



Elektrobit

EB GUIDE Studio

Widget template library documentation

Version 6.5.1.137731



Elektrobit Automotive GmbH
Am Wolfsmantel 46
D-91058 Erlangen
GERMANY

Phone: +49 9131 7701-0
Fax: +49 9131 7701-6333
<http://www.elektrobit.com>

Legal notice

Confidential and proprietary information.

ALL RIGHTS RESERVED. No part of this publication may be copied in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Elektrobit Automotive GmbH.

ProOSEK®, tresos®, and street director® are registered trademarks of Elektrobit Automotive GmbH.

All brand names, trademarks and registered trademarks are property of their rightful owners and are used only for description.

Copyright 2018, Elektrobit Automotive GmbH.

Table of Contents

1. Introduction	5
1.1. System requirements	5
1.2. Target audience: Modelers	5
1.3. Background information	5
2. Touch buttons	6
2.1. TL_ButtonTouch	6
2.1.1. Using and customizing	6
2.1.2. Interface properties	7
2.2. Derived templates	7
2.2.1. TL_ButtonTouch_Label	7
2.2.1.1. Interface properties	8
2.2.2. TL_ButtonTouch_Icon	8
2.2.2.1. Interface properties	8
2.2.3. TL_ButtonTouch_IconLabel	8
2.2.3.1. Interface properties	9
3. Toggle buttons	10
3.1. Using and customizing	10
3.2. Interface properties	10
3.3. Derived templates	11
3.3.1. TL_ButtonToggle_Label	11
3.3.1.1. Interface properties	11
3.3.2. TL_ButtonToggle_Icon	11
3.3.2.1. Interface properties	12
4. Radio groups	13
4.1. Using and customizing	13
4.2. Interface properties	13
4.3. Derived templates	14
4.3.1. TL_ButtonGroup_Radio_Horizontal	14
4.3.1.1. Interface properties	14
4.3.2. TL_ButtonGroup_Radio_Vertical	14
4.3.2.1. Interface properties	15
5. Progress bar	16
5.1. Using and customizing	16
5.2. Interface properties	16
5.3. Derived templates	17
5.3.1. TL_Progressbar_Horizontal	17
5.3.1.1. Interface properties	17
5.3.2. TL_Progressbar_Vertical	17
5.3.2.1. Interface properties	18

5.3.3. TL_Progressbar_Radial	18
5.3.3.1. Interface properties	18
6. Sliders	20
6.1. Using and customizing	20
6.2. Interface properties	20
6.3. Derived templates	21
6.3.1. TL_Slider_Horizontal	21
6.3.1.1. Interface properties	21
6.3.2. TL_Slider_Vertical	21
6.3.2.1. Interface properties	22
7. Lists	23
7.1. Using and customizing	23
7.2. Derived templates	23
7.2.1. TL_List_Empty	23
7.2.1.1. Using and customizing	24
7.2.1.2. Interface properties	24
7.2.2. TL_List_CoastdownAnimation	25
7.2.2.1. Using and customizing	25
7.2.2.2. Interface properties	25
7.2.3. TL_List_HorizontalArch	26
7.2.3.1. Interface properties	26
8. Spellers	28
8.1. Using and customizing	28
8.2. Interface properties	28
8.3. Derived templates	29
8.3.1. TL_Speller_Number	29
8.3.2. Interface properties	29
8.3.3. TL_Speller_Alphanumeric	29
8.3.3.1. Using and customizing	30
8.3.3.2. Interface properties	30

1. Introduction

1.1. System requirements

To use the EB GUIDE Studio widget template library, EB GUIDE Studio Version 6.5.1.137731 has to be installed on your computer.

1.2. Target audience: Modelers

Modelers use EB GUIDE Studio to create a human machine interface (HMI). In EB GUIDE the HMI is called EB GUIDE model. Communication with applications is carried out through determined events using the event mechanism, through datapool items using the datapool and through user-specific EB GUIDE Script functions.

