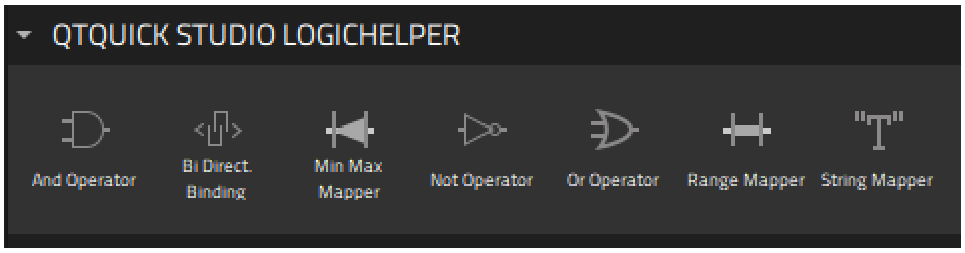
逻辑助手

若要让 UI 执行某些操作，可能需要为条件编写 JavaScript 表达式或将数字转换为字符串。为了简化此操作，Qt 设计工作室提供了一组称为*逻辑帮助器*的组件。



逻辑帮助程序可用于使用布尔 AND、NOT 和 OR 运算符绑定属性值，以及用于映射数字和数值范围。此外，还可以双向同步两个组件的属性值。

逻辑帮助程序是不可见的组件，可用于与控件（如滑块或复选框）结合使用。要使用逻辑帮助程序，请将其从**组件**> Qt **快速工作室逻辑帮助程序**拖放到导航器。如果在**组件**中找不到逻辑帮助程序，则需要将 **Qt Quick Studio 逻辑帮助程序**模块添加到项目中，如添加和删除模块中所述。

以下各节更详细地介绍了不同类型的逻辑帮助程序。

布尔助手

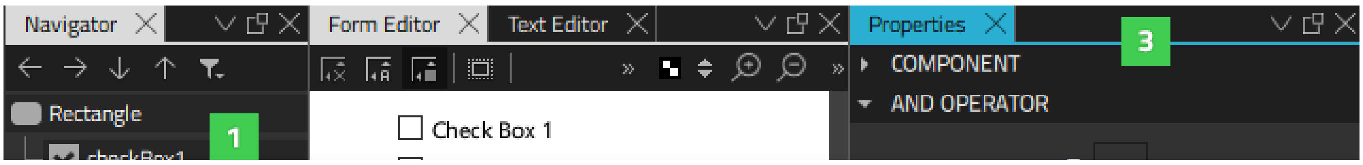
可以使用逻辑帮助程序通过布尔 AND、OR 和 NOT 运算符绑定属性值。

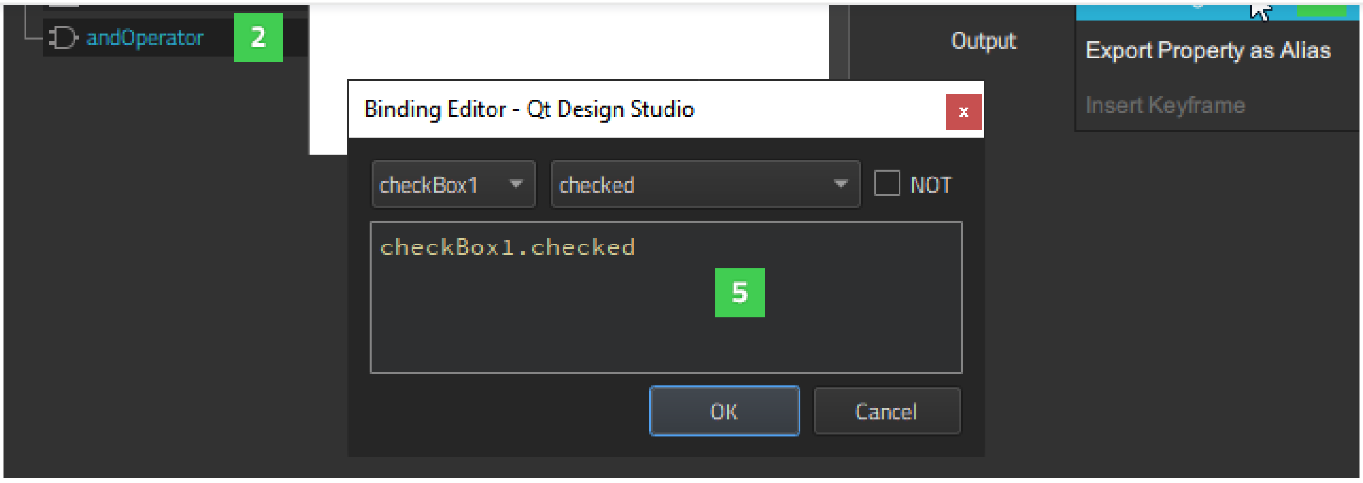
和运算符

And 运算符组件计算两个布尔输入。输出的计算方式就好像两个输入都是 一样。`truetrue`

例如，我们可以使用两个复选框的选中状态来确定第三个复选框的选中状态。首先，我们将**复选框**组件的三个实例和 And **运算符**组件的一个实例拖放到 **Navigator** （1）。然后，我们选择 **And 运算符**组件实例 （2） 并在属性 （3） 中设置其属性。

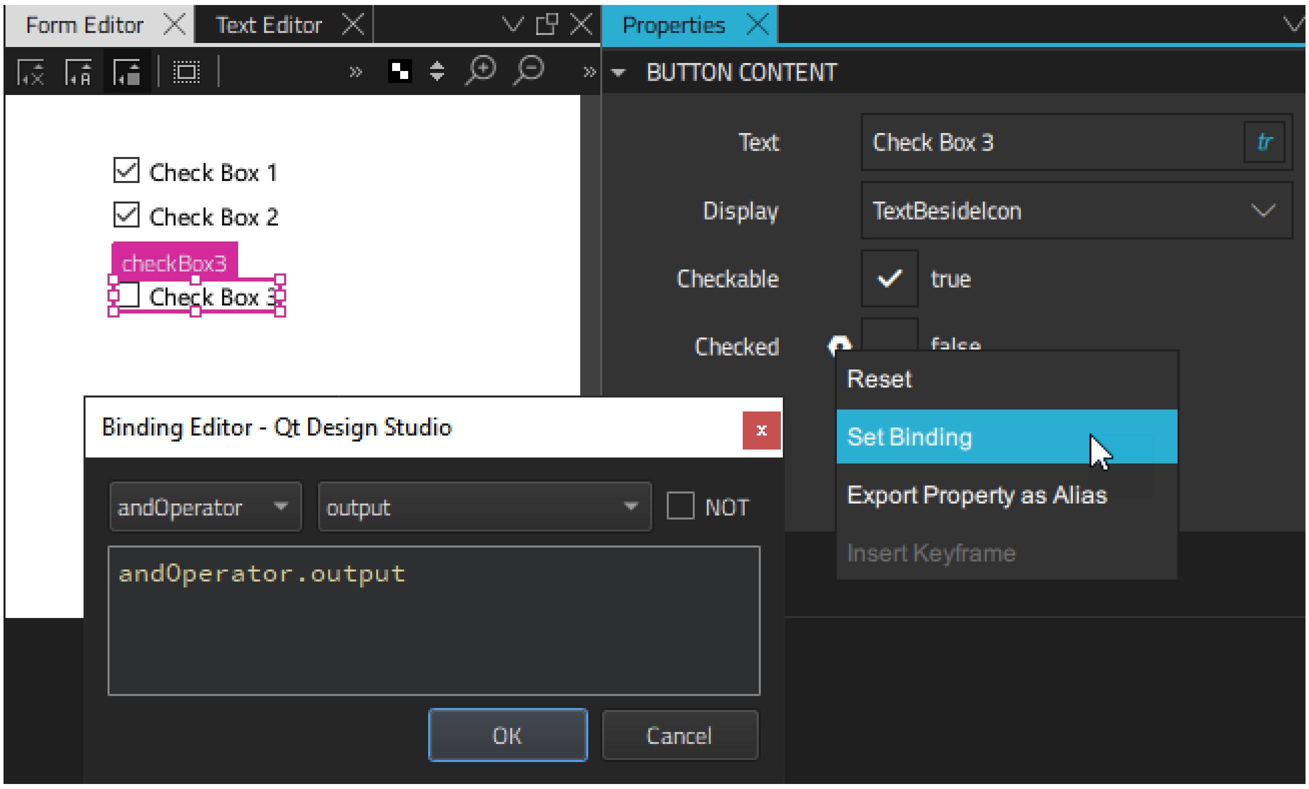
我们在“**输入 01**”字段旁边选择以显示“**操作**”菜单，然后选择“**设置绑定**”（4） 以打开**绑定编辑器** （5）。在那里，我们将 AND 运算符的属性值绑定到第一个复选框的属性值。然后，我们在**输入 02** 字段中执行相同的操作，在该字段中，我们将属性绑定到第二个复选框的属性。`input01checkedinput02checked`





