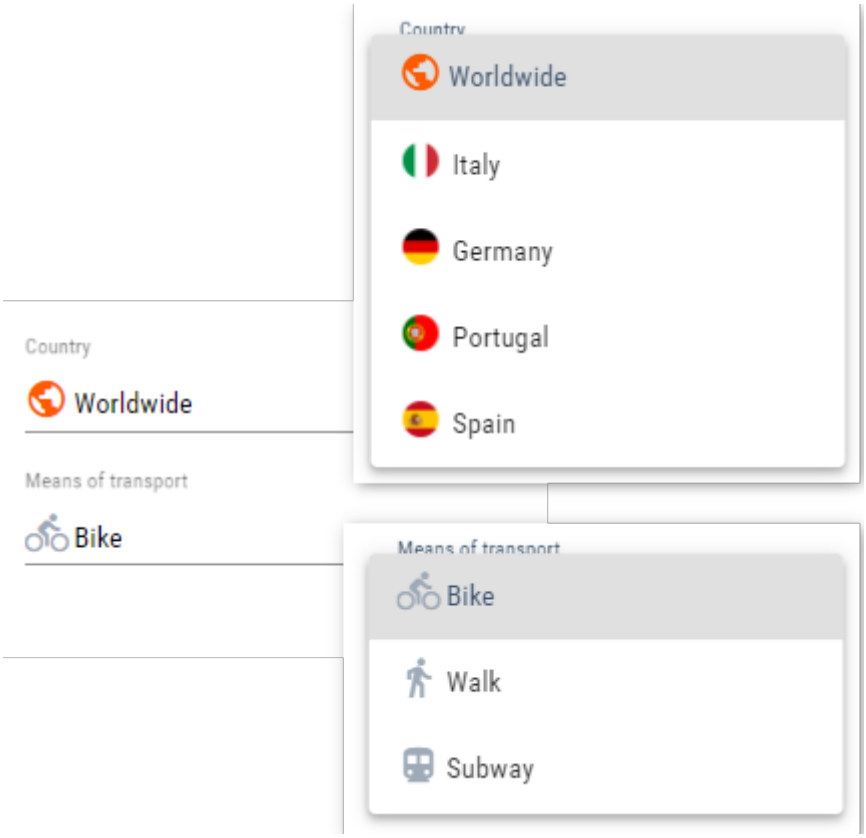
```
<?xml version="1.0" encoding="UTF-8"?>
<OkList>
    <Country>
        <Id>6</Id>
        <Name>Worldwide</Name>
        <icon>icon:public</icon>
        <color>ff5a00</color>
    </Country>
    <Country>
        <Id>2</Id>
        <Name>Italy</Name>
        <icon>italy.png</icon>
    </Country>
    <Country>
        <Id>5</Id>
        <Name>Germany</Name>
        <icon>germany.png</icon>
    </Country>
    <Country>
        <Id>3</Id>
        <Name>Portugal</Name>
        <icon>portugal.png</icon>
    </Country>
    <Country>
        <Id>4</Id>
        <Name>Spain</Name>
        <icon>spain.png</icon>
    </Country>
    <selected>6</selected>
</OkList>
```

The two example ComboBoxes are defined as follows:

Snippet from the definition file

```
...
<DetailComponent default="true" displayName="Dashboard" path="Dashboard"
                 process="folder_1/Options.wrf">
    <Properties>
        <Property name="selected" type="Integer"/>
        <Property name="direction" type="Integer"/>
        <Property name="Country" type="List">
            <Property name="Id" type="Integer"/>
            <Property name="Name" type="String"/>
            <Property name="icon" type="Image"/>
            <Property name="color" type="Color"/>
        </Property>
        <Property name="image" type="Image"/>
    </Properties>
    <FlowLayout>
        <ComboBox displayName="$Country" displayProperty="#Name"
                  iconColorProperty="#color"
                  iconProperty="#icon"
                  optionsList="#Country"
                  value="#selected"
                  valueProperty="#Id"/>
        <ComboBox displayName="Means of transport" value="#direction">
            <Option displayName="Bike" iconColor="A200" iconUrl="icon:directions_bike
" value="1"/>
            <Option displayName="Walk" iconColor="A200" iconUrl="icon:directions_walk
" value="4"/>
            <Option displayName="Subway" iconColor="A200" iconUrl="icon:directions_su
bway" value="3"/>
        </ComboBox>
    </FlowLayout>
</DetailComponent>
...
```

The result is as follows:



11.5 ComboBoxAutocomplete

`<ComboBoxAutocomplete>` controls are used to create pick lists with an auto-complete feature.

To trigger the Action "Select" when a `ComboBox` item is selected, a `SelectAction` action can be inserted within the `ComboBox` control.

The following additional attributes can be defined for the `<ComboBoxAutocomplete>` element:

Attribute	Description
`displayName`	Title of the ComboBox. Displayed small above the input/output field. • Translatable • Data binding possible **Possible values:** Any string
`optionsList`	*If the option list is outsourced*: Source of the list. Can only be used together with `displayProperty`. • Controlled by data binding (list). **Possible values:** String (data binding)

Attribute	Description
`value`	Value that is displayed and saved. • Data binding possible **Possible values:** Any string
`valueProperty`	Value that is processed technically. If this attribute is set, the property that is defined for this value must have the same name. If this attribute is not set, `value` is used. **Possible values:** Any string
`background`	Defines the background colour of the control. ⓘ • This setting overwrites the default colour of the colour scheme! • This attribute does not work for the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` ⚠ Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines whether the user can interact with the control. • Data binding (Boolean) possible **Possible values:** `true`/`false` or expression for data binding ⓘ The attribute `enabled` replaces the obsolete attribute `disabled`. *`enabled="true"`* thus corresponds to the obsolete attribute *`disabled="false"`*.

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the web app for this control. • If the attribute is defined on an element, the font family is inherited by the child elements of the element, unless the child elements have explicitly set the `fontFamily` attribute. • This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`. **Possible values:** • `MainFont:` deposited main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`* ; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in point, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Keywords: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Defines the width of each character. ⓘ This attribute overrides the default width of the characters for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute does not work for the elements `Image`, `Map`, `Chart` and `HtmlDocument`.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default font style for this control. **Possible values:** • `italic`: italic font • `normal`: normal font (default) • `oblique`: oblique font style (calculated) ⓘ This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute does not work for the `Image`, `Map` and `HtmlDocument`.
`foreground`	Defines a colour for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default colour of the colour scheme! • This attribute does not work for the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` ⚠ Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Specifies what should happen when the control is full. **Possible values:** • *`ellipsis`* : With `...` show that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: break within the word • *`allow`*(default): Break text between words
`visible`	Defines whether the control is visible. • Data binding (Boolean) possible **Possible values:** *`true`* / *`false`* or string for data binding

`ComboBox` contains at least one `<Option>` element that creates an option. The options of the pick list can either come from a Technical Process or be entered manually.

The `<Option>` element may have the following attributes:

Attribute	Description
`iconColor`	Color of the icon ⓘ This attribute is only used for Material Icons. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* Do not use a hash in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*

Attribute	Description
`iconUrl`	Path to the used icon ⓘ The icon can be either a resource from the `Resources` folder, a Material icon or an icon embedded via an external URL. **Possible values:** • String (URI), e.g. *`clock.png`* • Specification of a material icon according to the pattern `icon:<MaterialIconName>`, , e.g. *`icon:extension`* ✓ With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons. • External URL

11.5.1 Sample

In this example, an outsourced options list is used. The displayed name and the associated technical value differ.

The data for an outsourced options list have to be available as XML data. The options including the technical values must be contained in a list:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok>
    <ComboBoxValue></ComboBoxValue>
    <OptionList>
        <OptionValue>Red</OptionValue>
        <TechnicalValue>#FF0000</TechnicalValue>
    </OptionList>
    <OptionList>
        <OptionValue>Green</OptionValue>
        <TechnicalValue>#008000</TechnicalValue>
    </OptionList>
    <OptionList>
        <OptionValue>Blue</OptionValue>
        <TechnicalValue>#0000FF</TechnicalValue>
    </OptionList>
</Ok>
```

The properties definition is done according to the XML data.

```
<Properties>
    <Property
        name="ComboBoxValue"
        type="String"></Property>
    <Property
        name="OptionList"
        type="List">
        <Property
            name="OptionValue"
            type="String"></Property>
        <Property
            name="TechnicalValue"
            type="String"></Property>
    </Property>
</Properties>
```

The `ComboBoxAutocomplete` control in the component contains a `SelectAction`. The `SelectAction` sends the data to the defined Technical Process when an option is selected from the `ComboBoxAutocomplete`. In the `ComboBoxAutocomplete` the data from the swapped option list is used. The `OptionValue` elements are displayed, but the technical data from the `TechnicalValue` element are selected in the background.

```
<?xml version="1.0" encoding="UTF-8"?>
<DetailComponent
    xmlns="http://softproject.de/webapp/1.0"
    process="LoadData.wrf">
<Properties>
    <Property
        name="ComboBoxValue"
        type="String"></Property>
    <Property
        name="OptionList"
        type="List">
        <Property
            name="OptionValue"
            type="String"></Property>
        <Property
            name="TechnicalValue"
            type="String"></Property>
    </Property>
</Properties>
    <FlowLayout>
        <Header value="Welcome to my new Web App!" />
        <ComboBoxAutocomplete
            value="#ComboBoxValue"
            optionsList="#OptionList" valueProperty="#TechnicalValue">
            <SelectAction process="SaveData.wrf" />
        </ComboBoxAutocomplete>
    </FlowLayout>
</DetailComponent>
```

11.6 DateTimePicker

`<DateTimePicker>` controls are used to create a date and time picker.

The following additional attributes can be defined for the `<DateTimePicker>` element:

Attribute	Description
`value`	Value that is displayed and saved. • Data Binding possible **Possible values:** Any string
`format`	Format in which the time is displayed and stored. **Possible values:** • `Date Time With Minutes`: Time indicating hours and minutes in the format **hh:mm**. • `Date Time With Seconds`: Time indicating hours, minutes and seconds in the format **hh:mm:ss**. • `Date Time With Milliseconds`: Time indicating hours, minutes, seconds and milliseconds in the format **hh:mm:ss.sss**.
`background`	Defines the background color of the control. • This setting overwrites the default color of the color scheme! • This attribute does not work for the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e. g. `ff5a00` Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (see Theming), e. g. `A200`
`displayName`	Control title. • Translatable • Data binding possible **Possible values:** Any string

Attribute	Description
`enabled`	Defines whether the user can interact with the control. • Data binding (Boolean) possible **Possible values:** `true`/`false` or expression for data binding ⓘ The attribute `enabled` replaces the obsolete attribute `disabled`. *`enabled="true"`* thus corresponds to the obsolete attribute *`disabled="false"`*.
`fontFamily`	Defines the font family. ⓘ • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the child elements of the element, unless the child elements have explicitly set the `fontFamily` attribute. • This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`. **Possible values:** • `MainFont:` deposited main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. ⓘ • This attribute overrides the default font size of the Web App for this control. • This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`* ; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in point, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Keywords: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Defines the width of each character. ⓘ This attribute overrides the default width of the characters for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute does not work for the elements `Image`, `Map`, `Chart` and `HtmlDocument`.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default font style for this control. **Possible values:** • `italic`: italic font • `normal`: normal font (default) • `oblique`: oblique font style (calculated) ⓘ This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute does not work for the `Image`, `Map` and `HtmlDocument`.
`foreground`	Defines a colour for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work for the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e. g. `ff5a00` ⚠ Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (see Theming), e. g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Specifies what should happen when the control is full. **Possible values:** • *`ellipsis`* : With `...` show that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: break within the word • *`allow`*(default): Break text between words
`visible`	Defines whether the control is visible. • Data binding (Boolean) possible **Possible values:** *`true`* / *`false`* or string for data binding

11.7 File link

`FileLink` controls are used to include files for download. There are two possibilities for this download. On the one hand, the file to be downloaded can be located in the `Resources` folder as a static resource, on the other hand, the file can be loaded dynamically via a Technical Process.

Attribute	Description
`data`	⚠ Use only, if a base64 file is provided for download! Defines which data is to be downloaded. **Possible values:** Data binding expression ⓘ The Data Binding expression must refer to a property of type b ase64!
`displayName`	Name of the file that is displayed in the web application and is assigned as the file name for the downloaded file. • Translatable • Data binding possible **Possible values:** Any string incl. file extension

Attribute	Description
`fileName`	Identifier for the Technical Process that provides the data. • Translatable • Data binding possible **Possible values:** Any string
`process`	Technical Process that provides the data. **Possible values:** String (URI)
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`fileId`	Identifier that can be used by the process developer. • Data binding possible. **Possible values:** Any string
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines a color for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.7.1 Example <FileLink> (File download from a static resource)

The following example shows the usage of the <FileLink> tag. The file `1.pdf` which is to be made available for download is located directly in the `Resources` folder and is delivered via the Technical Process `process_1.wrf` in the `Services/Processes` folder. Both a file and a Technical Process must be available for a download.

Example FileLink

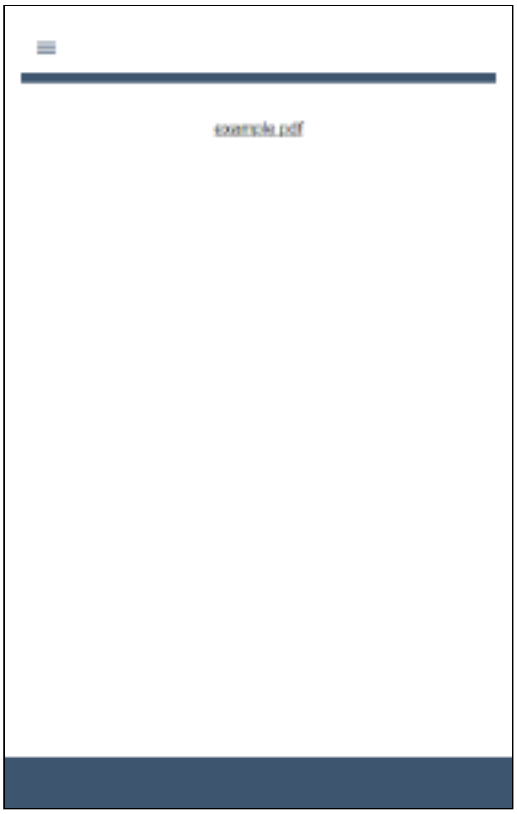
```
<WebApp xmlns="http://softproject.de/webapp/1.0" path="download">
    <Modules>
        <Module path="Module" displayName="My Module">
            <Components>
                <DetailComponent path="Dashboard"
                    displayName="Dashboard" default="true">
                    <FlowLayout>
                        <FileLink process="process_1.wrf" fileName="1.pdf"
displayName="example.pdf" />
                    </FlowLayout>
                </DetailComponent>
            </Components>
        </Module>
    </Modules>
</WebApp>
```

The file link is defined in the Web App definition above. For `FileLink` a Technical Process is stored that supplies the file (`process_1.wrf`). A display name is assigned (`displayName="example.pdf"`) that is used as text in the Web App and as file name for the download. The attribute `fileName` is not relevant in this example, this attribute can be used in more complex applications because it is returned to the processing Technical Process by the control:

Input FileLink

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<fileId>1.pdf</fileId>
```

The code from the example for `FileLink` creates the following view in the web application:



11.8 File Upload

The `<FileUpload>` control can be used to upload files to a Web App created with X4 Web Apps. The `<FileUpload>` control can only be used in Detail Components or Detail Structure Elements within Master/Detail Components.The `<FileUpload>` control uses the standard **Open File** dialog of the browser, where the user can select the file to be uploaded. After the user selects the file, the upload starts. The Technical Process that receives the uploaded file must return a confirmation or an error message. If the confirmation contains a message, this message is displayed as a pop-up window. Pop-up windows with error messages are displayed in any case.