For working with the widget template library the modeler must have profound knowledge of EB GUIDE Studio features

1.3. Background information

Widget template

A widget template allows the definition of a customized widget that can be used multiple times in an EB GUIDE model. You can define templates on the basis of existing widgets or derive a new template from an existing one. After creating, you modify the template according to your needs, for example by adding properties or widget features.

Widget template interface

Every widget template has a template interface. The template interface contains the properties of the template which are visible and accessible in widget instances.

Widget template library

The library contains widget templates for the most common widgets used in an EB GUIDE project such as buttons, lists and sliders. You can use and customize these templates for modeling your EB GUIDE project.

For more information refer to the EB GUIDE Studio user manual.

2. Touch buttons

A touch button is a simple button which executes a configurable action based on touch input. It has a different appearance depending on whether it is being pressed or not.

The widget template library contains four templates for touch buttons.

Table 2.1. Touch button templates

Template name	Purpose
TL_ButtonTouch	Basic button template
TL_ButtonTouch_Label	Derived from the basic button template TL_ButtonTouch displaying text
TL_ButtonTouch_Icon	Derived from the basic button template TL_ButtonTouch displaying an image
TL_ButtonTouch_IconLabel	Derived from the basic button template TL_ButtonTouch displaying text and an image

2.1. TL_ButtonTouch

TL_ButtonTouch is the basic template for all touch buttons. All further touch button templates are derived from it and therefore contain the same interface properties.

TL_ButtonTouch contains an image widget that displays different images depending on the status of the widget feature property **Touched**.

- ▶ When the button is touched, the status of the **Touched** property is automatically set to true and the image defined as `pressedImage` is shown.
- ▶ When the button is being released, the status of the **Touched** property is automatically set to false, the `releasedReaction` is executed and the image defined as `releasedImage` is shown.



Figure 2.1. TL_ButtonTouch

2.1.1. Using and customizing

Since the button should have the same appearance throughout the project, you can only adapt the width of the touch button instance. The image height depends on the height of the image resource.

To customize the appearance of the button, change the image file of the image properties `pressedImage` and `releasedImage`.

2.1.2. Interface properties

Table 2.2. Template interface properties

Property	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>releasedReaction</code>	The action to be executed when the button is released
<code>lineIndex</code>	The position of every child of an instantiator
<code>listIndex</code>	Adds a list number as unique position to every child of an instantiator. This is only valid, when the widget feature List layout is used.

2.2. Derived templates

The following button templates are derived from the basic button template `TL_ButtonTouch` and have therefore the same basic template interface properties. For further information refer to [section 2.1, “TL_ButtonTouch”](#).

2.2.1. TL_ButtonTouch_Label

`TL_ButtonTouch_Label` is derived from `TL_ButtonTouch` and has an additional label widget to display text.



Figure 2.2. `TL_ButtonTouch_Label`

2.2.1.1. Interface properties

TL_ButtonTouch_Label has the following additional interface properties.

Table 2.3. Additional interface properties of TL_ButtonTouch_Label

Property	Description
text	The text displayed on the button
font	The font of the text displayed on the button

2.2.2. TL_ButtonTouch_Icon

TL_ButtonTouch_Icon is derived from TL_ButtonTouch and has an additional image widget to display an image.



Figure 2.3. TL_ButtonTouch_Icon

2.2.2.1. Interface properties

TL_ButtonTouch_Icon has the following additional interface properties.

Table 2.4. Additional interface properties of TL_ButtonTouch_Icon

Property	Description
icon	The icon displayed on the button

2.2.3. TL_ButtonTouch_IconLabel

Is derived from TL_ButtonTouch and has an additional image and label widget to display text and an image.



Figure 2.4. TL_ButtonTouch_IconLabel

2.2.3.1. Interface properties

`TL_ButtonTouch_IconLabel` has the following additional interface properties.

Table 2.5. Additional interface properties of `TL_ButtonTouch_IconLabel`

Property	Description
<code>image</code>	The image displayed on the button
<code>text</code>	The text displayed on the button

3. Toggle buttons

A toggle button is a simple touch button which allows to change a setting between two or more states.