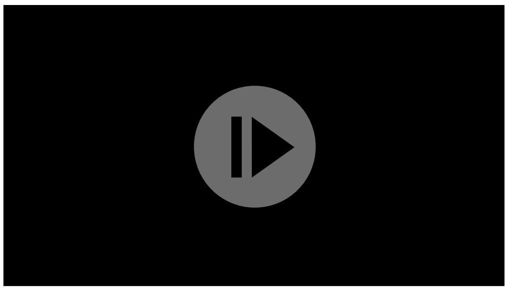
```
checkBox1.checked
```

最后，我们选中第三个复选框，并将其 **Checked** 属性绑定到 AND 运算符的“**输出**”属性。



```
andOperator.output
```

When we preview our UI, all the check boxes are initially unchecked. However, when we select the first and second check box, the third one also becomes checked.

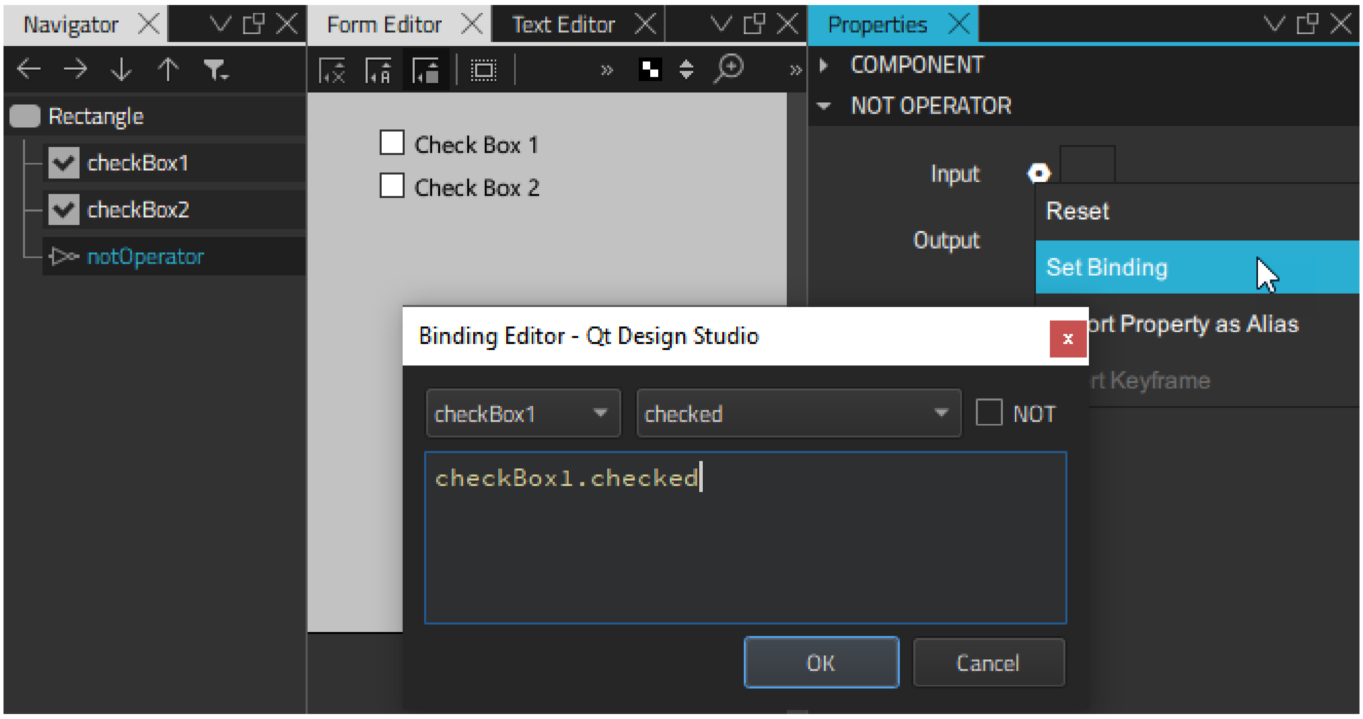


OR Operator

The **Or Operator** component does the same as the AND operator, but the output is if one or both inputs are `.truetrue`

The **Not Operator** component is evaluated to if the condition is not met.`true`