When using a mobile device, the integrated camera can also be accessed.

Attribute	Description

`acceptedFileTypes`	File type to filter by in the selection dialog.
	• Data Binding possible
	This attribute filters only by file types. The filter can be reset by the user to display and select all file types.
	Possible values: MIME types, also with wildcards (`*`), e.g.
	• `text/*`: Text files
	• `image/*`: Image files
	  • `image/jpg`: JPG files only
	  • `image/png`: PNG files only
	• `video/*`: Video files
	• `audio/*`: Audio files
	• `application/*`: Files bound to a specific program
	• `multipart/*`: multipart data
	• `message/*`: Messages
	• `model/*`: Files representing multidimensional structures

`displayName`	Display name in the interface. **Possible values:** Any string
`fileId`	Identifier that can be used by the process developer. • Data Binding possible **Possible values:** Any string
`maxSizeMB`	Maximum file size in Megabytes. **Possible values:** Integer
`process`	*Required*. Technical Process that receives the uploaded file. The Technical Process must be specified including file extension! **Possible values:** String (URI)

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme. • This attribute does not work for the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash before the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines whether the user can interact with the control. • Data Binding (Boolean) possible **Possible values:** `true`/`false` or expression for Data Binding The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false"`* attribute .
`fontFamily`	Defines the font family. • This attribute overrides the default Web App font for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute does not work for the `Image`, `Map` and `HtmlDocument`. **Possible values:** • `MainFont`: defined main font • Font code from the Font Palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute does not work for the `Image`, `Map` and `HtmlDocument` elements. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in point, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Keywords: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Defines the width of each character. This attribute overrides the default width of the characters for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` This attribute does not work for the elements `Image`, `Map`, `Chart` and `HtmlDocument`.

Attribute	Description
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default font style for this control. **Possible values:** • `italic`: italic font • `normal`: normal font (default) • `oblique`: oblique font style (calculated) ⓘ This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`.
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute does not work for the `Image`, `Map` and `HtmlDocument`.

Attribute	Description
`foreground`	Defines a colour for the foreground (texts etc.) of the control. • This setting overwrites the default colour of the colour scheme! • This attribute does not work for the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Specifies what should happen when the control is full. **Possible values:** • *`ellipsis`*: With `...` show that the text is not finished • *`hidden`*: Cancel text, watch out for whole words • *`wordBreak`*: break within the word • *`allow`* (default): Wrap text between words
`visible`	Defines whether the control is visible. • Data Binding (Boolean) possible **Possible values:** *`true`* / *`false`* or string for data binding

11.8.1 Input format for the Technical Process

The data is transmitted to the Technical Process in a specific input format. The format is described below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<File>
    <FileId>....</FileId>
    <FileName>....</FileName>
    <FileType>....</FileType>
    <FileData>
    ....Base64 data....
    </FileData>
</File>
```

The data in the `<FileData>` element can be decoded with the Base64 Converter, for example.

11.8.2 `<FileUpload>` example

The following example shows the use of the `<FileUpload>` element.

```
<FileUpload
    fileId="id"
    process="process_1.wrf"
    maxSizeMB="1"
    acceptedFileTypes="image/*"
/>
```

The above code leads to the following control:

11.9 Header

`<Header>` controls are used to create headers. A header can be placed not only at the top of a layout, but anywhere it is needed.The following table explains the attributes that can be used.

Attribute	Description
`value`	Contains the header text. • Translatable • Data binding possible **Possible values:** Any string
`titleLevel`	Defines the heading level. **Possible values:** • *`title`*(default) • *`subtitle`*

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *20*; *20.8*; *.9* • Font size in pixels, e.g. *20px* • Font size in points, e.g. *18pt* • Font size compared to the font size of the parent element, e.g. *.8em* or *120* • Key words: *xx-Small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*, *smaller*, *larger*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.

Attribute	Description
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`foreground`	Defines a color for the foreground (texts etc.) of the control. • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`* (default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.9.1 Example <Header>

The following example shows the usage of the <Header> tag.

```
<Header value="Header"/>
```

The above code creates the following header:

11.10 HTML Document

`<HtmlDocument>` controls are used to display static HTML (mark up text) e.g. e-mails. All common HTML5 elements that can be displayed by the browser are supported.

> ⚠ The display of HTML is implemented as an iFrame in sandbox mode. Therefore, JavaScript is not executed by default. If an embedded resource (for example, video, image, or audio), contains JavaScript, that resource is not displayed.

> ⚠ The elements `<video>` and `<audio>` are not supported by Safari.
> The elements `<video>`, `<picture>` and `<source>` are not displayed on an iPad.

> ✓ The display can be formatted with a `<style>` tag in the `<head>` element of the HTML document. Local CSS files are not supported.

The following attributes can be defined for the `<HtmlDocument>` element:

Attribute	Description
`process`	*Required*. Technical Process that provides the HTML document. **Possible values:** String (URI) ⚠ The called Technical Process must return a complete HTML document!
`allowSameOrigin`	Allows embedded content to share memory if the origin of the embedded content is the same as the origin of the host web application. **Possible values:** • `true` (default) • `false`
`allowScript`	Allows the content to use Javascript. **Possible values:** • `true` • `false` (default)

Attribute	Description
`enabled`	Defines whether the user can interact with the control. • Data binding (Boolean) possible **Possible values:** `true`/`false` or expression for Data Binding The `enabled` attribute replaces the deprecated `disabled` attribute. *`enabled="true"`* thus corresponds to the deprecated *`disabled="false"`* attribute.
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Specifies what should happen when the control is full. **Possible values:** • *`ellipsis`* : With `...` show that the text is not finished • *`hidden`* : Break text, care for whole words • *`wordBreak`* : Break within the word • *`allow`* (default): Break text between words
`visible`	Defines whether the control is visible. • Data binding (Boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

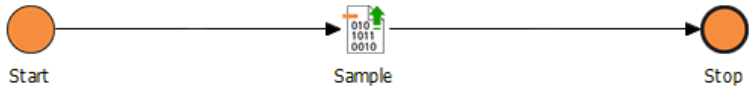
11.10.1 Example for `<HtmlDocument>`

In the web app definition (`.wad`), the `<HtmlDocument>` control is associated with the Technical Process `Sample.wrf`.

```
<WebApp xmlns="http://softproject.de/webapp/1.0"
        path="HtmlDocument">
   <Modules>
      <Module displayName="My Module" path="Module">
         <Components>
            <DetailComponent default="true"
                             displayName="Dashboard"
                             path="Dashboard"
                             process="load.wrf">
               <Properties>
                  <Property name="string" type="String"/>
               </Properties>
               <FlowLayout>
                  <HTMLDocument process="Sample.wrf" units="pixels"/>
               </FlowLayout>
            </DetailComponent>
         </Components>
      </Module>
   </Modules>
</WebApp>
```

The Technical Process `Sample.wrf` provides an HTML document:

Sample

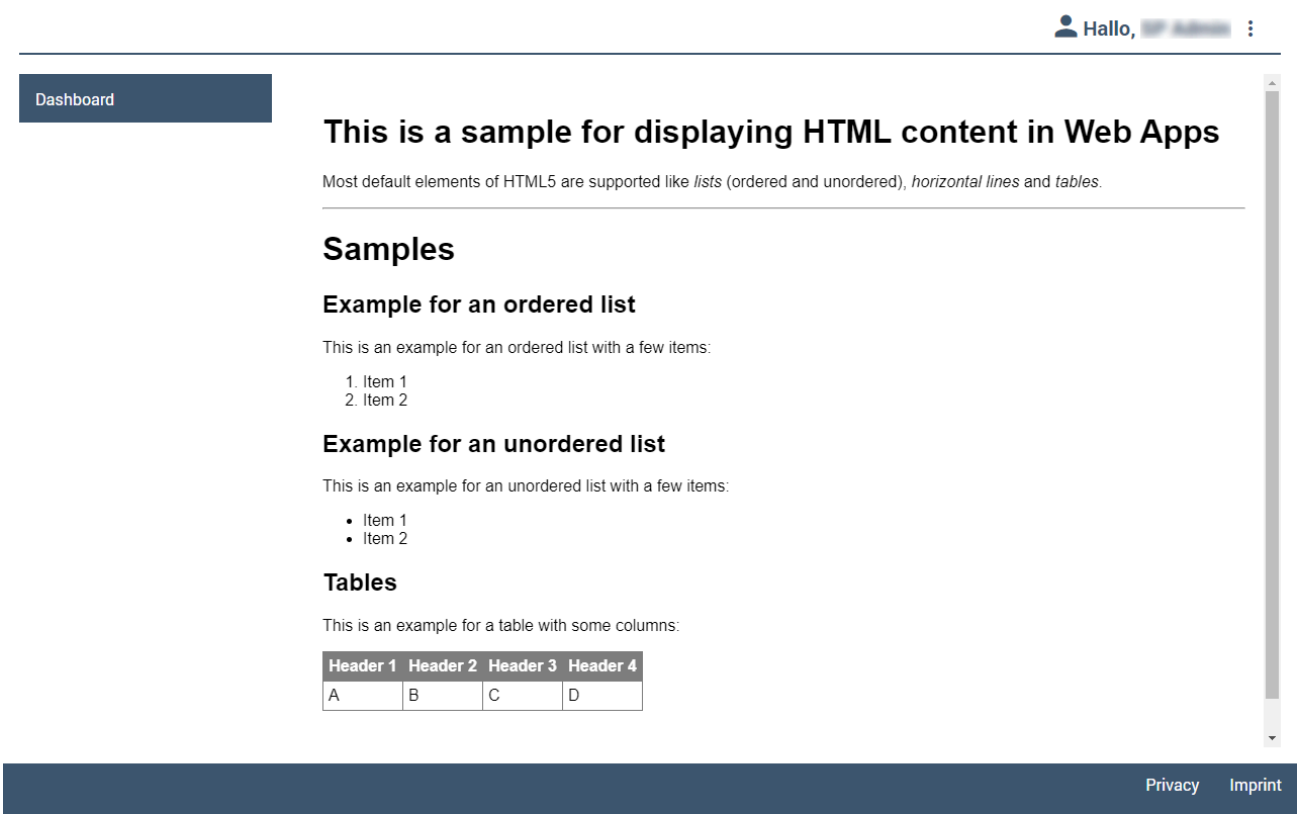


The HTML document supplied by the Technical Process looks like this:

```
<html>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
            <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
            <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0" />
            <style>
                body{
                    font-family: Arial, sans-serif;
                    padding: 10px;
                }

                table, td, th{
                    border: 1px solid grey;
                    padding: 5px;
                    border-collapse: collapse;
                }

                th{
                    background-color: grey;
                    color: white;
                }
            </style>
        </head>
        <body>
            <h1>This is a sample for displaying HTML content in Web Apps</h1>
            <p>Most default elements of HTML5 are supported like <i>lists</i>
(ordered and unordered), <i>horizontal lines</i> and <i>tables</i>.</p>
            <hr/>
            <h1>Samples</h1>
            <h2>Example for an ordered list</h2>
            <p>This is an example for an ordered list with a few items:</p>
            <ol>
                <li>
                    Item 1
                </li>
                <li>
                    Item 2
                </li>
            </ol>
            <h2>Example for an unordered list</h2>
            <p>This is an example for an unordered list with a few items:</p>
            <ul>
                <li>
                    Item 1
                </li>
                <li>
                    Item 2
                </li>
            </ul>
            <h2>Tables</h2>
            <p>This is an example for a table with some columns:</p>
```

```
            <table>
                <thead>
                    <th>Header 1</th>
                    <th>Header 2</th>
                    <th>Header 3</th>
                    <th>Header 4</th>
                </thead>
                <tr>
                    <td>A</td>
                    <td>B</td>
                    <td>C</td>
                    <td>D</td>
                </tr>
            </table>
        </body>
    </html>
```

The HTML document is displayed in the Web App as follows:



11.11 Image

`<Image>` controls are used to embed graphics.

Attribute	Description
`iconColor`	Defines the color of the icon. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* ⚠ Do not use a hash in front of the color value or a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*
`value`	Image that is displayed. Refers to a property (`Base64` or `Image`) within the component. Needed to dynamically display images. **Possible values**: File system path to the image file or data binding expression (Image, Base64, String, URL) ⚠ If an internal image file is used, the graphics file must be contained in the folder `Resources` directly below the Web App project. If the image is accessed via an external URL, then the attribute `type="external"` must be set for the input.
`background`	Defines a color for the background of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`border`	Defines if the control has a border. **Possible values:** *`true`* / *`false`*

Attribute	Description
`border-left`	Defines whether a left border is displayed **Possible values:** *`true`* / *`false`*
`border-right`	Defines whether a right border is displayed **Possible values:** *`true`* / *`false`*
`border-top`	Defines whether a top border is displayed **Possible values:** *`true`* / *`false`*
`border-bottom`	Defines whether a bottom border is displayed **Possible values:** *`true`* / *`false`*
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`units`	Defines the unit that applies to size specifications. **Possible values:** • *`pixels`*(default) • *`percents`*
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression
`width`	Width of the control **Possible values:** Integer

The size of the image can be adjusted using the general attributes `height` and `width`. The default value is in pixels.
If only height or width is specified, the aspect ratio is maintained. If both height and width are specified, the image will be distorted.

If size specifications are defined neither on the image control nor on the parent element (via the general attributes `height` and `width`), display problems may occur in some browsers.

11.11.1 "Select" Action within an image control

The **Select** action is created via the element `<SelectAction/>` within `<Image>` and is triggered as soon as an entry has been selected.

11.11.2 Example <Image> from static resource

The following example shows the usage of the `<Image>` tag.

```
<DetailComponent name="ExampleImage" path="ExampleImage" displayName="Example Image">
    <FlowLayout>
        <Image value="image.jpg"/>
    </FlowLayout>
</DetailComponent>
```

11.11.3 Example <Image> with data binding (dynamically from a Technical Process)

11.11.3.1 Images in Base64 format

The following example shows the usage of the `<Image>` tag with data binding.