The widget template library contains two templates for toggle buttons.

Table 3.1. Toggle button templates

Template name	Purpose
TL_ButtonToggle_Label	Toggle button based on the template TL_Button-Touch_Label and cycling through a list of labels.
TL_ButtonToggle_Icon	Toggle button based on the template TL_Button-Touch_Icon and cycling through a list of images.

3.1. Using and customizing

Every time a toggle button is pressed, the value of the property `currentState` increases and the corresponding image from the `imageList` is displayed. When the last button state is reached, the first state is activated again. Toggle reactions can be defined for each button state. Every time the button is pressed, the function `toggleReaction` is executed.

For example, when touching a `Play` button in a media player, a song is played and the button image changes from `Play` to `Pause`.

3.2. Interface properties

Table 3.2. Template interface properties

Property	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>toggleReaction</code>	The action to be executed on the release of the button

Property	Description
lineIndex	The position of every child of an instantiator
listIndex	Adds a list number as unique position to every child of an instantiator. This is only valid, when the widget feature List layout is used.
currentState	The current button state

3.3. Derived templates

3.3.1. TL_ButtonToggle_Label

TL_ButtonToggle_Label is derived from TL_ButtonTouch_Label. It additionally contains a list of texts for displaying different texts for each button state. For further information refer to [section 2.2.1, "TL_ButtonTouch_Label"](#).



Figure 3.1. TL_ButtonToggle_Label

3.3.1.1. Interface properties

TL_ButtonToggle_Label has the following additional interface properties.

Table 3.3. Additional interface properties of TL_ButtonToggle_Label

Property	Description
textList	List with texts for each button state
font	The font of the text displayed

3.3.2. TL_ButtonToggle_Icon

TL_ButtonToggle_Icon is derived from TL_ButtonTouch_Icon and additionally contains a list of images for the toggle reaction. For further information refer to [section 2.2.2, "TL_ButtonTouch_Icon"](#).



Figure 3.2. `TL_ButtonToggle_Icon`

3.3.2.1. Interface properties

`TL_ButtonToggle_Icon` has the following additional interface properties.

Table 3.4. Additional interface properties of `TL_ButtonToggle_Icon`

Property	Description
<code>imageList</code>	List with images that change on toggle reaction

4. Radio groups

A group of simple touch buttons where only one of the buttons can be selected at a time. The currently selected button is highlighted.

The widget template library contains two radio group templates.

Table 4.1. Radio group templates

Template name	Purpose
TL_ButtonGroup_Radio_Horizontal	Horizontally aligned group
TL_ButtonGroup_Radio_Vertical	Vertically aligned group

4.1. Using and customizing

Radio buttons are used if the user can select only one option in a list of various options.

To customize the appearance and highlighting, change the image widget in the `selectionOverlay` property.

4.2. Interface properties

Table 4.2. Template interface properties

Property	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>textList</code>	List of texts to be displayed on the buttons
<code>selectedIndex</code>	The index of the currently selected button. The index is automatically set by the system.
<code>selectedItemReaction</code>	The actions to be executed on the release of the button

4.3. Derived templates

4.3.1. TL_ButtonGroup_Radio_Horizontal

`TL_ButtonGroup_Radio_Horizontal` is an instantiator widget, which instantiates the instances of `TL_ButtonTouch_Label`. The width of the buttons is automatically adapted and depends on how many buttons are to be displayed.



Figure 4.1. `TL_ButtonGroup_Radio_Horizontal`

4.3.1.1. Interface properties

`TL_ButtonGroup_Radio_Horizontal` has the same properties as `TL_ButtonTouch_Label`. For further information refer to [section 2.2.1, “TL_ButtonTouch_Label”](#).

4.3.2. TL_ButtonGroup_Radio_Vertical

`TL_ButtonGroup_Radio_Vertical` is an instantiator widget, which instantiates the instances of `TL_ButtonTouch_IconLabel`.