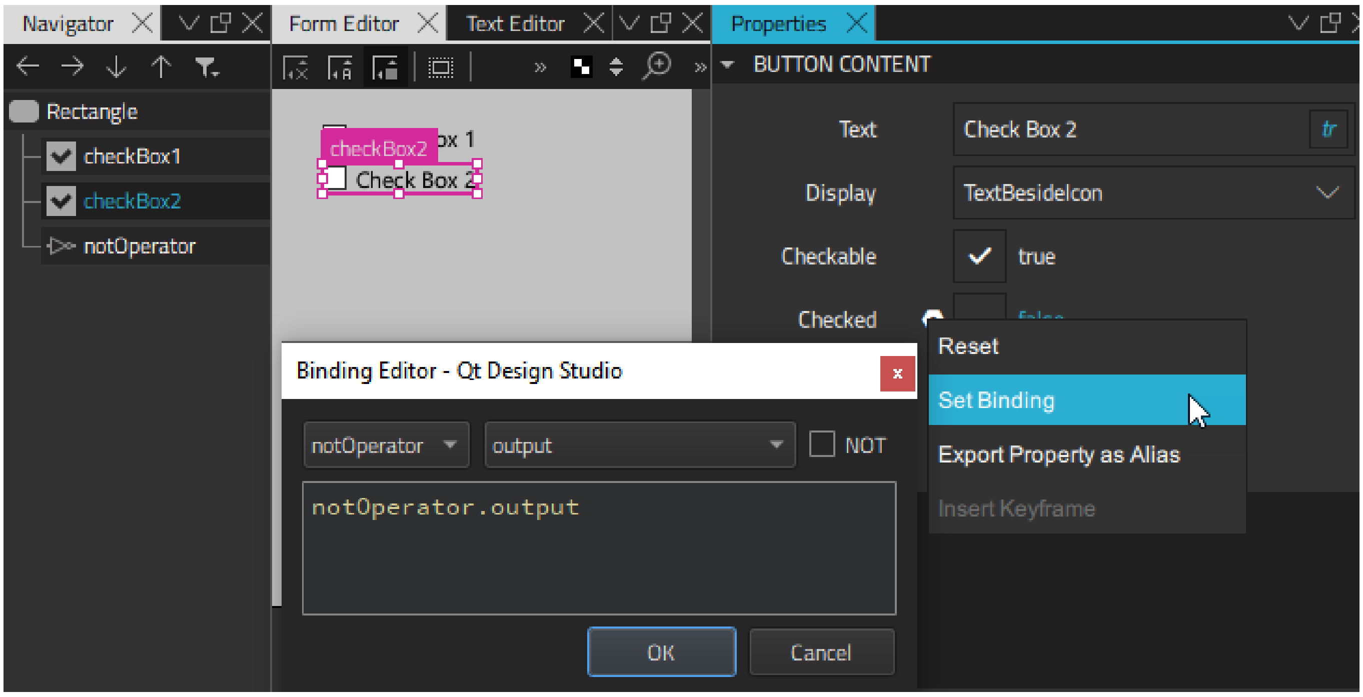
For example, we could specify that if one check box is checked, another one cannot be checked. First, we drag-and-drop two instances of the **Check Box** component and one instance of the **Not Operator** component to **Navigator**. Then, we select the **Not Operator** component instance and set its properties in **Properties**. In the **Binding Editor**, we bind the value of the property of the NOT operator to the value of the property of one check box instance.`input` `checked`



checkBox1 | checked | NOT

```
checkBox1.checked
```

We then select the other check box instance and bind the value of its **Checked** field to the value of of **Output** field of the **Not Operator** component.



Text	Check Box 2
Display	TextBesideIcon
Checkable	✓ true
Checked	false

notOperator | output | NOT

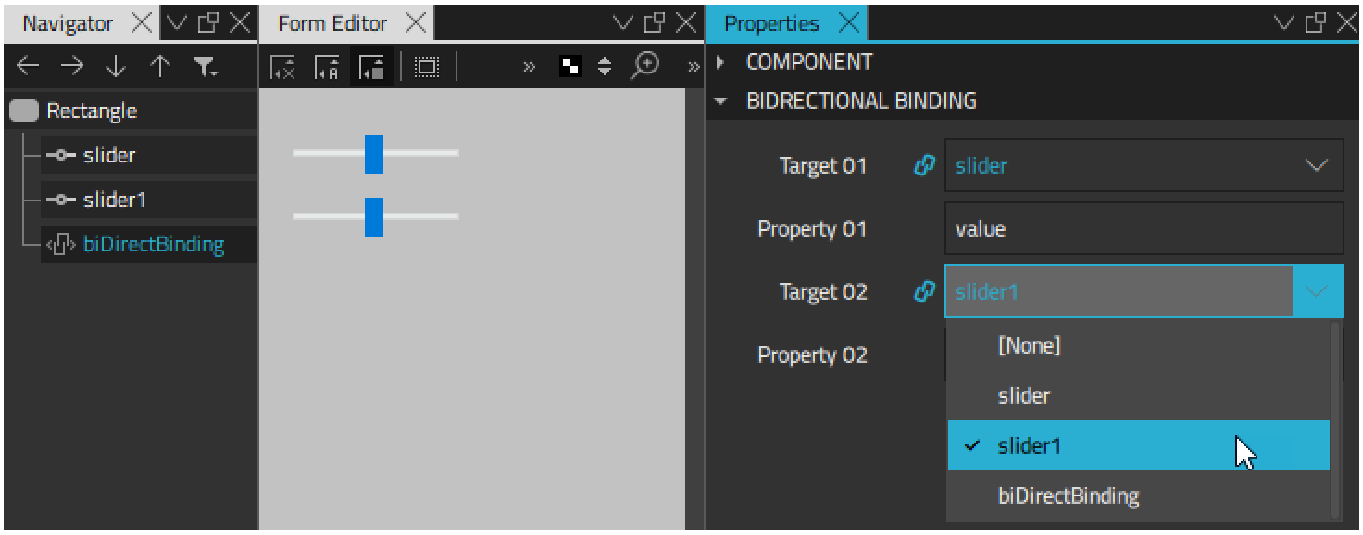
```
notOperator.output
```

When we preview our UI, the second check box is initially checked. However, when we select the first check box, the second one is automatically cleared.

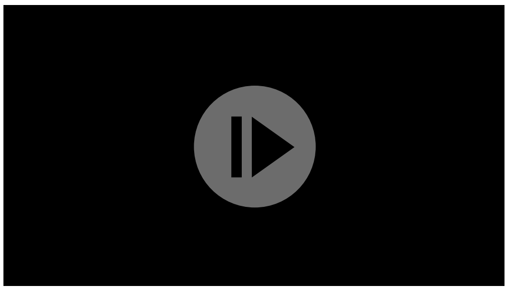
Bi-directional Binding

The **Bi Direct. Binding** component binds the values of two controls together, so that when one value is changed, the other one follows it. This component could be used to synchronize two sliders or a slider and checkbox. Typically, it is used to bind a backend value to a control, such as a slider.

For example, to synchronize the values of two sliders, we drag-and-drop two instances of the **Slider** component and one instance of the **Bi Direct. Binding** component to the same parent component in **Navigator**. Then, we select the bi-directional binding instance and set its properties in **Properties**.



In the **Target 01** and **Target 02** fields, enter the IDs of the components that you want to bind together. In the **Property 01** and **Property 02** fields, enter the names of the properties to synchronize. In our example, we bind the property of two slider components together, so that when we move one slider handle in the preview, the other one moves along with it.`value`