```
<DetailComponent path="ExampleImageWithDataBinding" name="ExampleImageWithDataBinding
" displayName="Example Image with Data Binding" default="true" process="Image.wrf">
    <Properties>
        <Property name="image" type="Base64" />
    </Properties>
    <FlowLayout>
        <Image value="#image"/>
    </FlowLayout>
</DetailComponent>
```

The data provided by the Technical Process `Image.wrf` must have the following format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Ok>
    <image mediaType="image/jpeg">
        <!-- Base64String -->
    </image>
</Ok>
```

11.11.3.2 Images from an external URL

The following example shows the usage of the `<Image>` tag with data binding.

```
<DetailComponent
    path="ExampleImageWithDataBinding"
    name="ExampleImageWithDataBinding"
    displayName="Example Image with Data Binding"
    default="true"
    process="Image.wrf">
        <Properties>
            <Property name="image" type="Image" />
        </Properties>
    <FlowLayout>
        <Image value="#image"/>
    </FlowLayout>
</DetailComponent>
```

The data provided by the Technical Process `Image.wrf` must have the following format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Ok>
    <image mediaType="image/png" type="external">
       <![CDATA[https://softproject.de/de/wp-content/uploads/sites/2/2019/01/
LogoSoftProject.png]]>
    </image>
</Ok>
```

11.12 Card

With the Card control, cards can be integrated into X4 Web Apps, for example to integrate teasers or dashboards into applications. A card can contain a title and subtitle, various actions and a layout with any controls.

11.12.1 Create Card Control

A card control is created with the `<Card>` element. The element can only be used in the layout of one Detail Component.

Possible attributes:

In addition to the standard attributes of a control, the `<Card>` element can have the following attributes:

Attribute	Description
`icon`	Icon for the card control • Data Binding with Base64 values are possible **Possible values:** • Base64 encoded graphic file • Path to a graphic file as a character string (URI), e.g. *`clock.png`* • Specification of a material icon, e.g. *`icon:extension`* Press `Ctrl+Space` to get an overview of the available icons. The selection may differ from the actual Material Icons available. • The graphic file must be contained in the `Resources` folder directly within the web app project. Path specification relative to the `Resources` folder. • The material icon must be specified with the prefix `icon`, e.g. `icon:<MaterialIconName>`. This attribute cannot be used at the same time as `iconUrl`.
`iconColor`	Defines the colour of the map icon. **Possible values:** • Hexadecimal colour value, e.g. *`ff5a00`* Do not use a hash (#) before the colour value or a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. *`A200`*
`subtitle`	Subtitle of the card. • Data binding possible
`title`	Title of the card. • Data binding possible

Attribute	Description
`background`	Defines the background colour of the control. • This setting overwrites the default colour of the colour scheme! • This attribute does not work for the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`
`border`	Defines whether a border is displayed around the control. **Possible values:** *`true`* / *`false`*
`border-left`	Defines whether a left border is displayed **Possible values:** *`true`* / *`false`*
`border-right`	Defines whether a right border is displayed **Possible values:** *`true`* / *`false`*
`border-top`	Defines whether a top border is displayed **Possible values:** *`true`* / *`false`*
`border-bottom`	Defines whether a bottom border is displayed **Possible values:** *`true`* / *`false`*

Attribute	Description
`fontStretch`	Defines the width of the characters. ⓘ This attribute overrides the default width of the characters for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute does not work for the elements `Image`, `Map`, `Chart` and `HtmlDocument`.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default font style for this control. **Possible values:** • `italic:` italic font • `normal:` normal font (standard) • `oblique:` oblique font style (calculated) ⓘ • This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ • This attribute does not work for the elements `Image`, `Map` and `HtmlDocument`.
`foreground`	Defines a colour for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default colour of the colour scheme! • This attribute does not work for the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal colour value, e.g. `ff5a00` ⚠ Do not use a hash (#) before the colour value! Do not use a shortened notation of the colour value! • Colour code from the colour palette of the web app (cf. Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Specifies what happens when the control is full. **Possible values:** • *`ellipsis`* : With `...` show that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: break within the word • *`allow`*(default): Break text between words
`visible`	Defines whether the control is visible. • Data binding (Boolean) possible **Possible values:** *`true`* / *`false`* or string for data binding

11.12.2 Actions

Actions can be defined within the `<Card>` element.

> If a **select** action (`<SelectAction/>`) has been defined in the `<Card>` element, then no further actions can be defined and the entire `<Card>` element can be clicked. If no select action is specified, the `<Card>` element may contain any number of other actions.

```
<Card>
    <FlowLayout>
    ...
    </FlowLayout>
    <Actions>
        <CustomAction displayName="Card action"/>
    </Actions>
</Card>
```

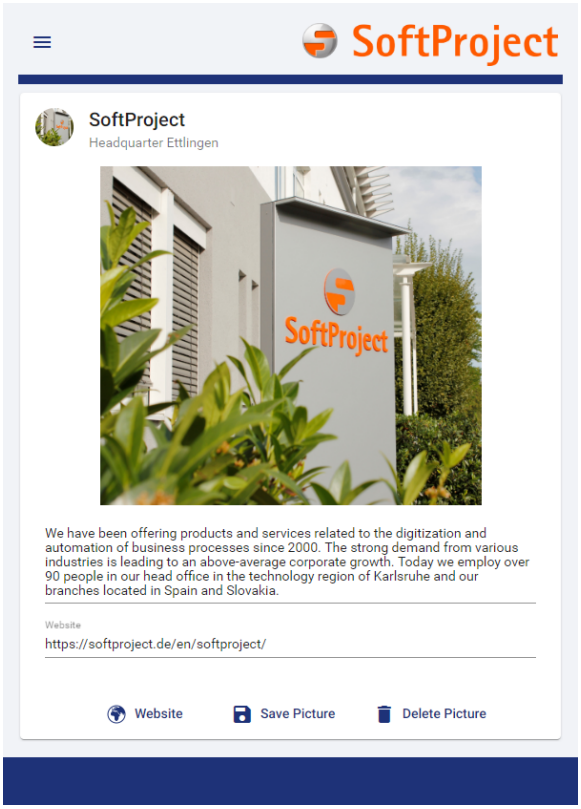
11.12.3 Examples

In this example, the attributes as well as the contents of the card control are defined.

Example structure

```
<Card border="true" icon="SPHeadquarters.jpg" title="SoftProject" subtitle="Headquart
ers Ettlingen">
    <FlowLayout>
        <Image resourceUrl="SPHeadquarters.jpg" width="400"/>
        <TextBox border="false" type="text" value="We have been offering products and
services related to the digitization and automation of business processes since 2000.
The strong demand from various industries is leading to an above-average corporate
growth. Today we employ over 90 people in our head office in the technology region of
Karlsruhe and our branches located in Spain and Slovakia." multiline="true"/>
        <TextBox type="url" value="https://softproject.de/en/softproject/"
displayName="Website"/>
    </FlowLayout>
    <Actions>
        <CustomAction icon="Website.png" iconPosition="left" displayName="Website"
externalLink="https://softproject.de/de/"/>
        <SaveAction displayName="Save Picture"/>
        <DeleteAction displayName="Delete Picture"/>
    </Actions>
</Card>
```

The above example leads to the following result:



11.13 Label

`<Label>` controls are used to display text. `<Label>` controls cannot be edited by the Web App user.

Attribute	Description
`value`	Contains the label text. • Translatable • Data binding possible **Possible values:** Any string
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines a color for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.13.1 Example <Label>

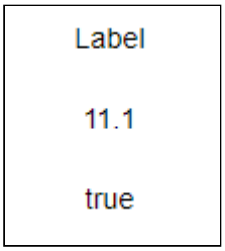
The following example shows the usage of the `<Label>` tag. In this example, the labels are filled via data binding.

```
<Properties>
    <Property name="number" type="Decimal" />
    <Property name="boolean" type="Boolean" />
</Properties>
<FlowLayout>
    <Label value="Label"/>
    <Label value="#number"/>
    <Label value="#boolean"/>
</FlowLayout>
```

The data that is displayed in the labels via data binding must be provided by a Technical Process in the following form:

```
<?xml version="1.0" encoding="UTF-8" ?>
<number>11.1</number>
<boolean>1</boolean>
```

The above code creates the following labels:



11.14 Link

`<Link>` controls are used to create hyperlinks. The hyperlink can be placed anywhere in the web app.

The `<link>` element may contain the following actions:

- Action „New“ `<NewAction/>`
- Action „Save“ `<SaveAction/>`
- Action „Delete“ `<DeleteAction/>`
- Action „File upload“ `<UploadAction/>`
- Action „Download file“ `<DownloadAction/>`
- Custom Action `<CustomAction/>`
- Action "Reload" `<ReloadAction>`

The following additional attributes can be defined for the `<Link>` element:

Attribute	Description
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true`/`false` or data binding expression
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • `left` (default) • `center` • `right`

Attribute	Description
`textOverflow`	Defines what happens if the page is full. **Possible values:** • `ellipsis`: Show with ... that the text is not finished • `hidden`: Break off text, paying attention to whole words • `wordBreak`: Break off within the word • `allow` (default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *true* / *false* or data binding expression

11.14.1 Example

```
<DetailComponent
    default="true"
    path="Dashboard">
    <GridLayout>
        <Link>
            <CustomAction
                externalLink="https://www.softproject.de"
                externalLinkTarget="new"
                displayName="Open website" />
        </Link>
    </GridLayout>
</DetailComponent>
```

11.15 ListView

`ListView` controls are used to display data objects (simple or complex) from lists. How a single object is displayed is defined as an item template. For this purpose, the display is defined within a layout (flow or Grid Layout). This display is adopted for each individual object. Since the data objects to be displayed are provided by a Technical Process, a corresponding Technical Process must be defined in the component that contains the `ListView`.

Attribute	Description
`elementWidth`	Defines the width of an element (corresponds to a list entry in the xml file). **Possible values:** Integer

Attribute	Description
`elementWidthUnits`	Defines the unit that applies to the sizes of the element. **Possible values:** • *`pixels`* • *`percents`* (default)
`MaxLength`	Defines the maximum number of entries to be displayed. If there are more entries in the list, the user can display them using a drop-down arrow. **Possible values:** Integer
`objectList`	*Required*. Source of the data. The number of entries defines the number of displayed elements. • Data binding (list) enabled **Possible values:** Any string
`background`	Defines a color for the background of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression ⓘ The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines a color for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`*(default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.15.1 "Select" Action within a ListView

The **Select** action is created via the element `<SelectAction/>` within a `<ListView>` and is triggered as soon as an entry has been selected in the ListView.

> The selected element is marked in the output with the attribute `selected="true"`.

11.15.2 Examples

11.15.2.1 List without actions

Input

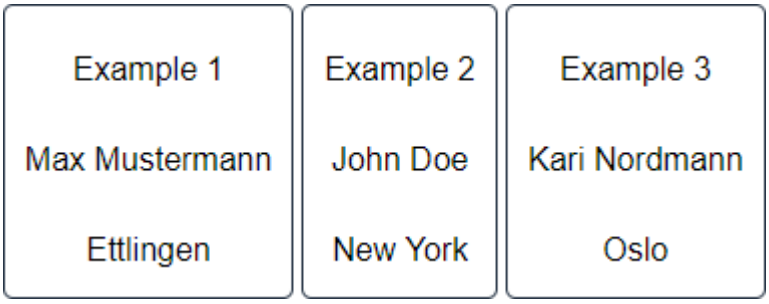
```
<DetailComponent name="ExampleListView" path="ExampleListView" displayName="Example
ListView" process="example.wrf">
    <Properties>
        <Property name="ListObject" type="List">
            <Property name="Caption" type="String" />
            <Property name="Name" type="String" />
            <Property name="City" type="String" />
        </Property>
    </Properties>
    <FlowLayout>
        <ListView objectList="#ListObject">
            <FlowLayout>
                <Label value="#Caption"/>
                <Label value="#Name"/>
                <Label value="#City"/>
            </FlowLayout>
        </ListView>
    </FlowLayout>
</DetailComponent>
```

Output

The data provided by the Technical Process have the following format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Ok>
    <ListObject>
        <Caption>Example 1</Caption>
        <Name>Max Mustermann</Name>
        <City>Ettlingen</City>
    </ListObject>
    <ListObject>
        <Caption>Example 2</Caption>
        <Name>John Doe</Name>
        <City>New York</City>
    </ListObject>
    <ListObject>
        <Caption>Example 3</Caption>
        <Name>Kari Nordmann</Name>
        <City>Oslo</City>
    </ListObject>
    ...
</Ok>
```

Above code creates the following ListView:



11.15.2.2 List with actions

Input

```
<DetailComponent
    path="Dashboard"
    displayName="Dashboard"
    process="DataInput.wrf"
    default="true">
    <Properties>
        <Property
            name="ListObject"
            type="List">
            <Property
                name="name"
                type="String" />
        </Property>
    </Properties>
    <FlowLayout>
        <ListView objectList="#ListObject">
            <SelectAction process="Response.wrf" />
            <FlowLayout>
                <TextBox value="#name" />
            </FlowLayout>
        </ListView>
    </FlowLayout>
</DetailComponent>
```

Output

The data provided by the Technical Process have the following format:

> The selected element is marked in the output with the attribute `selected="true"`.
> In order to delete all entries of the ListView completely, the attribute `empty="true"` must be set in the output.

```
<?xml version="1.0" encoding="UTF-8"?>
<Action>
   <ListObject selected="true">
      <name>Ettlingen</name>
   </ListObject>
   <ListObject>
      <name>New York</name>
   </ListObject>
   <ListObject>
      <name>Oslo</name>
   </ListObject>
</Action>
```

11.16 Maps

With map controls, maps from different vendors, including Google Maps and Openstreetmap, can be integrated into X4 Web Apps. Geodata can be displayed, locations searched and routes found.

11.16.1 Create map control

A map control is created with the element `<Map>`. The element can only be used in the layout of a Detail Component.

Possible attributes:

In addition to the standard attributes of a control, the element `<Map>` can have the following attributes:

Attribute	Description
`apiKey`	Defines the API key if required by the vendor. **Possible values:** API key ⓘ To use Openstreetmap, no API key is required. If an API key is entered when using Openstreetmap as a map provider, the API key is ignored. ⚠ If Google Maps is used as vendor, the following APIs have to be activated: • Maps JS API • Locations (for searching and routing) • Directions (for routing) ⚠ The `apiKey` attribute in the Map control overrides the `MapAPIKey` element in the Web App configuration `.wac`.