Each button has the same width as `TL_ButtonGroup_Radio_Vertical`. The height of the group is defined by the amount of buttons and the vertical gap.

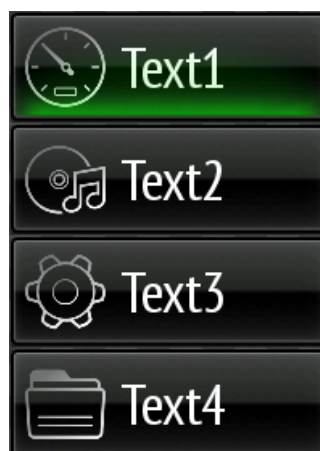


Figure 4.2. `TL_ButtonGroup_Radio_Vertical`



4.3.2.1. Interface properties

`TL_ButtonGroup_Radio_Vertical` has the following additional interface properties.

Table 4.3. Additional interface properties of `TL_ButtonGroup_Radio_Vertical`

Property	Description
<code>iconList</code>	List of icons to be displayed on the buttons
<code>verticalGap</code>	The vertical space between two buttons

5. Progress bar

A progress bar visualizes the progress of a status in percent. It can be used to show a continuous progress, as for example the fill state of a battery or a loading state.

The progress bar templates contain two image widgets: `BackgroundImage` and `FillImage`. The width or height of the `FillImage` depend on the progress and is expressed by the widget property `progressPercentage`.

The widget template library contains three progress bar templates.

Table 5.1. Progress bar templates

Template name	Purpose
<code>TL_Progressbar_Horizontal</code>	Horizontally aligned progress bar
<code>TL_Progressbar_Vertical</code>	Vertically aligned progress bar
<code>TL_Progressbar_Radial</code>	A progress bar arranged as a circle

5.1. Using and customizing

To customize the appearance of the progress bar, change the resources of the image widgets `BackgroundImage` and `FillImage`.

5.2. Interface properties

Table 5.2. Template interface properties

Property	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>labelVisible</code>	If true, the progress percent value is shown.
<code>progressPercentage</code>	Indicates the current status represented as a percentage

5.3. Derived templates

5.3.1. TL_Progressbar_Horizontal

The template is a container widget.



Figure 5.1. TL_Progressbar_Horizontal

5.3.1.1. Interface properties

TL_Progressbar_Horizontal has the following additional interface properties:

Table 5.3. Additional interface properties of TL_Progressbar_Horizontal

Property	Description
bgColor	The background color of the progress bar image
fillColor	The fill color of the progress bar image

5.3.2. TL_Progressbar_Vertical

The template is a container widget.



Figure 5.2. TL_Progressbar_Vertical

5.3.2.1. Interface properties

TL_Progressbar_Vertical has the following additional interface properties:

Table 5.4. Additional interface properties of TL_Progressbar_Vertical

Property	Description
bgColor	The background color of the progress bar image
fillColor	The fill color of the progress bar image
startFromBottom	If true, the fill image is top aligned. Else it is bottom aligned.

5.3.3. TL_Progressbar_Radial

The template TL_Progressbar_Radial is a container widget.

It consists of two image widgets `Background` and `Indicator` and a set of 30 image widgets instantiated by an instantiator widget. The 30 image widgets are positioned in a circle and their visibility depends on the value of the widget property `progressPercentage` and their position in the circle. The lucency of the images is calculated depending on the angle in relation to the starting angle `startPositionAngle`. The position of the image widget indicator is calculated as the percent, defined by the widget property `ProgressPercentage`, of the degrees of a full circle.