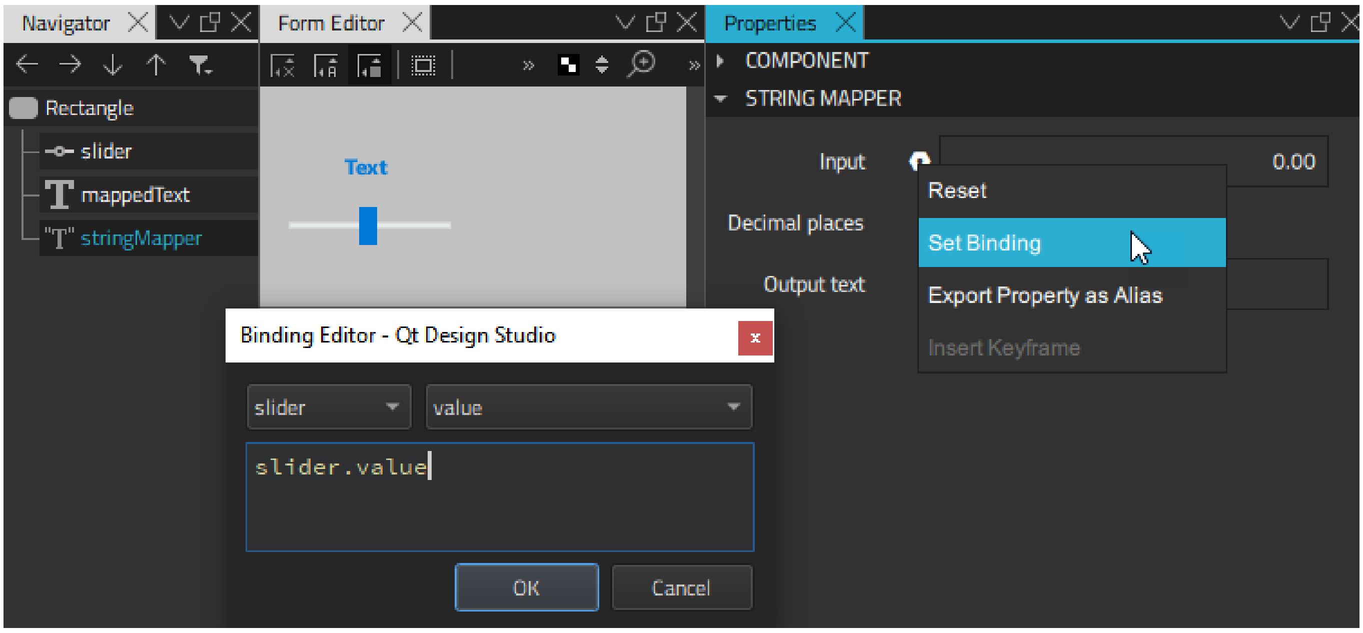


If you want to add a text field that displays the value of the slider, you can use a String Mapper component.

String Mapper

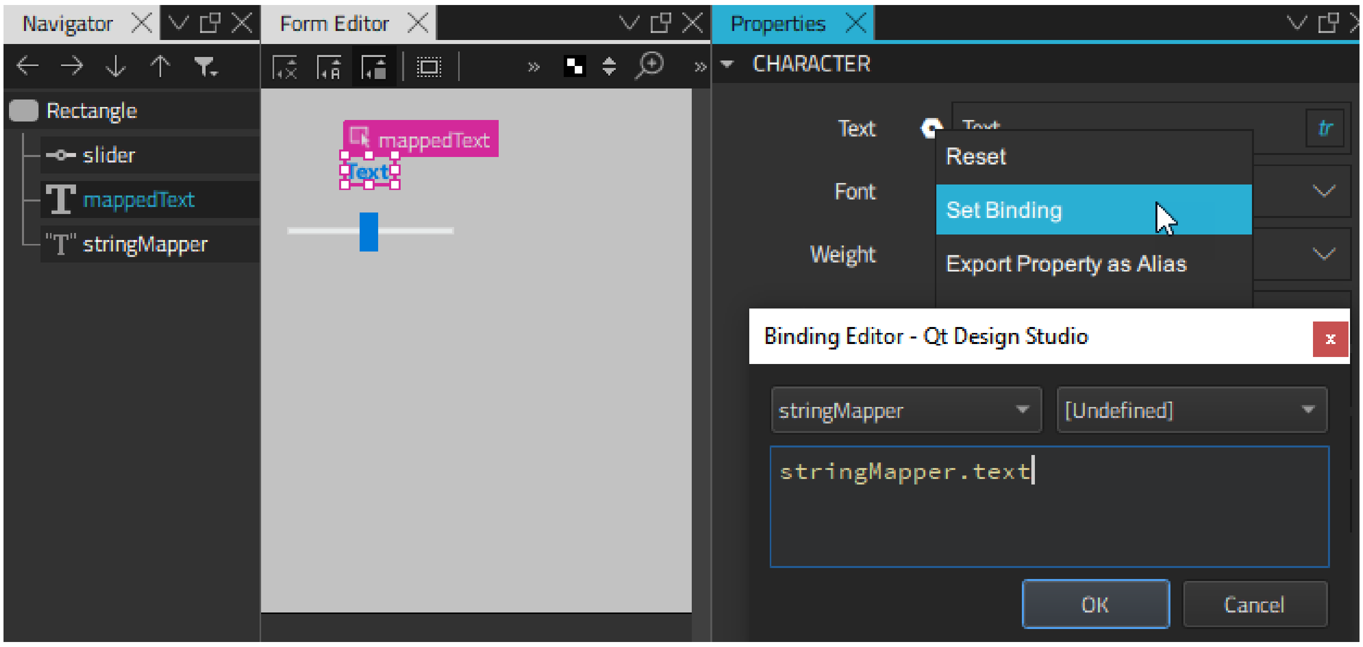
The **String Mapper** component maps numbers to strings. First, you add a binding between the string mapper property and the property of the control that you want to fetch the values from. Then, you add a binding between the property of the string mapper and that of the component that will display the string.`inputvaluetext`

For example, to use a Text component to display the value of a slider, we drag-and-drop **Text**, **Slider**, and **String**



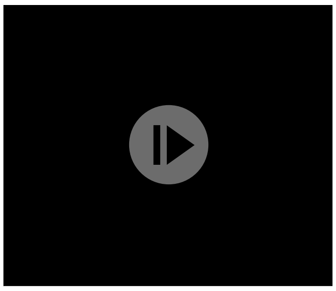
```
slider.value
```

Next, we select the **Text** instance and bind the value of the **Text** field to the value of the **Output text** field (property) of the **String Mapper** component.`text`



```
stringMapper.text
```

When we move the slider handle in the preview, the value displayed in the text component changes accordingly.

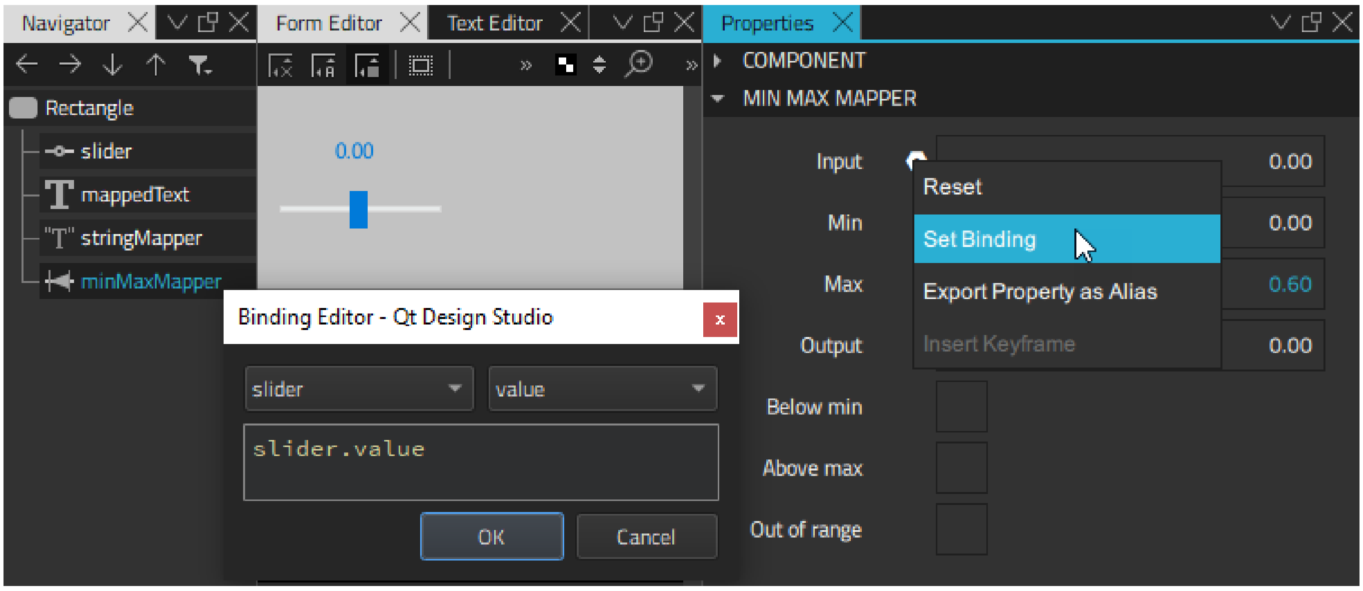


The value of the **Decimal places** field determines the number of digits after the decimal separator.

apply actions to values, such as changing a color in a state, even if they are below the minimum value or above the maximum value.