Attribute	Description
`initialPointLongitude`	Defines the longitude on which the map should be centered when it is loaded. • Data Binding with constant values possible **Possible values:** Longitude in degrees with decimal places (e.g. `6.75000`)
`initialPointLatitude`	Defines the longitude on which the map should be centered when it is loaded. • Data Binding with constant values possible **Possible values:** Latitude in degrees with decimal places (e.g. `50.41871`)
`restrictScrolling`	Restrict scrolling behavior in maps **Possible values:** • `true` (default): When scrolling with the mouse or finger on mobile devices, a message informing about alternative scrolling is displayed • `false`: Alternative scrolling behavior is disabled
`satelliteView`	Defines if the satellite view of the map is activated. **Possible values:** `true` / `false` (default) ⓘ The satellite view cannot be used if Openstreetmap is used as map vendor.
`selectedIndex`	Selected index ⓘ The attribute `selectedIndex` takes over the `index` value of a single marker (`SingleMarker`) or a marker group (`Markers`) as soon as they are selected. **Possible values:** String Binding
`vendor`	*Required*. Defines the map vendor. **Possible values:** • `OpenStreetMap` • `Google` • `Bing`

Attribute	Description
`zoomLevel`	Specifies the card's initial zoom level. ⓘ The values give an approximate indication of the meters shown on the map. Please note that there may be deviations in the scale bar, as the scale changes depending on the distance from the equator. **Possible values:** • `10` • `25` • `50` • `100` • `250` • `500` • `750` • `1000` • `2000` • `5000` • `10000` • `25000` • `50000` • `100000` • `2500000` • `5000000`
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression ⓘ The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute .`*
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*

Attribute	Description
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`*(default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.16.2 Search

To allow a Web App user to search for locations in the map control, the `<Search/>` element is used within `<Map>`.

```
<Map>   <Search/> </Map>
```

Possible attributes:

The `<Search />` element may have the following attributes:

Attribute	Description
`searchIconHorizontalPosition`	Defines the horizontal position of the icon relative to coordinates searched for. **Possible values:** • `left`: Icon is left to the coordinate • `center` (default): Icon is on the coordinate • `right`: Icon is right to the coordinate
`searchIconUrl`	Path to the graphics file to be used as icon. The icon marks the search result. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)
`searchIconVerticalPosition`	Defines the vertical position of the icon relative to the coordinates searched for. **Possible values:** • `top`: Icon is above the coordinate • `center` (default): Icon is on the coordinate • `bottom`: Icon is below the coordinate

11.16.3 Routing

To allow a Web App user to calculate routes in the map control, the `<Routing />` element is used within `<Map>` and, if necessary, after `<Search />`.

```
<Map ...>   <Routing/> </Map>
```

Possible attributes:

The `<Routing/>` element may have the following attributes:

Attribute	Description
`destinationIconHorizontalPosition`	Defines the horizontal position of the destination icon relative to the coordinates searched for. **Possible values:** • `left`: Icon is left to the coordinate • `center` (default): Icon is on the coordinate • `right`: Icon is right to the coordinate
`destinationIconUrl`	Path to the graphics file to be used as icon for the destination. The icon marks the destination of the route. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)
`destinationIconVerticalPosition`	Defines the vertical position of the destination icon relative to the coordinates searched for. **Possible values:** • `top`: Icon is above the coordinate • `center` (default): Icon is on the coordinate • `bottom`: Icon is below the coordinate
`startIconHorizontalPosition`	Defines the horizontal position of the start icon relative to the coordinates searched for. **Possible values:** • `left`: Icon is left to the coordinate • `center` (default): Icon is on the coordinate • `right`: Icon is right to the coordinate
`startIconUrl`	Path to the graphics file to be used as icon for the start. The icon marks the start of the route. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)

Attribute	Description
`startIconVerticalPosition`	Defines the vertical position of the start icon relative to the coordinates searched for. **Possible values:** • `top`: Icon is above the coordinate • `center` (default): Icon is on the coordinate • `bottom`: Icon is below the coordinate

11.16.4 Single marker

To display a single marker in a map control, the `<SingleMarker />` element is used within `<Map>` and, if necessary, after `<Search />` and `<Routing />`.

```
<Map>   ...     <SingleMarker/> </Map>
```

Possible attributes:

The `<SingleMarker />` element may have the following attributes:

Attribute	Description
`componentName`	Defines the name of the component for internal navigation. **Possible values**: String (Name of a component)
`description`	Defines the description of the marker. • Data binding possible **Possible values:** String incl. spaces
`externalLink`	Defines the link that is shown in the pop-up of the marker. • Data binding possible **Possible values**: String (URL)
`externalLinkTarget`	Defines whether the link (`externalLink`) is opened within the same tab or a new tab. **Possible values:** • `same`: The link is opened within the same tab • `new`: The link is opened in a new tab

Attribute	Description
`iconHorizontalPosition`	Defines the horizontal position of the icon relative to the given coordinates. **Possible values:** • `left`: Icon is left to the coordinate • `center` (default): Icon is on the coordinate • `right`: Icon is right to the coordinate
`iconUrl`	Path to the graphics file to be used as icon for the marker. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)
`iconVerticalPosition`	Defines the vertical position of the icon relative to the given coordinates. **Possible values:** • `top`: Icon is above the coordinate • `center` (default): Icon is on the coordinate • `bottom`: Icon is below the coordinate
`index`	Marker index • Data binding possible **Possible values:** Any string
`latitude`	*Required if `searchString` is not used.* Defines the latitude of the marker. • Data binding possible **Possible values:** Latitude in degrees with decimal places (e.g. `50.41871`) ⓘ This attribute cannot be used at the same time as `searchString`.
`longitude`	*Required if `searchString` is not used.* Defines the longitude of the marker. • Data binding possible **Possible values:** Longitude in degrees with decimal places (e.g. `6.75000`) ⓘ This attribute cannot be used at the same time as `searchString`.

Attribute	Description
`name`	Defines the name of the marker. • Data binding possible **Possible values:** String of alphanumeric characters
`navigationDisplayName`	Defines the text that is displayed in the marker pop-up instead of an external link or component name. • Data binding possible **Possible values:** String incl. spaces
`searchString`	*Required if `latitude/longitude` is not used.* Address line for searching for selected services of the vendor. • Data binding possible **Possible values:** String for a service search, e.g. `Restaurant` ⓘ This attribute cannot be used at the same time as the attributes `latitude/longitude`.

11.16.5 Marker set

To display a set of markers in a map control, the `<Markers />` element is used within `<Map>` and, if necessary, after `<Search />` and `<Routing />`.

```
<Map>   ...     <Markers/> </Map>
```

Possible attributes:

The `<Markers/>` element may have the following attributes:

Attribute	Description
`cluster`	Defines if markers are grouped if they are very close together on the map. **Possible values:** `true` / `false` (default) ⚠ `cluster` and `number` cannot be used at the same time!
`componentName`	Defines the name of the component for internal navigation. **Possible values**: String (Name of a component)

Attribute	Description
`data`	*Required*. Defines which data is used for the marker set. • Data binding required **Possible values:** Data binding expression
`description`	Defines the descriptions of the markers. • Data binding required **Possible values:** Data Binding expression relative to the expression in `data`
`externalLink`	Defines the link that is shown in the pop-up of the marker. • Data binding required **Possible values:** Data Binding expression relative to the expression in `data`
`externalLinkTarget`	Defines whether the link (`externalLink`) is opened within the same tab or a new tab. **Possible values:** • `same`: The link is opened within the same tab • `new`: The link is opened in a new tab
`iconHorizontalPosition`	Defines the horizontal position of the icon relative to the given coordinates. **Possible values:** • `left`: Icon is left to the coordinate • `center` (default): Icon is on the coordinate • `right`: Icon is right to the coordinate
`iconUrl`	Path to the graphics file to be used as icon for the markers. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)
`iconVerticalPosition`	Defines the vertical position of the icon relative to the given coordinates. **Possible values:** • `top`: Icon is above the coordinate • `center` (default): Icon is on the coordinate • `bottom`: Icon is below the coordinate

Attribute	Description
`index`	Marker group index • Data binding possible **Possible values:** Any string
`latitude`	*Required if `searchString` is not used*. Defines the latitude of a marker. • Data binding possible **Possible values:** Data Binding expression relative to the expression in `data`
`longitude`	*Required if `searchString` is not used*. Defines the longitude of a marker. • Data binding possible **Possible values:** Data Binding expression relative to the expression in `data`
`name`	Defines the names of the markers. • Data binding required **Possible values:** Data Binding expression relative to the expression in `data`
`navigationDisplayName`	Defines the text that is displayed in the marker pop-up instead of an external link or component name. • Data binding possible **Possible values:** String incl. spaces
`number`	Defines if the markers are numbered. **Possible values:** `true` / `false` (default) ⚠ `cluster` and `number` cannot be used at the same time!
`searchString`	*Required if `latitude/longitude` is not used*. Address line for searching for selected services of the vendor. • Data binding possible **Possible values:** String for a service search, e.g. `Restaurant` ⓘ This attribute cannot be used at the same time as the attributes `latitude/longitude`.

Attribute	Description
`textHorizontalPosition`	Defines the horizontal position of the numbering relative to the given coordinates. **Possible values:** • `left`: Numbering is left to the coordinate (default) • `center`: Numbering is on the coordinate • `right`: Numbering is right to the coordinate ⚠ Bing Maps automatically positions the numbering. Therefore this function has no effect on Bing Maps.
`textVerticalPosition`	Defines the vertical position of the numbering relative to the given coordinates. **Possible values:** • `top`: Numbering is above the coordinate (default) • `center`: Numbering is on the coordinate • `bottom`: Numbering is below the coordinate ⚠ Bing Maps automatically positions the numbering. Therefore this function has no effect on Bing Maps.

11.16.6 "Select" Action within a Map

The **select** action is created via the element `<SelectAction/>` within `<SingleMarker>`, `<Markers>` or `<SelectedLocationMarker>` and is triggered as soon as a corresponding marker is selected.

11.16.7 Examples

In this example, the properties are defined and a group of markers is displayed.

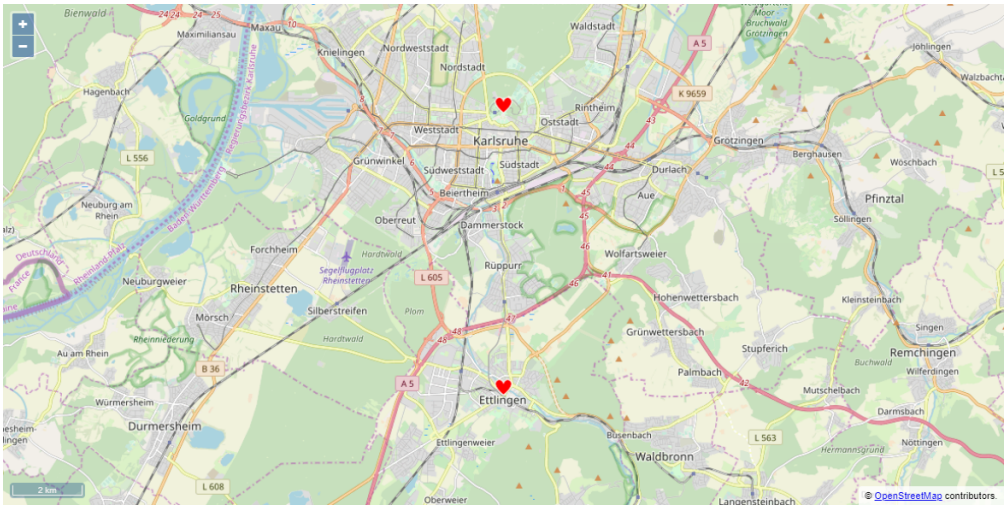
Example 1 Card Control (*.wad)

```
<Properties>    <Property name="favorites" type="List">         <Property name="lat"
type="Decimal" />      <Property name="lng" type="Decimal" />  </Property> </
Properties> ... <Map vendor="OpenStreetMap">  <Markers longitude="#lng" data="#favori
tes" latitude="#lat" iconUrl="heart.png" name="Favorites" description="Places, I have
visited." iconVerticalPosition="top"/> </Map>
```

Example 1: Data provided by the Technical Process (*.xml)

```
<OkList>    <favorites>         <lat>48.94061</lat>         <lng>8.40471</lng>  </
favorites>        <favorites>         <lat>49.01396</lat>         <lng>8.40445</lng>
</favorites> </OkList>
```

The above example leads to the following result:



In this example, two sets of markers are displayed and searching and routing are activated.

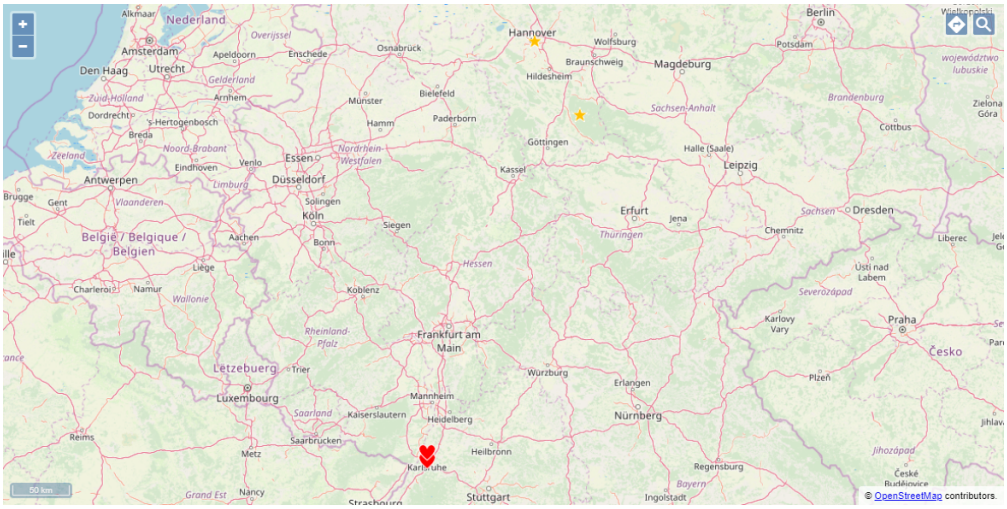
Example 2 Card control

```
<Properties>
    <Property name="favorites" type="List">
        <Property name="lat" type="Decimal" />
        <Property name="lng" type="Decimal" />
    </Property>
</Properties>

...

<Map vendor="OpenStreetMap">
    <Markers longitude="#lng" data="#favorites" latitude="#lat" iconUrl="heart.png"
name="Favorites" description="Places, I have visited." iconVerticalPosition="top"/>
</Map>
```

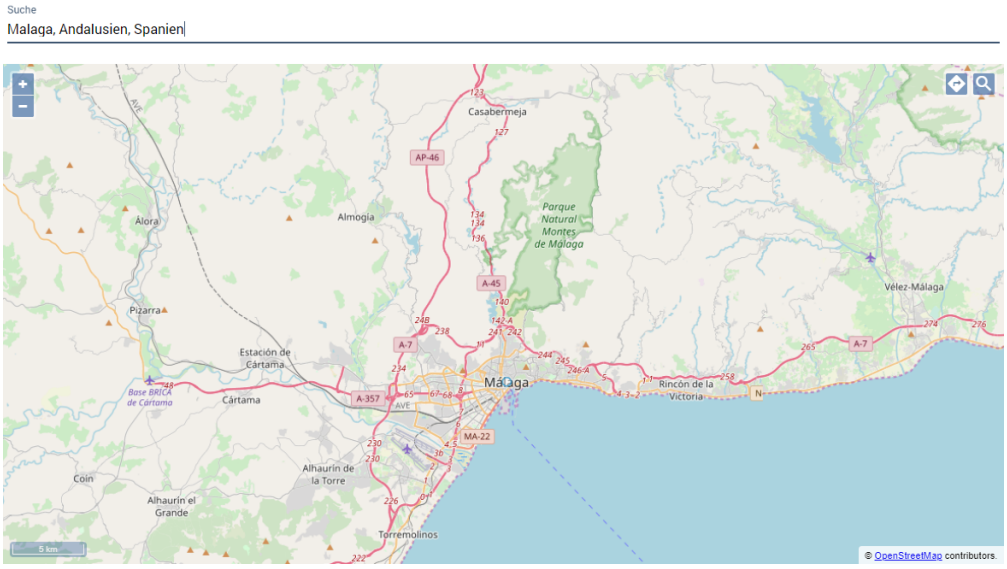
Example 2: Data provided by the Technical Process (*.xml)

```
<OkList>
    <favorites>
        <lat>48.94061</lat>
        <lng>8.40471</lng>
    </favorites>

    <favorites>
        <lat>49.01396</lat>
        <lng>8.40445</lng>
    </favorites>
</OkList>
```