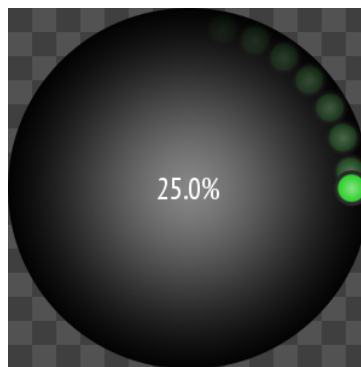


Figure 5.3. TL_Progressbar_Radial

5.3.3.1. Interface properties

Table 5.5. Additional interface properties of TL_Progressbar_Radial

Property	Description
fillImage	The image showing the progress indicator



Property	Description
startPositionAngle	The start angle of the first indicator (progress = 0%). Angles are calculated starting at the center top and going clockwise.
bgImage	The background image

6. Sliders

A slider visualizes the progress of a status. In contrast to a progress bar, the progress value can be changed by touch input. A slider knob can be moved to change the progress value. Touching the slider background moves the slider knob to the position where the background was touched.

The widget template library contains two slider templates.

Table 6.1. Slider templates

Template name	Purpose
TL_Slider_Horizontal	Horizontally aligned slider using different colors to the left and right of the slider knob to indicate a continuous progress.
TL_Slider_Vertical	Vertically aligned slider. It can be used as a scroll bar in a list.

6.1. Using and customizing

To customize the appearance of the slider, change the resources of the image widget `BG`, the properties `Left`, `Right`, and the `FillImage`.

The slider uses the widget feature **touchMoved** to move the slider knob along with the touch movement of the finger. The widget container uses the widget feature **touchPressed** to place the widget slider at the touched position.

6.2. Interface properties

Table 6.2. Template interface properties

Property	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget

Property	Description
sliderReaction	The reaction when the slider is moved

6.3. Derived templates

6.3.1. TL_Slider_Horizontal

The template `TL_Slider_Horizontal` is a container widget, containing three additional image widgets: `Left`, `Right`, and `Slider`.

The position of the slider knob is defined by the value of the widget property `sliderPercentage` based on the width of the slider. The widget `Left` fills the space to the left of the slider knob and the widget `Right` fills the space to the right.



Figure 6.1. `TL_Slider_Horizontal`

6.3.1.1. Interface properties

`TL_Slider_Horizontal` has the following additional interface properties.

Table 6.3. Additional interface properties of `TL_Slider_Horizontal`

Property	Description
sliderPercentage	The position of the slider in percent

6.3.2. TL_Slider_Vertical

The template `TL_Slider_Vertical` is an image widget. It contains a second image widget for the slider knob. The height of the slider knob is calculated as the relation of the widget property `height` to the widget property `totalListHeight`.



Figure 6.2. `TL_Slider_Vertical`

6.3.2.1. Interface properties

`TL_Slider_Vertical` has the following additional interface properties.

Table 6.4. Additional interface properties of `TL_Slider_Vertical`

Property	Description
<code>height</code>	The height of the widget in pixels
<code>bgImage</code>	The background image of the slider
<code>scrollValueMax</code>	The maximum scroll value
<code>scrollValue</code>	The current scroll value position of the slider
<code>totalListHeight</code>	The total height of the list

7. Lists

A list is a group of items. If there are items that cannot be displayed in the visible area of the list, the user can scroll to the not visible item through touch move.

The widget template library contains three list templates.

Table 7.1. List templates

Template name	Description
<code>TL_List_Empty</code>	An empty list with touch move handling
<code>TL_List_CoastdownAnimation</code>	<p>An animation which continues the scrolling of the list for a short period of time after the last touch input was received.</p> <p>This makes the scrolling behavior of the list appear smoother as the scrolling does not break off directly when the touch is released.</p>
<code>TL_List_HorizontalArch</code>	<p>Horizontally aligned list.</p> <p>The list consist of a vertically aligned image and text widget. It can be scrolled by touch and uses the <code>TL_List_CoastdownAnimation</code> widget.</p>

7.1. Using and customizing

The list templates can be customized as described in the following chapters.

7.2. Derived templates

7.2.1. `TL_List_Empty`

The template `TL_List_Empty` is an instantiator widget representing a list. The Instantiator widget is enhanced with the widget feature properties **List layout** and **View port**. Additionally the instantiator widget handles the touch move behavior of the list.