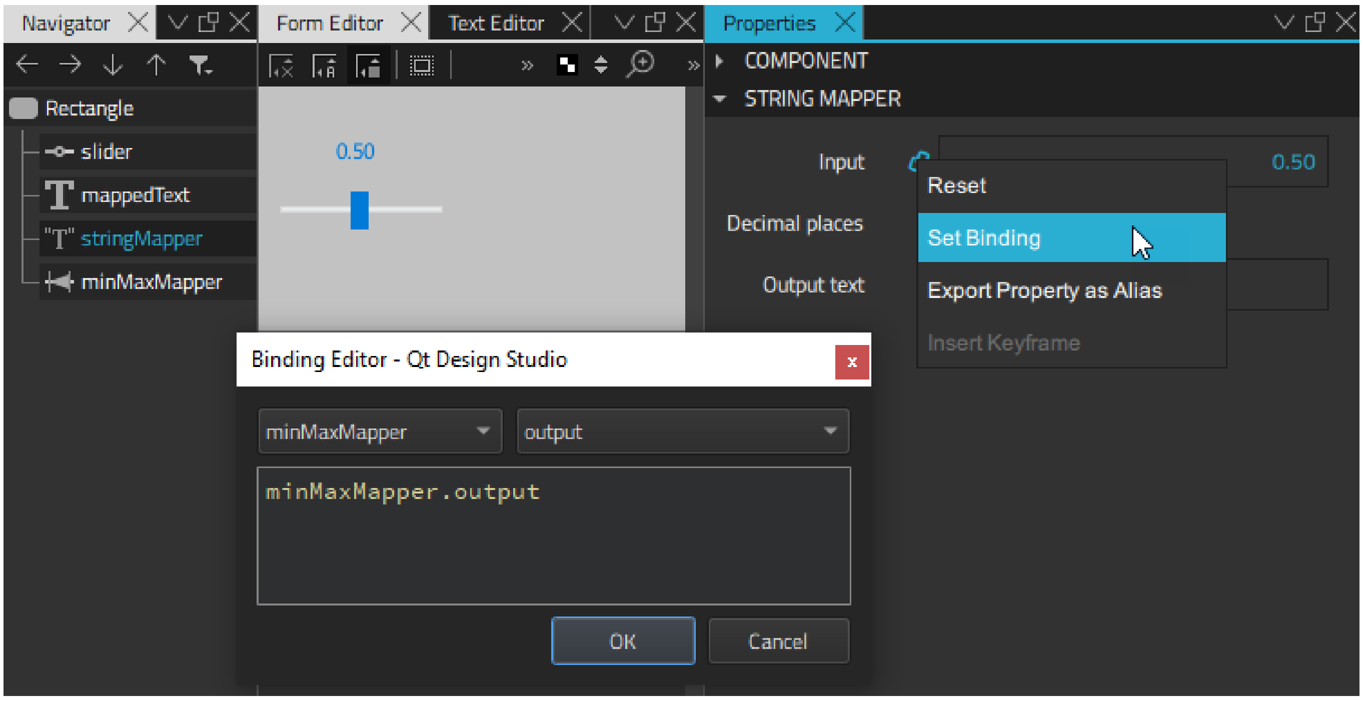
To access the values of a control, bind the **Input** property of the minimum-maximum mapper to that of the property of the control.`value`

For example, to restrict the maximum value of a slider to 0.60, regardless of the maximum value set in the slider properties, we drag-and-drop a **Min Max Mapper** to our example above. We select it to display its properties in **Properties**. Then, we bind the value of the **Input** property of the mapper to the value of the property of the slider and set the value of the **Max** field to 0.60.`value`



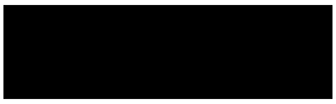
```
slider.value
```

To have the maximum value displayed by the Text component, we select the **String Mapper** component and change the binding we set as the value of the **Input** field from to `.slider.valueminMaxMapper.output`



```
minMaxMapper.output
```

When we move the slider handle in the preview, it only moves up to the value 0.60.

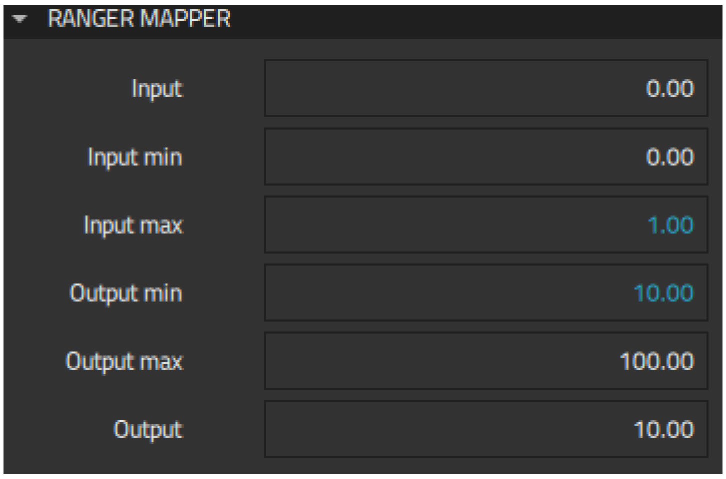


The **Out of range**, **Above max** and **Below min** check boxes are set to if the value of the **Input** field is out of range.`true`

For example, in the context of speed, **Above max** being would mean *too fast*, whereas **Below min** being , would mean *too slow*, and **Out of range** being would mean *either too fast or too slow*.`truetruetrue`

Range Mapper

The **Range Mapper** component maps a numerical range to another range, so that the output value of the second range follows the input value of the original range. This is useful when remapping the current frame on the timeline, for example.