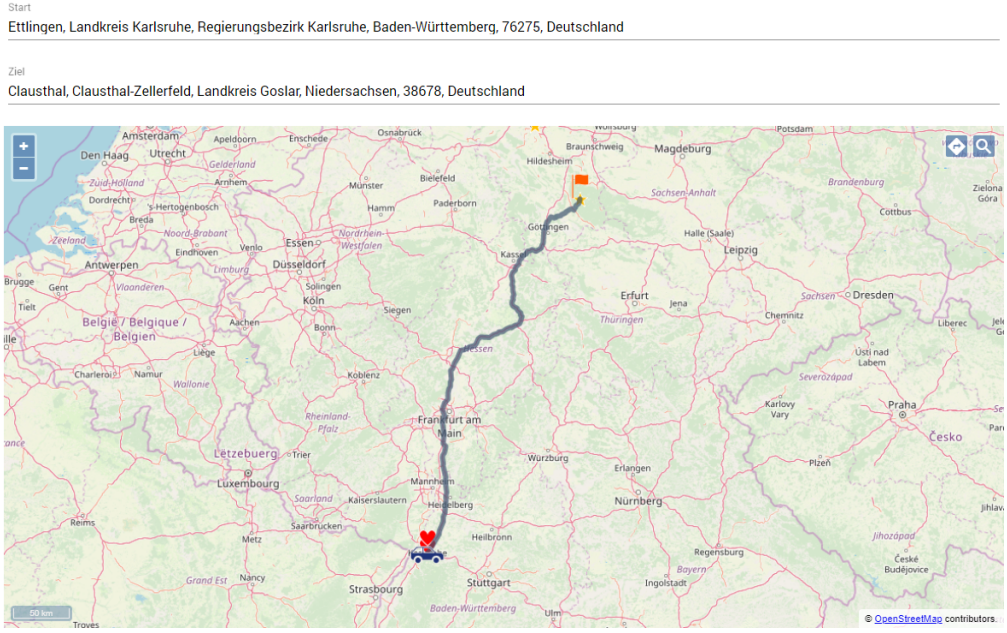
The above example leads to the following result:



Several groups of markers with different icons (Openstreetmap)



Searching (Openstreetmap)



Routing (Openstreetmap)

Other map providers are also available. The following example shows a map control with Google Maps.

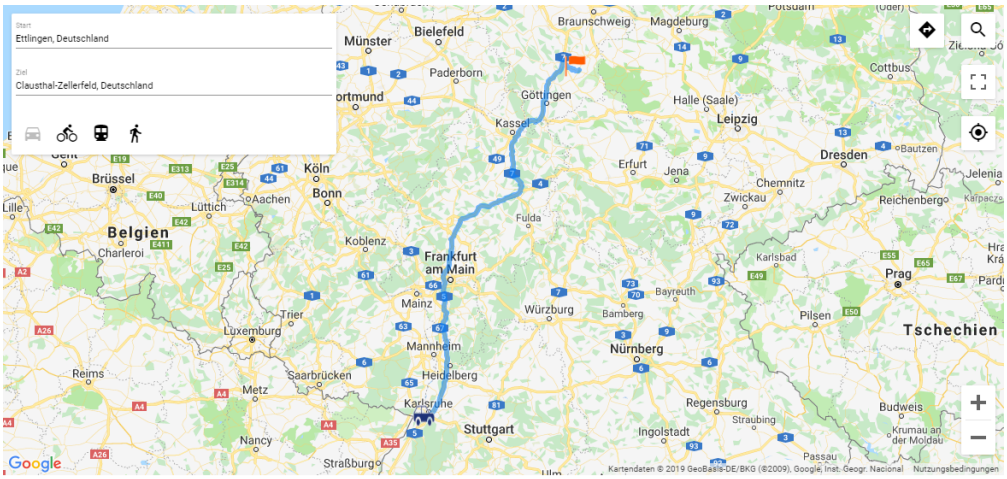
Example 2 Card control

```
<Properties>
    <Property name="favorites" type="List">
        <Property name="lat" type="Decimal" />
        <Property name="lng" type="Decimal" />
    </Property>
    <Property name="destinations" type="List">
        <Property name="lat" type="Decimal" />
        <Property name="lng" type="Decimal" />
    </Property>
</Properties>

...

<Map vendor="Google">
    <Search/>
    <Routing destinationIconUrl="finish.png" startIconUrl="car.png"
destinationIconVerticalPosition="top" destinationIconHorizontalPosition="center"/>
    <Markers longitude="#lng" data="#favorites" latitude="#lat" iconUrl="heart.png"
name="Favorites" description="Places, I have visited." iconVerticalPosition="top"/>
<Markers longitude="#lng" data="#destinations" latitude="#lat" iconUrl="star.png"
name="Destinations" description="Places, I want to visit."/>
</Map>
```

Example 2: Data provided by the Technical Process (*.xml)

```
<OkList>
    <favorites>
        <lat>48.94061</lat>
        <lng>8.40471</lng>
    </favorites>

    <favorites>
        <lat>49.01396</lat>
        <lng>8.40445</lng>
    </favorites>

    <destinations>
        <lat>51.8058</lat>
        <lng>10.3343</lng>
    </destinations>

    <destinations>
        <lat>52.3807</lat>
        <lng>9.7706</lng>
    </destinations>
</OkList>
```

The above example leads to the following result:



11.16.8 Set marker by click in the map

The `<SelectedLocationMarker>` element allows the user of a web app to set a marker by clicking on a specific location in the map. The location marked by the user can be further processed by a Technical Process.

```
<Map>   ...     <SelectedLocationMarker .../> </Map>
```

Possible attributes:

The `<selectedLocationManager/>` element may have the following attributes:

Attribute	Description
`longitude`	*Required.* Longitude of the marker **Possible values:** Property ⚠ A property must be set as the value for the attribute.
`latitude`	*Required.* Latitude of the marker **Possible values:** Property ⚠ A property must be set as the value for the attribute.

Attribute	Description
`iconUrl`	Path to the graphics file to be used as icon. The icon marks the selected location. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)

11.16.9 Using the user's location

Within the `<Map>` element, the `<Geolocation>` element can be used to query the user's location. If the `<Geolocation>` element is used, then a button is displayed in the map. By clicking on the button the location of the user is queried and marked on the map.

Possible attributes:

Attribute	Description
`iconUrl`	Path to the graphics file to be used as icon for the markers. The graphics file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. **Possible values:** String (URI)
`iconHorizontalPosition`	Horizontal position of the icon
`iconVerticalPosition`	Vertical position of the icon

11.17 PasswordBox

`<PasswordBox>` controls can be used to create password fields. Depending on the control definition, the appearance changes.

The following attributes can be defined for the `<PasswordBox>` element:

Attribute	Description
`displayName`	Title of the PasswordBox. Displayed small above the input/output field. • Translatable • Data binding possible **Possible values:** Any string
`value`	Content of the PasswordBox. • Translatable • Data binding possible **Possible values:** Any string

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`displayToggle`	Defines whether the password field can be toggled to display the password. **Possible values:** • `true`: The password field can be toggled to display the password. • `false`: The password field cannot be toggled. The password remains hidden.
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .

Attribute	Description
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*

Attribute	Description
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`foreground`	Defines a color for the foreground (texts etc.) of the control. ⓘ • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`* (default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.18 Radio button

`<RadioButton>` controls are used to create a group of buttons. The user can only select one radio button from this group at a time.

Attribute	Description
`displayName`	Label of the radio button • Translatable • Data binding possible **Possible values:** Any string
`optionValue`	*Required*. Technical value that is processed. • Data binding (string or boolean) possible **Possible values:** Any string
`value`	*Required*. Radio buttons with the same `value` form a group from which only one radio button can be selected at a time. • Data binding (string or boolean) required **Possible values:** Any string

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. ⓘ • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Sets the width of the single characters. ⓘ This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal`(default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` ⓘ This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.

Attribute	Description
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

Attribute	Description
`foreground`	Defines a color for the foreground (texts etc.) of the control. • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`* (default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.18.1 "Select" Action within a radio button

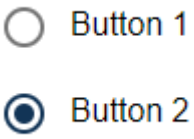
The **Select** action is created via the element `<SelectAction/>` within `<RadioButton>` and is triggered as soon as an entry has been selected.

11.18.2 Example `<RadioButton>`

The following example shows the usage of the `<RadioButton>` tag.

```
<DetailComponent name="ExampleRadioButton" path="ExampleRadioButton" displayName="Exa
mple RadioButton" process="exampleProcess.wrf">
    <FlowLayout>
        <RadioButton optionValue="true" value="#group" displayName="Button 1" />
        <RadioButton optionValue="false" value="#group" displayName="Button 2"/>
    </FlowLayout>
</DetailComponent>
```

Above code creates the following radio buttons:



11.19 TextBlock

`<TextBlock>` Controls are used to format text blocks within Web Apps.

`TextBlock` Controls are defined using the general attributes for controls.

The element `<TextBlock>` can contain the following elements:

- Free text
- `<Paragraph>`
- `<Binding>`
- `<Break>`
- `<Span>`
- `<Translation>`
- `<Link>`

11.19.1 Paragraph

The element `<Paragraph>` is used to divide texts into different paragraphs and can contain the following elements:

- Free text
- `<Binding>`
- `<Break>`
- `<Span>`
- `<Translation>`
- `<Link>`

A `Paragraph` can be defined using the following attributes:

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute does not work for the Maps control! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value or a shortened notation! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.

`fontStyle`	Defines the font style. This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`fontWeight`	Defines the font weight. This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`justifyText`	Defines the text alignment within the text block. **Possible values:** • `justify` (default): full justification • `center`: centered • `left`: left-justified • `right`: right-justified
`textOverflow`	Defines what happens if the control has reached its space limit. **Possible values:** • `ellipsis`(default): Show with `...` that the text is not finished • `hidden`: Break off text, paying attention to whole words • `wordBreak`: Break off within the word • `allow`: Break off text between words

Sample

```
<Paragraph justifyText="justify" textOverflow="allow">
    Paragraph
    <Binding value="#test" />
</Paragraph>
```

11.19.2 Span

The element <Span> is used to format text segments or single text elements and can contain the following elements:

- Free text
- <Binding>
- <Break>
- <Span>
- <Translation>

- `<Link>`

The element `Span` can be defined using the following attributes:

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute does not work for the Maps control! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value or a shortened notation! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *`20`*; *`20.8`*; *`.9`* • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120`* • Key words: *`xx-Small`*, *`x-small`*, *`small`*, *`medium`*, *`large`*, *`x-large`*, *`xx-large`*, *`smaller`*, *`larger`*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.

`fontStyle`	Defines the font style. This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`fontWeight`	Defines the font weight. This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

`foreground`	Defines the font color. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Sample

```
<Span fontWeight="ExtraBold" foreground="GaugeAxisMiddle">
    Span
    <Binding value="#test" />
</Span>
```

11.19.3 Break

The element `<Break>` is used to insert line breaks within a text section.

Sample

```
<Paragraph>
    SoftProject GmbH
    <Break/>
    Am Erlengraben 3
    <Break/>
    D-76275 Ettlingen
    <Break />
    E-Mail: info@softproject.de
</Paragraph>
```

11.19.4 Binding

The element `<Binding>` is used for Data Binding.

The element `Binding` is defined using the following attribute:

Attribute	Description
value	Binds a valid data binding string. ⓘ Note that constants are not allowed.

Sample

```
<Span>
    Span with binding
    <Break />
    value = '<Binding value="#test" />'
</Span>
```

11.19.5 Translation

The element <Translation> is used to specify a translation key for multilingual Web apps.

The element Translation is defined using the following attribute:

Attribute	Description
value	Specifies the translation key.

Sample

```
<Span>
    Span with translation
    <Break />
    value = '<Translation value="$TranslationKey" />'
</Span>
```

11.19.6 Link

The <Link> element is used to create a link. To generate the link, an action must be defined.

The following actions are supported:

- Upload Action
- Download Action
- Select Action
- CustomAction

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<DetailComponent xmlns="http://softproject.de/webapp/1.0">
    <FlowLayout>
        <Header value="Welcome to my new Web App!" />
        <TextBlock>
            <Link>
                <SelectAction
                    process="myProcess.wrf"
                    displayName="Click here"></SelectAction>
            </Link>
        </TextBlock>
    </FlowLayout>
</DetailComponent>
```

11.20 TextBox

<TextBox> controls are used to work with text, numbers or dates. Depending on the type of TextBox its appearance and behavior changes.

To trigger the Action "Select" when a TextBox item is deselected, a SelectAction action can be inserted within the TextBox control.

The following attributes can be defined for the <TextBox> element:

Attribute	Description
border	Defines whether a frame is displayed around the control. Works only if multiline="true". **Possible values:** *true* / *false*
border-left	Defines whether a left border is displayed. Works only if multiline="true". **Possible values:** *true* / *false*
border-right	Defines whether a right border is displayed. Works only if multiline="true". **Possible values:** *true* / *false*
border-top	Defines whether a top border is displayed. Works only if multiline="true". **Possible values:** *true* / *false*
border-bottom	Defines whether a bottom border is displayed. Works only if multiline="true". **Possible values:** *true* / *false*

Attribute	Description
`iconUrl`	Path to an image file or specification of a Material Icon to be used as an icon. • Data binding possible ⓘ • The image file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. • The Material Icon must be defined with the `icon` prefix, e.g. `icon:<MaterialIconName>` . **Possible values:** • String (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* ✓ With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons.
`multiline`	Defines whether multi-line inputs/outputs are possible in the TextBox. Works only for the `text` type. **Possible values:** *`true`* / *`false`*
`rows`	Defines the visible height of the text field. Works only with multiline text fields. More lines can be entered, these are then visible via scrolling. **Possible values:** Integer (default: *5*)
`showSeconds`	Defines whether the seconds should also be displayed for text boxes of the `datetime` type. **Possible values:** • *`true`*true: Seconds are also displayed in the date/time picker • *`false`*: Seconds are not displayed in the date/time picker (default).

Attribute	Description
`type`	Type of TextBox, changes behavior and appearance. **Possible values:** • *`date`* : Date with date picker • *`datetime`*: Date and time with date/time picker ⓘ With the attribute `showSeconds` you can additionally define whether the date/time picker also displays seconds. ⚠ The date and time are displayed according to the browser's time zone. • *`email`* : Email address • *`month`* : Month and year with date picker • *`number`* : Number with dial arrows (spinner) • *`search`* : Search field • *`tel`* : Phone number • *`text`* (default): Text • *`time`* : Time with dial arrows (spinner) • *`url`* : URL • *`week`* : Calendar week and year with date picker and dial arrows (spinner) ⚠ All values containing a date or time must be stored and provided in UTC.
`value`	Content of the TextBox. • Translatable • Data binding possible **Possible values:** Any string

Attribute	Description
`background`	Defines a color for the background of the control. • This setting overwrites the default color of the color scheme! • This attribute doesn't work with the `Maps` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`enabled`	Defines if the user can interact with the control. • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false" attribute`* .
`fontFamily`	Defines the font family. • This attribute overrides the default font of the Web App for this control. • If the attribute is defined on an element, the font family is inherited by the element's child elements, unless the child elements have the `fontFamily` attribute explicitly set. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • `MainFont`: Stored main font • Font code from the font palette, e.g. *`Font04`*

Attribute	Description
`fontSize`	Defines the font size. • This attribute overrides the default font size of the Web App for this control. • This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls. **Possible values:** • Any integer or decimal number with a dot as decimal separator, e.g. *20*; *20.8*; *.9* • Font size in pixels, e.g. *20px* • Font size in points, e.g. *18pt* • Font size compared to the font size of the parent element, e.g. *.8em* or *120* • Key words: *xx-Small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*, *smaller*, *larger*
`fontStretch`	Sets the width of the single characters. This attribute overrides the default width of the characters of the Web App for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded` This attribute doesn't work with the `Image`, `Maps`, `Charts` and `HtmlDocument` controls.

Attribute	Description
`fontStyle`	Defines the font style. ⓘ This attribute overrides the default style of the characters of the Web App for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated) ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.
`fontWeight`	Defines the font weight. ⓘ This attribute overrides the default font weight of the Web App for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight` ⓘ This attribute doesn't work with the `Image`, `Maps` and `HtmlDocument` controls.

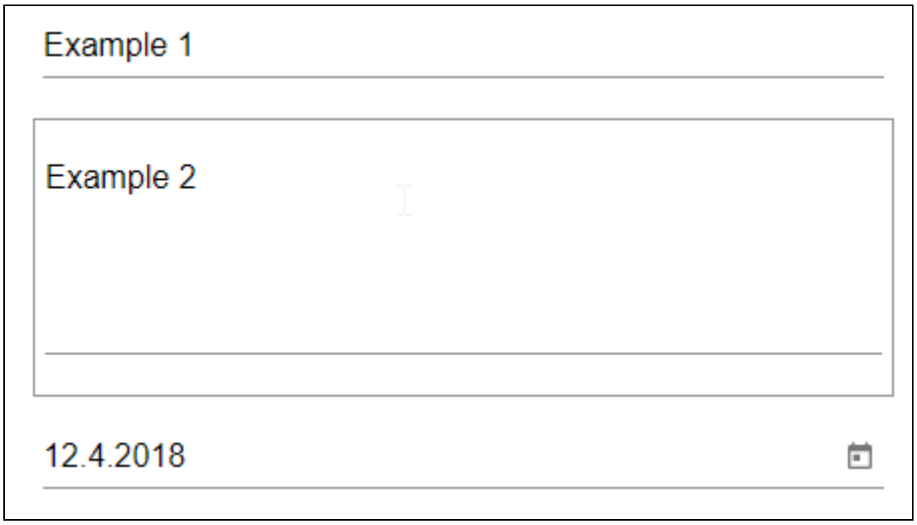
Attribute	Description
`foreground`	Defines a color for the foreground (texts etc.) of the control. • This setting overwrites the default color of the color scheme! • This attribute does not work with the `Map` and `HtmlDocument` controls! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`horizontalAlign`	Direction in which the elements flow. The order of the elements corresponds to their declaration. **Possible values:** • *`left`* (default) • *`center`* • *`right`*
`textOverflow`	Defines what happens if the page is full. **Possible values:** • *`ellipsis`*: Show with `...` that the text is not finished • *`hidden`*: Break off text, paying attention to whole words • *`wordBreak`*: Break off within the word • *`allow`* (default): Break off text between words
`visible`	Defines if the control is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

11.20.1 Example <TextBox>

The following example shows the usage of the `<TextBox>` tag.

```
<FlowLayout>
    <TextBox value="Example 1"/>
    <TextBox value="Example 2" multiline="true"/>
    <TextBox value="2018-04-12" type="date"/>
</FlowLayout>
```

The above code creates the following TextBoxes:



11.21 Video

`<Video>` controls are used to embed videos.

In addition to the standard attributes for controls, the following attributes are available for the element `<Video>`:

Attribute	Description
`type`	Defines the type of the embedded video. **Possible values:** • `YouTube`: Embed video using the video ID from YouTube • `File`: Embed video from the file system (data binding possible) ⓘ The HTML element `<video>` is used for this. ⚠ The browser must support the file type of the video!

Attribute	Description
`value`	Embedded video **Possible values:** • YouTube ID of the video (if `type="YouTube"`) • URI to a video file (if `type="File"`). Data binding is possible.
`autoplay`	Defines the auto-play behavior of the video. **Possible values:** • `true` (default): The video starts as soon as the page is loaded. • `false`: The video doesn't start automatically. ⚠ Some browsers or browser settings prevent auto-play!
`aspectRatioX` `aspectRatioY`	X-part or Y-part of the aspect ratio. ⚠ For the aspect ratio, both `aspectRatioX` and `aspectRatioY` must be specified! ⚠ If `height` and `width` are specified, then `aspectRatioX` and `aspectRatioY` are ignored! ⓘ By default, YouTube videos have an aspect ratio of 16:9. For videos from the file system, the aspect ratio of the file is applied.

Examples

Embedded YouTube video

```
<DetailComponent displayName="YouTube" path="YouTube" default="true">
    <FlowLayout>
        <Video type="YouTube" value="feY13KgCMow" />
    </FlowLayout>
</DetailComponent>
```

Embedded static resource

```
<DetailComponent displayName="Resource File" path="File">
    <FlowLayout>
        <Video type="File" value="video.mp4" />
    </FlowLayout>
</DetailComponent>
```

Embedded video from the file system using data binding

```
<DetailComponent displayName="Resource Binding" path="Binding">
    <Properties>
        <Property name="VideoPath" type="String" />
    </Properties>
    <FlowLayout>
        <ComboBox value="#VideoPath">
            <Option displayName="Video 1" value="video1.mp4" />
            <Option displayName="Video 2" value="video2.mp4" />
        </ComboBox>
        <Video value="#VideoPath" type="File" autoplay="true" aspectRatioX="16"
aspectRatioY="9" />
    </FlowLayout>
</DetailComponent>
```

Video from an external URL

```
<DetailComponent displayName="External Video" path="External" process="ExternalVideo.
wrf">
    <Properties>
        <Property name="ExternalVideo" type="Video" />
    </Properties>
    <FlowLayout>
        <Video value="#ExternalVideo" type="File" aspectRatioX="4" aspectRatioY="3" /
>
    </FlowLayout>
</DetailComponent>
```

For the data element provided by the Technical Process, the attribute `type="external"` must be used to specify that an external URL is used. Otherwise, an internal URL is expected and an error occurs.

```
<Ok>
    <ExternalVideo type="external">
        <!-- Video URL -->
    </ExternalVideo>
</Ok>
```

11.21.1 Data Binding for `value`

The `value` attribute of the `<video>` element can be filled via data binding. For this, the property must be of type `String`, `Base64` or `Video`.

If a YouTube video (`type="YouTube"`) is embedded, then a valid YouTube video ID must always be provided via data binding.

If video files are included (`type="File"`), then the following content is supported:

- Property type **String**: The property must contain a file path to a video file. The video file must be located directly in the `Resources` folder of the Web App project.
- Property type **Base64**: The property must contain a Base64-encoded video file. The media type of the video must be defined in the data element that is returned by the Technical Process as XML.

```
<MyVideo mediaType="video/mp4">
    <!-- Base64 -->
</MyVideo>
```

- Property type **Video**: The property can contain either a file path to a video file, a Base64-encoded video file or an external URL. What is provided is defined by the `type` attribute in the data element of the responding Technical Process. For `type` the values `resource` (default), `base64` or `external` can be specified.

Safari does not support videos that are included by file systems. Use external URLs that support buffering instead!

File system videos and Base64-encoded videos do not support buffering, but are loaded all at once. Therefore, use file system videos and Base64-encoded videos only for small videos. For larger videos, use external URLs that support buffering.

12 Actions

Within a component, buttons can be created that trigger actions. These buttons are always displayed at the top of the screen before the content of the component. Typical actions include deleting and saving of content. Actions are defined within the `<Actions>` element.The following actions can be defined:

- Action „New“: `<NewAction/>`
- Action „Save“: `<SaveAction/>`
- Action „Delete“: `<DeleteAction/>`
- Action „Select“: `(<SelectAction/>)`
- Action „File upload“: `<UploadAction/>`
- Action „Download file“: `<DownloadAction/>`
- Custom Action: `(<CustomAction/>)`
- Action "Reload": `<ReloadAction>`
- Action "Cancel": `<CancelAction>`
- Action "Select": `<SelectAction>`

  ⚠ The `SelectAction` action does not create a button.

All actions except custom actions and the `SelectAction` have predefined display names and icons. These may be overwritten if necessary.

The following table explains the attributes that can be used for all actions.

Attribute	Description
`background`	Defines a color for the background of the button. ⓘ This setting overwrites the default color of the color scheme! **Possible values:** • Hexadecimal color value, e.g. `ff5a00` ⚠ Do not use a hash (`#`) in front of the color value! Do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`

Attribute	Description
`componentName`	Name of the component to navigate to at the end of the action. **Possible values:** Any string ⓘ This attribute cannot be defined for the action "Cancel" and will result in a validation error if the attribute is set anyway.
`displayName`	Label of the button. • Translatable **Possible values:** Any string or translation key
`enabled`	Activates the action • Data binding (boolean) possible **Possible values:** `true` / `false` or data binding expression ⓘ The `enabled` attribute replaces the obsolete `disabled` attribute. *`enabled="true"`* thus corresponds to the obsolete *`disabled="false"`* attribute.
`fontFamily`	Defines the font family. This attribute overwrites the default font family of the Web App for this control. **Possible values:** Font code from the font palette, e.g. `Font04`
`fontSize`	Defines the font size. This attribute overrides the default font size of the Web App for this control. **Possible values:** • Font size in pixels, e.g. *`20px`* • Font size in points, e.g. *`18pt`* • Font size compared to the font size of the parent element, e.g. *`.8em`* or *`120%`* • Key words, e.g. *`small`* or *`larger`*

Attribute	Description
`fontStretch`	Defines the width of the single characters. This attribute overrides the default width of the characters for this control. **Possible values:** • `Condensed` • `Expanded` • `ExtraCondensed` • `ExtraExpanded` • `Medium` • `Normal` (default) • `SemiCondensed` • `SemiExpanded` • `UltraCondensed` • `UltraExpanded`
`fontStyle`	Defines the font inclination. This attribute overrides the default inclination of the characters for this control. **Possible values:** • `italic`: italic characters • `normal`: normal characters (default) • `oblique`: italic characters (calculated)
`fontWeight`	Defines the font weight. This attribute overrides the default weight of the characters for this control. **Possible values:** • `Black` • `Bold` • `DemiBold` • `ExtraBlack` • `ExtraBold` • `ExtraLight` • `Heavy` • `Light` • `Medium` • `Normal` (default) • `Regular` • `SemiBold` • `Thin` • `UltraBlack` • `UltraBold` • `UltraLight`

Attribute	Description
`foreground`	Defines a color for the foreground (texts, icons) of the button. • This setting overwrites the default color of the color scheme! • If a custom icon is used for the action with the `iconUrl` attribute, this icon is not displayed in the custom color. The graphic file that is used as icon must be provided directly in the desired color. **Possible values:** • Hexadecimal color value, e.g. `ff5a00` Do not use a hash in front of the color value! do not use a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. `A200`
`iconColor`	Defines the color of the icon. **Possible values:** • Hexadecimal color value, e.g. *`ff5a00`* Do not use a hash in front of the color value or a shortened notation of the color value! • Color code from the color palette of the Web App (see Theming), e.g. *`A200`*
`iconPosition`	Defines the position of the icon on the button. **Possible values:** • `left`: The icon is displayed on the left of the button (default). • `right`: The icon is displayed on the right of the button.

Attribute	Description
`iconUrl`	Path to an image file or specification of a Material Icon to be used as an icon. • The image file must be contained in the folder `Resources` directly below the web app project. Path relative to folder `Resources`. • The Material Icon must be defined with the `icon` prefix, e.g. `icon:<MaterialIconName>` . **Possible values:** • String (URI), e.g. *`clock.png`* • `icon:<MaterialIconName>` , e.g. *`icon:extension`* With `Ctrl+Space` you get an overview of the available icons. The selection may differ from the actual available Material Icons.
`process`	Path to the `.wrf` file. The Technical Process must be contained in the folder `Services/Processes`. Path relative to folder `Services/Processes`. **Possible values:** String (URI) This attribute cannot be defined for the action "Cancel" and will result in a validation error if the attribute is set anyway.
`selectionNeeded`	Defines if an entry in the list has to be selected to perform the action. Works only in list components. **Possible values:** *`true`* / *`false`*
`tooltip`	Defines the text that is displayed as tool-tip of the action when the action button is hovered with the cursor. **Possible values:** String If the value of the `tool-tip` attribute is empty, no tool-tip is displayed.
`visible`	Defines if the button for the action is visible. • Data binding (boolean) possible **Possible values:** *`true`* / *`false`* or data binding expression

Each of the above actions can be extended with the `<Parameters>` element. Within the `<Parameters>` element, you can set further definitions using the `<Parameter>` element.

The `<Parameter>` element contains the following mandatory attributes:

Attribute	Description
`key`	Name of the parameter • Data binding (boolean) possible **Possible values:** string or data binding expression
`value`	Parameter value • Data binding (boolean) possible **Possible values:** string or data binding expression

12.1 Action "New"

The action **New** is created with the element `<NewAction/>` within `<Actions>`. The action **New** creates a new entry. To do this, a new object is created in the detail structural element. The fields contained therein are empty.

For the **New** action the standard attributes for Actions can be defined.

The attribute `componentName` is required for the element `<NewAction/>`.

Example

Example action "New"

```
<DetailComponent path="Dashboard" displayName="Dashboard" default="true">
    <Actions>
        <NewAction componentName="personDetail"/>
    </Actions>
    <FlowLayout>
    …
    </FlowLayout>
</DetailComponent>
```

Above code creates the following action:

12.2 Action "Save"

The action **save** is created with the element `<SaveAction/>` within `<Actions>`. The action **save** saves entered data. The status depends on the validation status.

> For the **save** action the standard attributes for Actions can be defined.

Example

Example action "Save"

```
<DetailComponent path="Dashboard" displayName="Dashboard" default="true">
    <Actions>
        <SaveAction componentName="personDetail"/>
    </Actions>
    <FlowLayout>
        …
    </FlowLayout>
</DetailComponent>
```

Above code creates the following action:

12.3 Action "Delete"

The action **delete** is created with the element `<DeleteAction/>` within `<Actions>`. The action **delete** deletes data.

> For the **delete** action the standard attributes for Actions can be defined.

Example

Example action "Delete"

```
<DetailComponent path="Dashboard" displayName="Dashboard" default="true">
    <Actions>
        <DeleteAction/>
    </Actions>
    <FlowLayout>
        …
    </FlowLayout>
</DetailComponent>
```

Above code creates the following action:

12.4 Action "Upload"

The action **Upload** is created with the element `<UploadAction/>` within `<Actions>`. The action **Upload** enables to upload data into the web application.

The Technical Process that receives the uploaded file must return a confirmation or an error message. If the confirmation contains a message, this message is displayed as a pop-up window. Pop-up windows with error messages are displayed in any case.

> The attribute `process` is required for the element `<UploadAction/>`.

In addition to the standard attributes for Actions, the element `<UploadAction />` can have the following attributes:

Attribute	Description

`acceptedFileTypes`	File type to filter by in the selection dialog.
	• Data binding possible
	⚠ This attribute filters only by file types. The filter can be reset by the user to display and select all file types.
	Possible values: MIME types, also with wildcards (`*`), e.g.
	• `text/*`: Text files
	• `image/*`: Graphic files
	  • `image/jpeg`: Only JPG files
	  • `image/png`: Only PNG files
	• `video/*`: Video files
	• `audio/*`: Audio files
	• `application/*`: Files that are linked to a specific program
	• `multipart/*`: Multipart data
	• `message/*`: Messages
	• `model/*`: Files that represent multidimensional structures

`fileId`	Identifier that can be used by the process developer. • Data binding possible. **Possible values:** Any string
`maxSizeMB`	Maximum file size in mebibyte. **Possible values:** Integer

Format of the uploaded data

The data is provided to the Technical Process that is defined with the attribute `process` in a specific input format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<File>
    <FileId>....</FileId>
    <FileName>....</FileName>
    <FileType>....</FileType>
    <FileData>
    ....Base64 data....
    </FileData>
</File>
```

The data in the element `<FileData>` can be decoded with the Base64 Converter, for example.