7.2.1.1. Using and customizing

Attach the list elements as children to the `TL_List_Empty` and set the values for the list size and layout.

To make the move handling smoother the template can be combined with `TL_List_CoastdownAnimation` which adds a coast down animation to the list so the list does not stop abruptly but continues scrolling for a while.

In order to add a scrollbar to the list, add the template `TL_Slider_Vertical` and link the properties `scrollValueMax` and `scrollValue`. The property `totalListHeight` of the `TL_Slider_Vertical` depends on the size of the list.

7.2.1.2. Interface properties

Table 7.2. Template interface properties

Property	Description
<code>visible</code>	If true, the widget and its child widgets are visible
<code>width</code>	The width of the widget in pixels
<code>height</code>	The height of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>numItems</code>	The number of instantiated child elements
<code>listLength</code>	The number of list elements
<code>layoutDirection</code>	The direction in which the list elements i.e. the child widgets are positioned
<code>scrollValue</code>	The current scroll value
<code>scrollValueMax</code>	The maximum scroll value, which is mapped to the end of the list
<code>wrapAround</code>	Possible values: <ul style="list-style-type: none">▶ <code>true</code>: If <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded, the <code>scrollValue</code> property continues at the inverse border.▶ <code>false</code>: If <code>scrollValueMin</code> or <code>scrollValueMax</code> is exceeded, the <code>scrollValue</code> property does not decrease or increase.

Property	Description
<code>lineMapping</code>	Defines which child of the instantiator is the template for which line.
<code>touched</code>	Set to true when the widget is touched
<code>scrollOffset</code>	The amount of pixels to scroll the list
<code>touchMoveDistance</code>	The distance in pixels, the list was shifted since the last touch move.
<code>scrollOffsetRebase</code>	If the <code>scrollOffsetRebase</code> property changes, the current <code>scrollOffset</code> is automatically translated to <code>scrollIndex</code> . The remaining offset is written to the <code>scrollOffset</code> property.

7.2.2. TL_List_CoastdownAnimation

The template `TL_List_CoastdownAnimation` is a container widget with two animation widgets: `Timer` and `Coastdown`.

The widget `Timer` keeps track of the time for how long the list is being touched. This time is used to calculate the distance (`moveSpeed`) for the scrolling to be continued. The widget `Coastdown` then uses a fast start integer animation widget to animate the scrolling behavior of the list.

7.2.2.1. Using and customizing

With `TL_List_CoastdownAnimation` a coast down animation is added to a list, so the list does not stop abruptly but continues scrolling for a while.

7.2.2.2. Interface properties

Table 7.3. Interface properties of `TL_List_CoastdownAnimation`

Property	Description
<code>listScrollOffset</code>	The amount of pixels to scroll the list
<code>listTouched</code>	If true, the list is currently touched
<code>listTouchMoveDistance</code>	The distance in pixels, the list was shifted since the last touch move.
<code>cancelAnimation</code>	If true, the animation is canceled

7.2.3. TL_List_HorizontalArch

The template `TL_List_HorizontalArch` is a container widget containing the templates `TL_List_Empty` and `TL_List_CoastdownAnimation`.

The list item for the `TL_List_Empty` widget is the container widget `Item`. It has the same height as the template widget. The widget `Item` contains the container widget `InnerItem` consisting of the image widget `Icon` and the label widget `Label`. The widget `InnerItem` is positioned within the widget `Item` depending on its place in the arch.