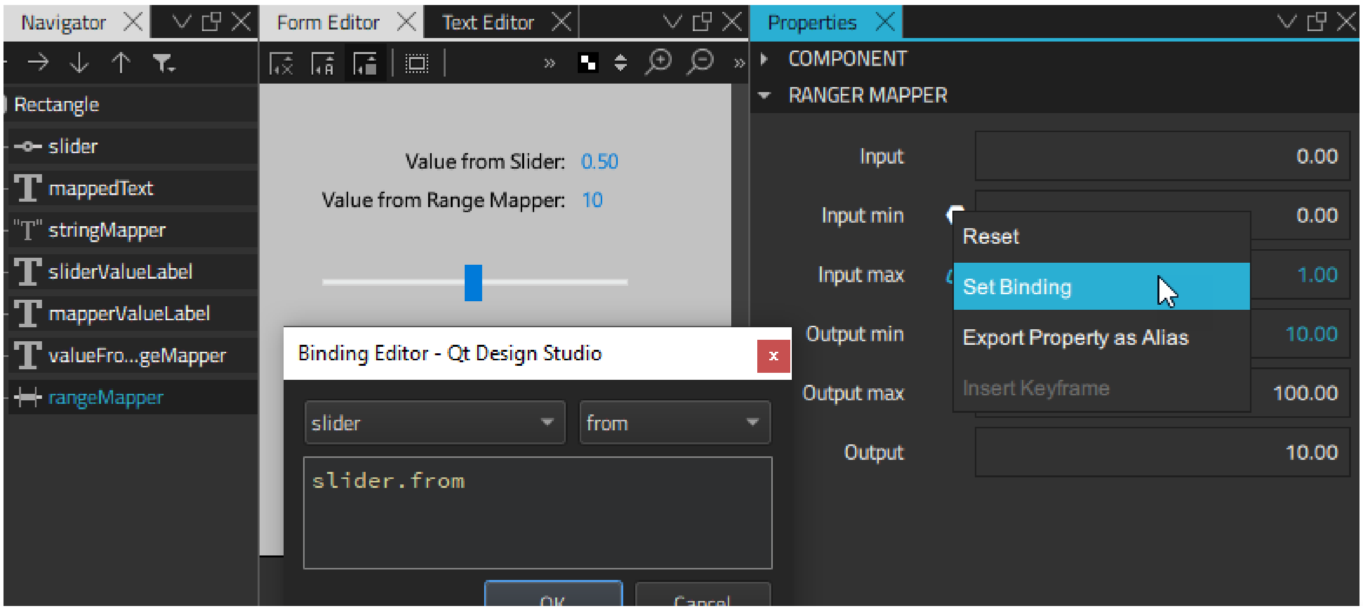


RANGER MAPPER

Input	0.00
Input min	0.00
Input max	1.00
Output min	10.00
Output max	100.00
Output	10.00

Specify the minimum and maximum input and output values in the **Input min**, **Input max**, **Output min**, and **Output max** fields and the original value in the **Output** field.

For example, we can specify that the values of a slider range from 0 to 1. If we want to display values from 10 to 100, instead, we can bind the values of the **From** and **To** fields of the Slider component to the values of the **Input min** and **Input max** fields of a **Range Mapper** component. We then set the value of the **Output min** field to 10 and the value of the **Output max** field to 100.



Value from Slider: 0.50
Value from Range Mapper: 10

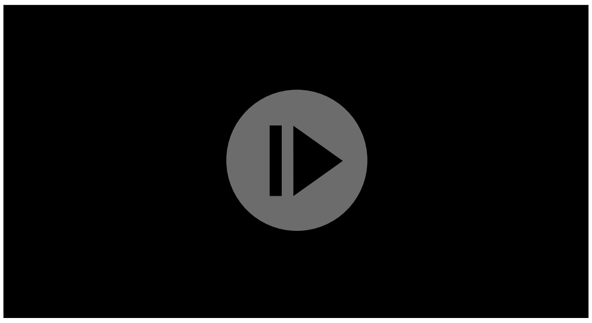
Binding Editor - Qt Design Studio

slider | from

```
slider.from
```

Input	0.00
Input min	0.00
Input max	1.00
Output min	10.00
Output max	100.00
Output	10.00

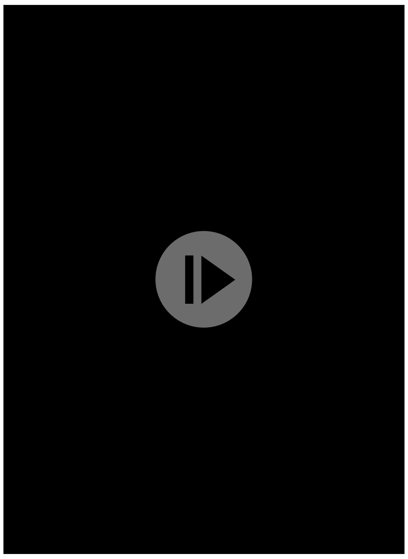
to 1, the output value changes from 10 to 100.



Combining Several Logic Helpers

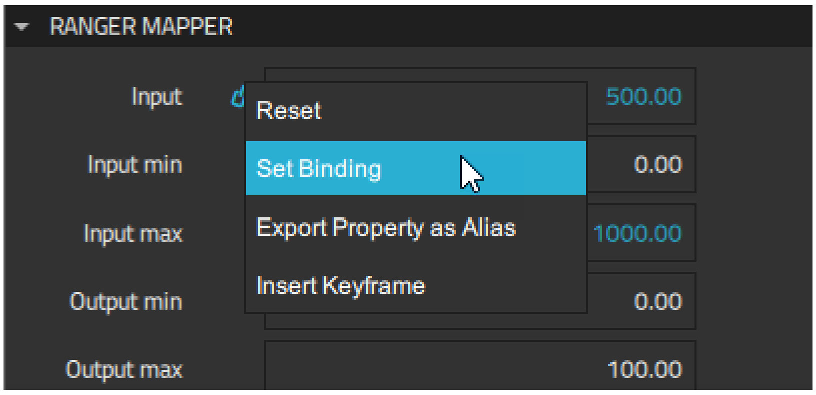
You can combine several logic helpers of the same type or different types to define the application logic.

For example, we create a small application for selling a commodity. We use a **Range Mapper** component to set the price range and **Min Max Mapper** components to create a where the price is either too low or too high and a *badValueRange* where the price is under or over the going value. We then use **And Operator** components to determine whether the value is below minimum or above maximum.`blockedRange`



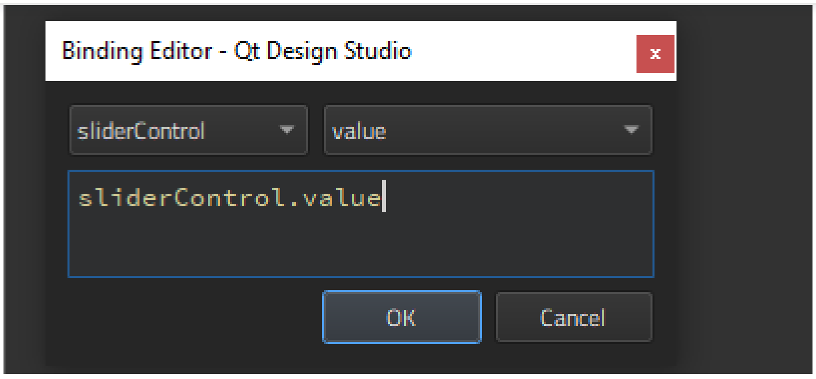
We use one **String Mapper** component to map the slider value to a **Text** component that displays the price, as instructed in String Mapper.

To define a price range from 0 to 1000, we bind the **Input** property of the **Range Mapper** component to the slider value and set the maximum input value for the price range in the **Input max** field to 1000. The minimum input value is 0 by default, so we don't need to change it.