Example

Example action "Upload file"

```
<DetailComponent path="Dashboard" displayName="Dashboard" default="true">
    <Actions>
        <UploadAction />
    </Actions>
    <FlowLayout>
        …
    </FlowLayout>
</DetailComponent>
```

Above code creates the following action:

For further information visit the section

- Messages

12.5 Action "Download"

The action **Download** is created with the element `<DownloadAction/>` within `<Actions>`. The action **Download** enables to download data from the web application.

The attributes `fileId` and `fileName` are required for the element `<DownloadAction/>`.

In addition to the standard attributes for Actions, the element `<DownloadAction />` can have the following attributes:

Attribute	Description
`data`	Use only, if a base64 file is provided for download! Defines which data is to be downloaded. **Possible values:** Data binding expression The Data Binding expression must refer to a property of type `base64`!

Attribute	Description
`fileId`	*Required*. Identifier that can be used by the process developer. • Data binding possible. **Possible values:** Any string
`fileName`	*Required*. Name of the file to be downloaded. It is also used as the file name of the downloaded file by the user. • Data binding possible **Possible values:** Any string

The action "Download File" not only supports the download of data from a Technical Process, but also base64 type properties can be downloaded. the attribute `data` must be used instead of the attribute `process`.

Example Download Base64 File

```
<Property name="data" type="Base64"/>
...
<DownloadAction fileId="tst" fileName="test.ext" data="#data"/>
```

12.6 Custom Action

The custom action is created with the element `<CustomAction/>` within `<Actions>`. With a custom action simple actions without predefined text or predefined icon can be implemented.

In addition to the standard attributes for actions, the element `<CustomAction/>` can have the following attributes:

Attribute	Description
`displayName`	Label of the button. • Translatable **Possible values:** Any string or translation key

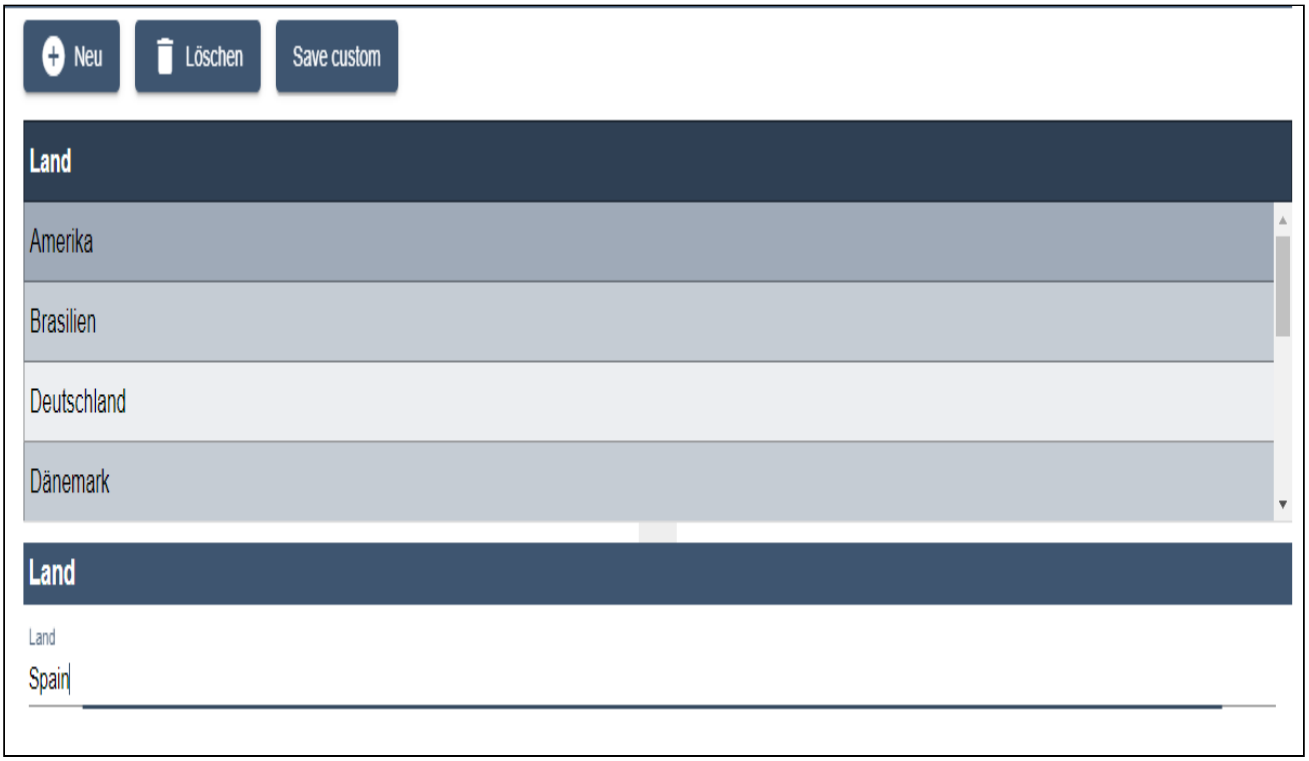
Attribute	Description
`externalLink`	Allows the navigation to an external website by defining an external link. • Data binding possible **Possible values:** String (URL) ⓘ The external link can also be provided by a Technical Process. A Technical Process (attribute `process`) that contains the external link in its response must be linked to the action: `<Ok externalLink="http://www.google.de" />`
`externalLinkTarget`	Defines whether the link (`externalLink`) is opened within the same tab or a new tab. **Possible values:** • `same`: The link is opened within the same tab • `new`: The link is opened in a new tab
`validated`	Allows validation of the input **Possible values:** • `true`: The validation is enabled ⓘ If the validation fails, the button is disabled. • `false`: The validation is not enabled (default) ⓘ If the validation fails, the button will not be disabled.
`url`	Allows the navigation to an external website by defining an external link. • Data Binding

Example

Example action "CustomAction"

```
<DetailComponent path="Dashboard" displayName="Dashboard" default="true">
    <Actions>
    <CustomAction componentName="CountryList"
                  displayName="Save custom"
                  validated="true"
                  process="/WebApp/Administration/Country/CreateOrUpdateCountry.wrf"/
>
    </Actions>
    <FlowLayout>
        …
    </FlowLayout>
</DetailComponent>
```

The above example creates the custom action *Save customer*:



12.7 Action "Reload"

The action **Reload** is created via the element `<ReloadAction/>` within `<Actions>`. The **Reload** action allows the properties of a component to be periodically reloaded (using a Technical Process specified in `process`) or a navigation to be triggered (specified using `componentName`).

⚠ The periodic reloading of data can cause a high load and should only be used in rare cases.

The element `<ReloadAction/>` can have the following attributes:

Attribute	Description
`componentName`	Name of the component to be navigated to at the end of the action. ⓘ Required, if the attribute `process` is not set. **Possible values:** Any string
`enabled`	Defines whether the reload action is started periodically or not. • Data binding (Boolean) possible **Possible values:** `true`/`false` or expression for data binding
`process`	Path to the `.wrf` file. The Technical Process must be contained in the folder `Services/Processes`. Path relative to folder `Services/Processes`. ⓘ Required, if the attribute `componentName` is not set. **Possible values:** String (URI)
`refreshTime`	Time until the next reload in seconds **Possible values:** Any integer

12.8 Action "Cancel"

The **Cancel** action is created via the `<CancelAction/>` element within `<Actions>` and effects that an open Overlay Structural Element or Grid component can be closed without saving and reloading the data.

ⓘ For the **Cancel** action the standard attributes for Actions can be defined. However, the attributes `componentName` and `process` can not be defined for this action and cause validation errors, if they are set.

Example

Sample action "Cancel"

```
<DetailComponent>
    <Actions>
        <CancelAction displayName="Cancel"/>
    </Actions>
    <FlowLayout>
        …
    </FlowLayout>
</DetailComponent>
```

Above code creates the following action:

12.9 Action "Select"

The **Select** action is created via the element `<SelectAction/>` and can, for instance, call a Technical Process or an external web site.

In addition to the standard attributes for Actions,the element `<SelectAction />` can have the following attributes:

Attribute	Description
`externalLink`	Allows the navigation to an external website by defining an external link. • Data binding possible **Possible values:** String (URL) ⓘ The external link can also be provided by a Technical Process. A Technical Process (attribute `process`) that contains the external link in its response must be linked to the action: `<Ok externalLink="http://www.google.de" />`
`externalLinkTarget`	Defines whether the link (`externalLink`) is opened within the same tab or a new tab. **Possible values:** • `same`: The link is opened within the same tab • `new`: The link is opened in a new tab

Attribute	Description
`url`	Allows the navigation to an external website by defining an external link. • Data Binding

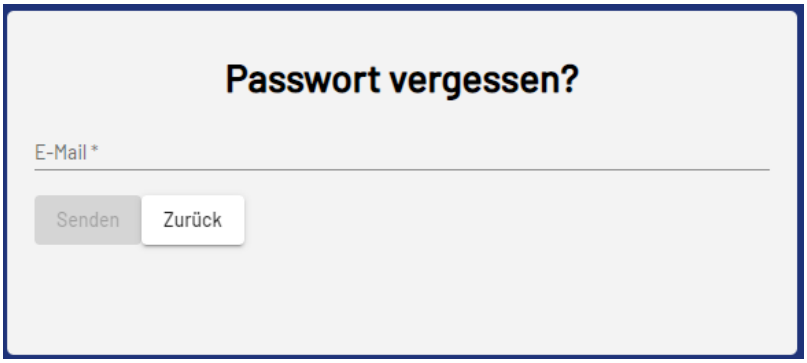
12.9.1 Example

```
<DetailComponent
    xmlns="http://softproject.de/webapp/1.0">
    <Properties>
    <Property name="text" type="Integer"></Property>
    <Property name="date" type="DateTime"></Property>
    </Properties>
    <FlowLayout>
        <Header value="Welcome to my new Web App!" />
        <TextBox value="#text" type="number">
            <SelectAction process="sendData.wrf"></SelectAction>
        </TextBox>
        <TextBox value="#date" type="datetime">
            <SelectAction process="sendData.wrf"></SelectAction>
        </TextBox>
    </FlowLayout>
</DetailComponent>
```

12.10 Action "Forgot password"

The Action **Forgot Password** is created using the `<ForgotPasswordAction/>` element and calls a page to enter the email address for sending the password.

> ⚠ The Forgot Password action can be used only within the `.login` login page.



12.10.1 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Login
    xmlns="http://softproject.de/webapp/1.0">
    <DetailComponent>
        <FlowLayout>
            <Button>
                <ForgotPasswordAction displayName="$ForgotPassword"/>
            </Button>
        </FlowLayout>
    </DetailComponent>
</Login>
```

12.11 Action "LogoutAction"

The action **LogoutAction is created with the element** `<LogoutAction>` und logs out of the Web App. The Web App page is reloaded.

> For the **LogOut** action the standard attributes for Actions can be defined.

12.11.1 Example