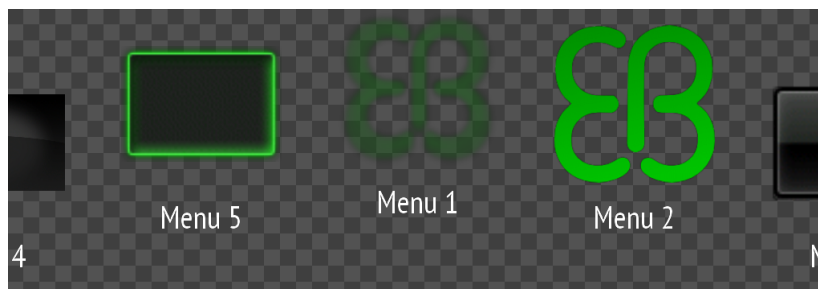


Figure 7.1. `TL_List_HorizontalArch`

7.2.3.1. Interface properties

Table 7.4. Interface properties of `TL_List_HorizontalArch`

Property	Description
<code>width</code>	The width of the widget in pixels
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>iconList</code>	The list of images to be displayed
<code>stringList</code>	The list of texts to be displayed
<code>selectedIndex</code>	The index of the currently selected button
<code>selectedItemReaction</code>	The actions to be executed on the release of the button
<code>initiallyCenteredItem</code>	The index of the item which is to be centered on initialization
<code>listItem_width</code>	The width of the list item in pixels
<code>listItem_gap</code>	The gap between the list items in pixels



Property	Description
<code>arc_height</code>	The height of the arc of the list in pixels
<code>font</code>	The text font of the list elements

8. Spellers

A speller is a group of buttons to be used as a touch input device for characters and numbers.

The widget template library contains two speller templates.

Table 8.1. Speller templates

Template name	Description
<code>TL_Speller_Number</code>	A speller arranged as a keypad to be used for number input.
<code>TL_Speller_Alphanumeric</code>	A speller arranged as a keyboard to be used for character input. It has a button to toggle between Latin and Arabic characters and a button to switch to numbers. The shift button can be used to shift to special characters.

8.1. Using and customizing

To customize the appearance, change the resource of the internal image widget `BG`, respectively `Background`. The width of the buttons can be changed by setting the internal widget property `ButtonWidth`.

In the `selectedCharacter` function define how the character input is handled.

8.2. Interface properties

Table 8.2. Template interface properties

Property	Description
<code>visible</code>	If set to true, the widget and its child widgets are visible.
<code>x</code>	The x-coordinate of the widget relative to its parent widget
<code>y</code>	The y-coordinate of the widget relative to its parent widget
<code>selectedCharacterReaction</code>	The reaction when one of the buttons is pressed
<code>selectedCharacter</code>	The character of the button that is pressed

8.3. Derived templates

8.3.1. TL_Speller_Number

The template `TL_Speller_Number` is a container widget. It contains nine `TL_Button_Touch` widgets which are arranged in a 3x3 grid using an instantiator widget and the widget feature **Grid layout**.

Each button is enhanced with two label widgets. The label widget `FirstLine` shows the number and the label widget `SecondLine` shows the characters.



Figure 8.1. `TL_Speller_Number`

8.3.2. Interface properties

The template `TL_Speller_Number` has no additional interface properties.

8.3.3. TL_Speller_Alphanumeric

The template `TL_Speller_Alphanumeric` is a container widget and can have two layouts: one for Latin and one for Arabic characters.

Each layout consists of a container widget with three rows. Each row is represented by an instantiator widget, instantiating `TL_ButtonTouch_Label` widgets. There are three `TL_ButtonToggle_Label` widgets to toggle the different layout configurations: `SpecialSymbols`, `ShiftKey`, and `AlphaNumeric`. Depending on the

button widget properties of `currentState`, `SwitchLanguage` and `Alphanumeric`, one of the two layouts is shown. For each layout there is a list of characters for the buttons texts. The text of the instantiated buttons depends on the state of all three widgets: `SpecialSymbols`, `ShiftKey`, and `AlphaNumeric`.



Figure 8.2. `TL_Speller_Alphanumeric`

8.3.3.1. Using and customizing

In the `selectedCharacterReaction` define how character input is handled.

In the `deletedCharacterReaction` define how the backspace input is handled.

8.3.3.2. Interface properties

Table 8.3. Additional template interface properties of `TL_Speller_Alphanumeric`

Template name	Description
<code>deletedCharacterReaction</code>	The reaction when the backspace button is pressed