RANGER MAPPER

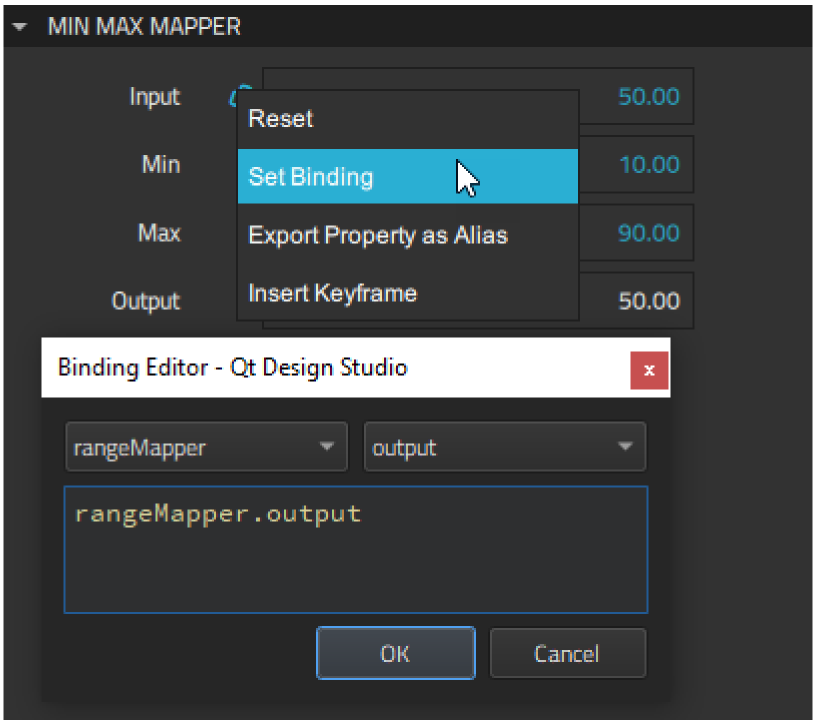
Input	500.00
Input min	0.00
Input max	1000.00
Output min	0.00
Output max	100.00



```
sliderControl.value
```

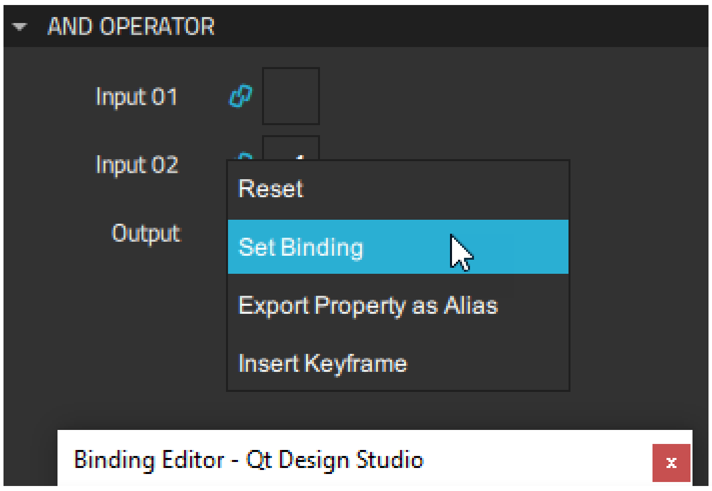
Next, we use two **Min Max Mapper** components to create a *blocked range* where the sell button will be hidden and a *bad value range* where selling is discouraged by changing the text and color of the sell hint.

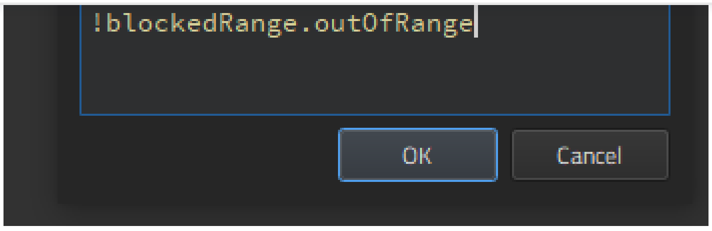
For the blocked range, we bind the **Input** property of the minimum-maximum mapper to the **Output** value of the **Range Mapper** component and specify the maximum input value in the **Max** field.



```
rangeMapper.output
```

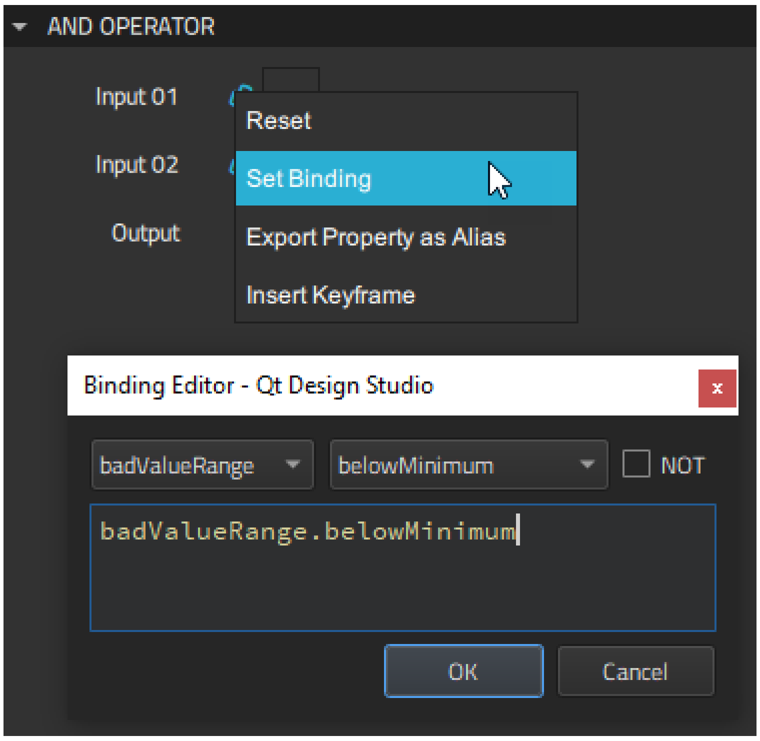
We use two **And Operator** components to determine that the sell button should be hidden when the value is in the blocked range. We do this by binding the value of the **Input 02** field to an evaluation of the value of **Out of range** field of the minimum-maximum mapper. We specify that when the value is not out of range, it evaluates to *true*.





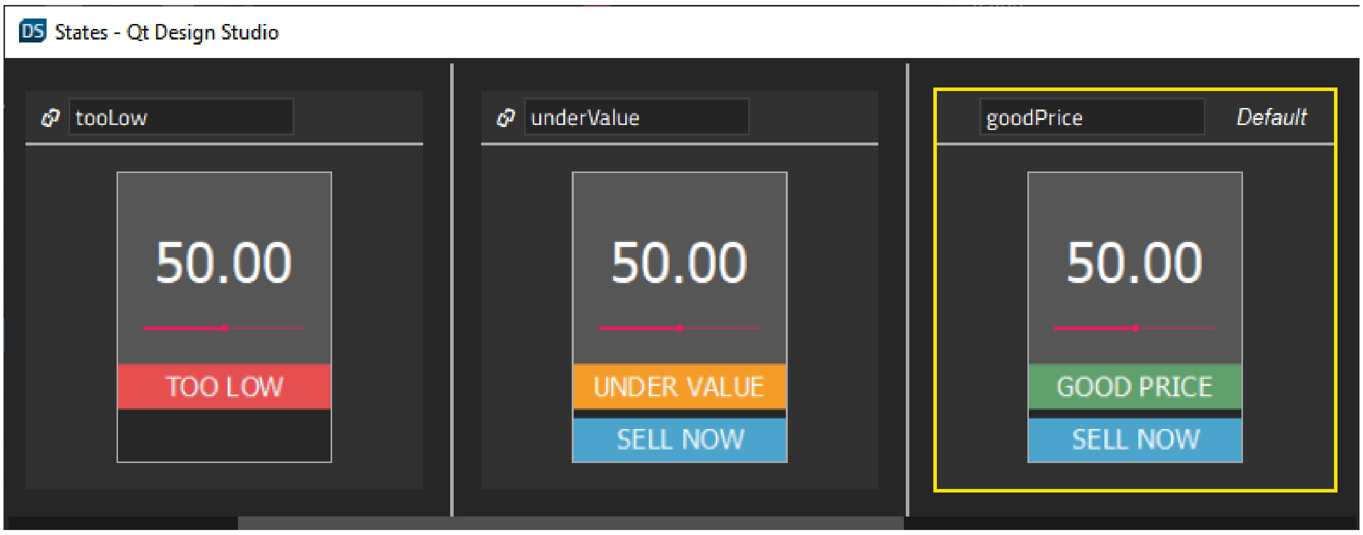
```
!blockedRange.outOfRange
```

For the *underValueAnd* operator, we additionally bind the value of the **Input 01** field to the value of the **Below min** field of the minimum-maximum mapper for the bad value range. For the *overValueAnd* operator, we bind it to the value of the **Above max** field of the the same mapper.