```
<DetailComponent>
    <FlowLayout>
        <TextBox value="#TEST"/>
        <Button>
            <LogoutAction process="logout.wrf"/>
        </Button>
    </FlowLayout>
</DetailComponent>
```

13 HeaderSection

You can use the HeaderSection file to individually design the header of a Web App. All layouts and controls of the X4 Web Apps are available.

13.1 Create HeaderSection

The HeaderSection is named according to the Web App project und has the file ending .headersection.

There can only be one .headersection file per Web App project.

The element `<HeaderSection>` creates the header.

All layouts and controls of X4 Web Apps can be used in the header section.

⚠ The custom header overwrites the standard header.

1. Right-click on the Web App project.
2. Select **New > HeaderSection**.
   The headersection file is created.

13.1.1 <HeaderSection>

The following attribute can be defined for the element `<HeaderSection>`:

Attribut	Description
`process`	Path to the `.wrf`file returning the data for the header. The process needs to be stored in the folder `Services/Processes`. **Possible Values:** String (URI)

14 Integrating JavaScript in Web Apps

X4 BPMS developers can use the **Web App Control Project** component to extend X4 Web Apps by custom JavaScript functions based on web technologies and the X4 ESB with its web service interface. This approach allows to access the data and features of the web app via JavaScript API functions. You should not only have a detailed knowledge of the data, workflows, and common web technologies such as HTML, CSS, and JavaScript, but you should also be well grounded in XML, XSLT, and XPath.

14.1 Working with Web App Control Projects

You can use the **Web App Control Project** project type to extend X4 Web Apps with custom JavaScript functions in order to access the data and actions of the web app via JavaScript API functions. You can use `<CustomControls>` to incorporate your own JavaScript code into a web app and call actions in order to interact with the web app. In addition, you can control the layout by setting property parameters. In order to use a custom control in a web app, you just have to reference the Web App Control Project in a Web App Project. A Web App Control Project contains the content which is displayed in the Web App.

14.1.1 Creating a Web App Control Project

To create a new Web App Control Project:

1. In the X4 Designer, right-click in the **Projects** view and choose **New** from the context menu.
2. Choose **Web App Control Project...**.

   > Alternatively, you can click **New** in the **File** menu and choose **Web App Control Project...**.

3. Enter a project name in the **Project Creation Wizard** dialog.
4. Choose **Finish**.

Web App Control Projects have a pre-defined and non-modifiable folder structure which is created automatically when creating a new Web App Control Project.

`Sources`	Once a project has been created, this folder contains three files: • **index.html**: This file contains a basic HTML structure which you can extend by the desired elements. • **x4-webapp-api.d.ts**: This file contains the current API declarations (needed by TypeScript developers). • **x4-webapp-api.js**: This file contains the current JavaScript API. See Available API Functions for a list of the available API functions with a description. ⓘ The **Sources** folder cannot be deleted, moved, or renamed. You can, for example, also create a `.css` file in the **Sources** folder.
`<Project>.webcontrol`	Project-related file which is created automatically when creating a project. The first part of the file name is the project name. If needed, you can add a definition for the cookie consent in this file. To do so, you can enter any text in the fieds or provide information for a third-party service. Each Web App Control Project contains exactly one `.webcontrol` file. If you would like to reference more than one Web App Control Project in a Web App Project, you create a new `.detail` component in the **Components** folder of the Web App Project for every additional Web App Control Project.

Note:
Using the developer tools, you can display all files from a Web App Control Project in the browser without authentication.

14.1.2 Using Custom Controls in a Web App

ⓘ **Prerequisite**
You have a Web App Project containing at least one technical process. For more information on creating Web App Projects, see Creating a new Web App Project.

The following sections explain how to connect a Web App Control Project with a Web App Project in order to use one or more custom controls in the Web App.

To embed a custom control:

1. Open the Web App Project and double-click on the ***<Project>*.wac** file. Choose `SAMEORIGIN` in the **X-Frame-Options** drop-down in the **HTTP Security Header** section.
2. Add the desired properties in the **Dashboard.detail** file in the **Components** folder.
   In this file, you insert the desired custom controls using the `<CustomControl>` element and reference the Web App Control Project you have created previously via the mandatory `project` attribute.

> You can create additional detail components in the **Components** folder which you use, for example, to set layout properties.

In addition, you can define the following attributes for the `<CustomControl>` element:

Attribute	Description
`horizontalAlign`	Direction of the element flow. The order of the elements corresponds to their declaration. **Possible values:** • `left` (default) • `center` • `right`
`textOverflow`	Specifies which action is to be performed when the control is full. **Possible values:** • `ellipsis`: Use `...` to indicate that the text is continued • `hidden`: truncate text, do not break whole words • `wordBreak`: break within words • `allow` (default): wrap text within words
`visible`	Specifies whether the control is visible. • Data binding (boolean) possible **Possible values:** `true`/`false` or string for data binding

Within the `<CustomControl>` element, you use the `<APIAction>` element to specify the API functions to be used in the web app. The name of the function to be called is specified with the mandatory `name` attribute.

In addition, you can define the following attributes for the `<APIAction>` element:

Attribute	Description
`componentName`	Name of the component which is the target of the navigation at the end of the action. **Possible values:** any string
`externalLink`	Specifies an URL here. **Possible values:** any URL
`externalLinkTarget`	Specifies an external link. **Possible values:** • `new`(default): The page is opened in a new tab. • `same`: The page is opened in the same tab.
`process`	▪ Path to the `.wrf` file. The technical process must be contained in the `Services/Processes`folder. The path is specified relatively to the `Services/Processes` folder. **Possible values:** any string (URI)

3. In the Web App Control Project, adjust the **index.html** file in the **Sources** folder and add the desired elements.
4. Open the web app in the browser. You will find a list of the supported browsers here.

14.1.3 Available API Functions

Each Web App Control Project contains the **x4-webapp-api.d.ts** and **x4-webapp-api.js** files in the **Sources** folder. The following functions are available for the `X4API` class declared in this files:

Function	Description
`init(): Promise<boolean>;`	Initializes the connection to the parent custom control component. **Return value:** • Promise with a boolean value specifying whether the JavaScript API is supported by the current version of the web app

Function	Description
`getPropertyDefinitions(): Promise<any[]>;`	**Return value:** • Promise with an array value of property definitions
`subscribeToDataModelChanges(func: (v: any) => void): string;`	Developers can use this function to react to data model changes by setting the function to process incoming modifications. **Return value:** • ID that can be used for the `Unsubscribe` function later
`subscribeToPropertyChanges(propertyName: string, func: (v: object[]) => void): string;`	Developers can use this function to react to property changes by setting the function to process incoming modifications. **Return value:** • ID that can be used for the `Unsubscribe` function later
`getPropertyValue(v: string): Promise<string>;`	**Return value:** • Promise with a string representation of a property value
`getPropertyValidationErrors(v: string): Promise<string[]>;`	**Return value:** • Promise with an array of validation messages (the array is empty if there are no errors)
`setPropertyValue(n: string, v: string): void;`	Sets the property value by name.
`unsubscribeDataModelChanges(sid: string): void;`	Terminates the subscription of data model changes by subscription ID.
`getAllPropertyValues(): Promise<any>;`	**Return value:** • Promise with all properties that are available in the component
`unsubscribePropertyValueChanges(sid: string): void;`	Terminates the subscription of value changes by subscription ID.
`setControlWidth(value: string): void;`	Sets the desired width of the control. **Possible values:** • CSS values for width/height (e.g. 150px, 33%, 73vw)
`setControlHeight(value: string): void;`	Sets the desired height of the control. **Possible values:** • CSS values for width/height (e.g. 150px, 33%, 73vw)
`getControlWidth(): Promise<any>;`	**Return value:** • Promise with the height of the control

Function	Description
`getControlHeight(): Promise<any>;`	**Return value:** • Promise with the width of the control
`getActions(): Promise<any>;`	**Return value:** • Promise with the action name
`executeAction(name: string, parameters?: any): Promise<any>;`	Accepts the action name and an optional custom parameter array of the `[ {key:'test', value:'test'}, ..... ]` type. **Return value:** • Promise with action response

15 Data Binding

Data Binding is a mechanism for synchronizing data between objects and UI elements to display data and affect the behavior of controls. Usually, data is passed from a data object (such as a customer list) to a control on the user interface (such as a table) and vice versa. If this mechanism is used, a loop does not have to be used for each object in the dataset, but the control identifies how many objects exist and adjusts itself automatically. If changes are made by the user in the control, they can be restored to the dataset.

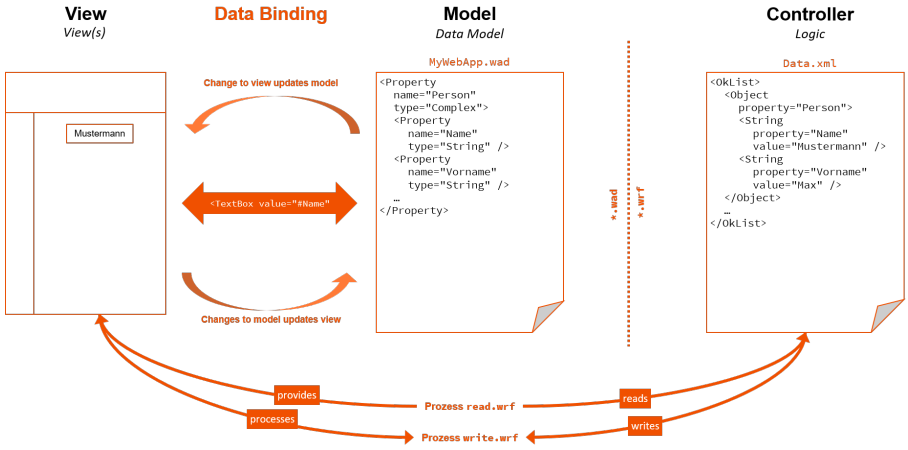
The name attributes of the elements are used for data binding.

Example Data Binding:

```
<DetailComponent name="DetailExample" path="DetailExample" displayName="Detail
example">
    <Properties>
        <Property name="Example" type="String" />
        <Property name="Example2" type="Complex">
            <Property name="Internal" type="String" />
            <Property name="Visible" type="Boolean" />
        </Property>
    </Properties>
    <FlowLayout>
        <Label value="#Example" />
        <Checkbox checked="#Example2.Visible" displayName="Visible" />
        <Label value="#Example2.Internal" visible="#Example2.Visible" />
    </FlowLayout>
</DetailComponent>
```

The above example shows the usage of data binding. The `value`attribute of the `Label` control allows data binding, so the name of the property can be used with a leading # character. The path mechanism is used to guarantee access to subproperties of `List` or `Complex` properties. A dot (`.`) is used as a separator between path segments.

One of the properties (`Example2.Visible`) is used to define if the checkbox is checked and as attribute `visible` of the second label. This makes it possible to control the visibility of the second label using the checkbox: If the value of the checkbox changes (selected or not selected), the data binding mechanism updates it at every point.



If a # character is to be used in a data binding expression, the character must be escaped. A backslash (\) is used as escaping character.
Example: `\#123456`

16 Navigation within a Web Application

Within a web application, there are various ways of navigation, that is, switching between different component. By default, the menu is created from the display names (`displayName`) of the components. Users can use this menu to access the corresponding components.

Other navigation options include are defining a custom action and overwriting the `select` action in the list component of a master structural element within a master/detail component. These two options can also be used to navigate to subcomponents within a master/detail component.

16.1 Navigation with an Action

The following examle shows how to define an action that navigates to the component with the `componentName` *name*.

```
<CustomAction displayName="Navigate" componentName="name" />
```

The action in this example navigates regardless of wether something is selected in the master/detail component. The `selectionNeeded` attribute can be used to bind the action to a selection:

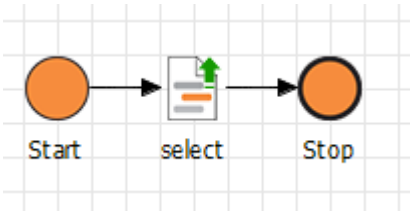
```
<CustomAction displayName="Navigate" componentName="name" selectionNeeded="true" />
```

16.2 Navigation by overwriting the `select` action

The following example shows a master/detail component with overwritten `select` action in the master structural element:

```
<MasterDetailComponent xmlns="http://softproject.de/webapp/1.0">
    <Properties>
        <Property name="List" type="Complex">
            <Property name="Value" type="String" />
        </Property>
    </Properties>
    <Master>
        <ListComponent name="list" process="fillListProcess.wrf">
            <Actions>
                <SelectAction process="select.wrf" /> <!-- Overwritten select action
-->
            </Actions>
        </ListComponent>
    </Master>
    <Detail>
        <DetailComponent name="Detail">
            <FlowLayout>
                <Label value="#List.Value" />
            </FlowLayout>
        </DetailComponent>
    </Detail>
</MasterDetailComponent>
```

select.wrf:



select.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Ok componentName="Detail2" />
```

In this example, the Detail2 component is always used as detail component for the selection in the master/detail component. The Technical Process that is bound to the overwritten `select` action returns a response containing the given `componentname` attribute. This gives the instruction to navigate to the corresponding component.

This example is very simple, it is possible to include complex mappings and conditions in the Technical Process that returns the response to select a detail for display in each individual case.

An example for a master/detail component with project structure can be found in the corresponding section.

The attributes of components, actions, properties and the data binding mechanism can also be found in the corresponding sections.

17 Sorting

Tables within a web application created with X4 Web Apps can be sorted by the values within a column.

In the example already used for Paging, some changes are made to be able to sort. First, the column in the List component must be marked as sortable (`sortable="true"`).

```
<ListComponent name="Sorting" path="Sorting" displayName="Paging" process="fillListPr
ocess.wrf" default="true">
    <Properties>
        <Property name="List" type="Complex">
            <Property name="Description" type="String" displayName="Description"/>
            <Property name="Id" type="Integer" displayName="Id"/>
        </Property>
     </Properties>
     <Columns>
        <Column value="#List.Id" sortable="true" />
        <Column value="#List.Description" />
     </Columns>
</ListComponent>
```

The column header now looks like this:

. The symbol means that the sorting mechanism is activated. However, clicking on the column header does not change anything yet, the list is only reloaded without sorting.

New requests are necessary for sorting:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Search offset="0" limit="50">
    <OrderBy>
        <Asc property="#List.Id" />
    </OrderBy>
    <Where />
</Search>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Search offset="0" limit="50">
    <OrderBy>
        <Desc property="#List.Id" />
    </OrderBy>
    <Where />
</Search>
```

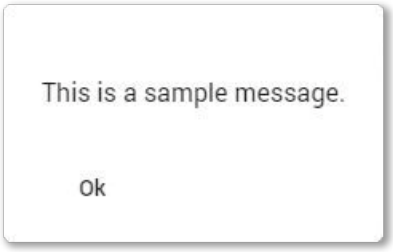
The `<Asc>` and `<Desc>` tags stand for *ascending* and *descending* sorting respectively. By using them, queries, mappings, etc. can be created to get data in a way that the sorting mechanism uses it, and thus support sorted lists.

18 Messages

Messages can be displayed within a web application that is created with X4 Web Apps. `Message` and `Error` elements can be used for this in the XML file that contains the response. These messages are displayed as dialog box at the corresponding position.

> The XML file that contains the message must be provided by a Technical Process. This Technical Process can be called with a custom action, for example.

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok>
    <Message>This is a sample message.</Message>
</Ok>
```

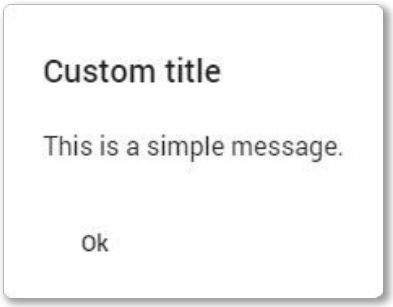


Messages can be localized by entering a language key as content of the message:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok>
    <Message>$translationKey</Message>
</Ok>
```

With the optional attribute title, a title can be assigned to the dialog box that is displayed in the web application:

```
<?xml version="1.0" encoding="UTF-8"?>
<Ok>
    <Message  title="Custom title">This is a simple message.</Message>
</Ok>
```



If the element <Error> is used instead of the element <Ok>, the message is output as an error message. For error messages, the title "Error" or the corresponding translation is displayed as the title. No user-defined title can be assigned for error messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
    <Message>This is a sample error.</Message>
</Error>
```

19 Profiling

In a profiling file (`.profiling`), you can define whether processes within your project should be logged at runtime. In the profiling file, you can define exactly which processes are included or excluded from logging.

The pofiling file only needs to be created if you want to exclude specific processes from profiling.

19.1 Create profiling file

⚠ Only one `.profiling` file can be created in a project.

1. In the **Projects** view, right-click a project.
2. Select **New > Profiling** in the context menu.
   The profiling configuration file is created and opened.

19.2 Configure profiling

The root element of the profiling file is `<Profiling>`. The element can contain individual processes of the project that should be explicitly covered or excluded from logging.

The `<Profiling>` element can have the following attribute:

Attribute	Description
`active`	Defines whether logging of processes is enabled or disabled. **Possible values:** • *`true`*: Logging of the processes is enabled. (default) • *`false`*: Logging of the processes is disabled.

The `<Profiling>` element contains the `<Processes>` child element, which may contain the following attribute:

Attribute	Description
`filterType`	Defines whether the stored processes should be included or excluded from logging. **Possible values:** • *`include`*: The stored processes are explicitly covered by the logging. • *`exclude`*: The stored processes are explicitly excluded from logging.

The <Processes> element may contain <Process> child elements if one or more processes are to be explicitly covered or excluded from logging.

The Process element contains the following attribute:

Attribute	Description
path	*Required*. Contains the path to the Technical Process to be captured or excluded. **Possible values:** Repository path to a Technical Process (e.g. Processes/Process.wrf)

19.3 Example of profiling configuration file

In the following example, the Technical Processes Process_B.wrf and Process_C.wrf are excluded from logging.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Profiling>
    <Processes filterType="exclude">
        <Process path="Processes/Process_B.wrf"/>
        <Process path="Processes/Process_C.wrf"/>
    </Processes>
</Profiling>
```