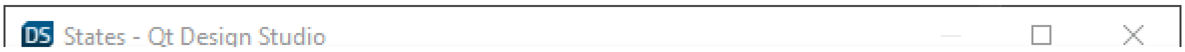


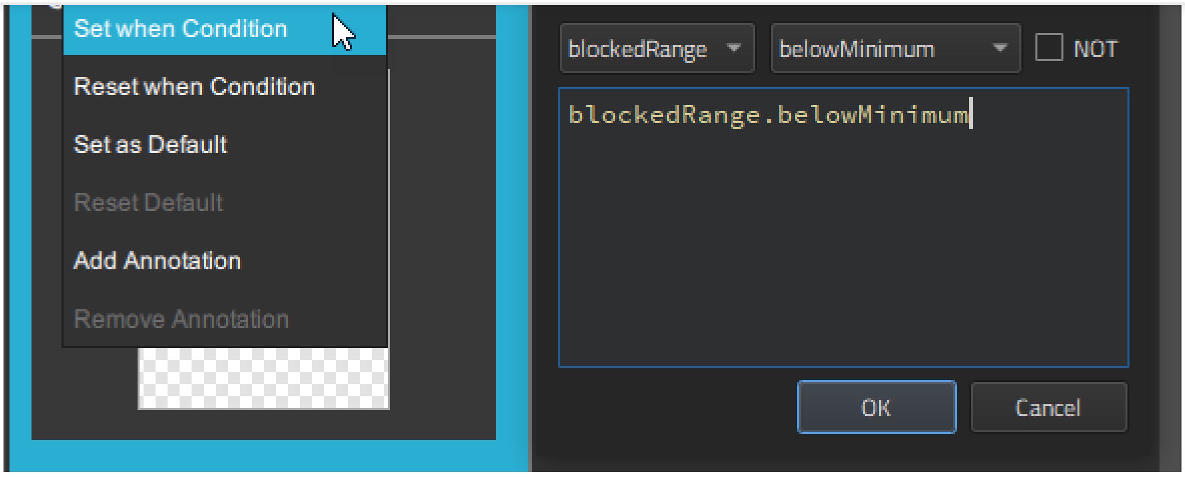
```
badValueRange.belowMinimum
```

We can now evaluate values of the **Min Max Mapper** and **And Operator** components to apply different states that display different button text and sell hints. For this purpose, we create the states and set conditions for them.when



First, we create a *tooLow* state and set a condition to apply it when the value of the **Below min** field of the **Min Max Mapper** component for the blocked range evaluates to .`whentrue`

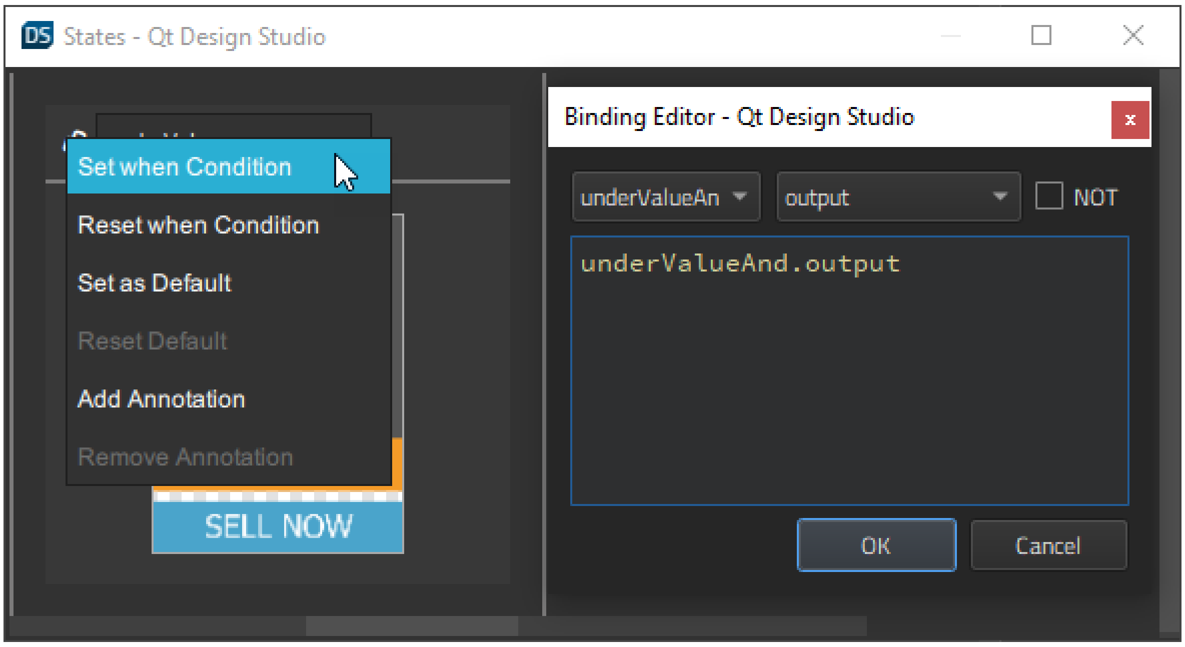




```
blockedRange.belowMinimum
```

For the *tooHigh* state at the other end of the scale, we set the condition to apply it when the value of the **Above max** field of the **Min Max Mapper** component for the blocked range evaluates to .when`true`

Next, we specify that the *underValue* state is applied when the value of the **Output** field of the *underValueAnd* **And Operator** component evaluates to .`true`



```
underValueAnd.output
```

For the *overValue* state, we set the condition to apply it when the value of the **Output** field of the *overValueAnd* component evaluates to .when`true`

When we preview our application, we can see the states applied when the slider value changes.

Summary of Logic Helpers

The following table summarizes the available logic helpers.

Icon	Logic Helper	Description
	And Operator	Boolean AND.
	Bi Direct Binding	A bi-directional binding that binds two values in both directions and keeps them in sync.

Icon Logic Helper Description

⊳∘	Not Operator	Boolean NOT.
⊳	Or Operator	Boolean OR.
⊢⊣	Range Mapper	Maps a numerical range to another range, so that the output value of the second range follows the input value of the original range.
"T"	String Mapper	Maps a number to a string with the defined precision.

‹ 2D Effects   Animations ›

Qt DOCUMENTATION

Downloads
Marketplace

Feedback Sign In