

# 用户交互方法

您可以创建预设基本组件的实例，以将交互方法添加到 UI，例如通过使用定点设备或键盘执行操作，或者水平或垂直轻拂屏幕的可见区域。它们在“**基本**”>**组件**>“**默认组件**”中可用。

此外，还可以创建预设 **UI 控件** 的实例，以通知用户有关应用程序进度或从用户收集输入。

以下基本组件可用于用户交互：

- › 鼠标区域
- › 聚焦范围
- › 可轻弹
- › 基本交互方法摘要

您可以在“**属性**”视图中指定组件实例的属性值。

## 鼠标区域

信号和处理程序用于提供鼠标交互。具体来说，您可以使用**鼠标区域**组件来定义 JavaScript 回调（也称为信号处理程序），这些回调在定义的区域中接受鼠标事件。

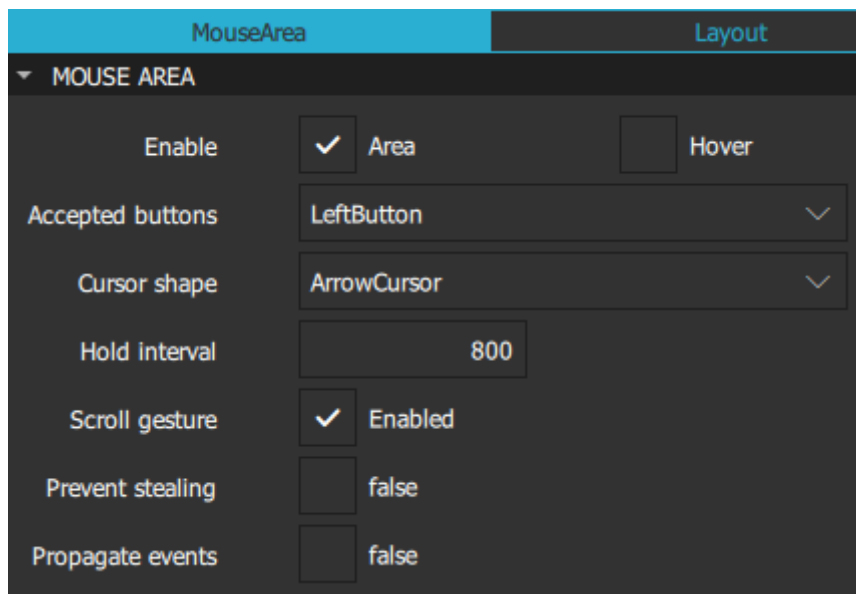
鼠标区域接收已定义区域内的事件。定义此区域的一种快速方法是将鼠标区域**锚定**到其父区域。如果父项是**矩形**（或从 **Item** 派生的任何组件），则鼠标区域将填充由父项的尺寸定义的区域。或者，您可以定义一个小于或大于父级的区域。多个控件（如**按钮**）包含鼠标区域。

鼠标区域会发出**信号**以响应不同的鼠标事件：

- › canceled()
- › clicked()
- › doubleClicked()
- › entered()
- › exited()
- › positionChanged()
- › pressAndHold()
- › pressed()
- › released()

## 鼠标区域属性

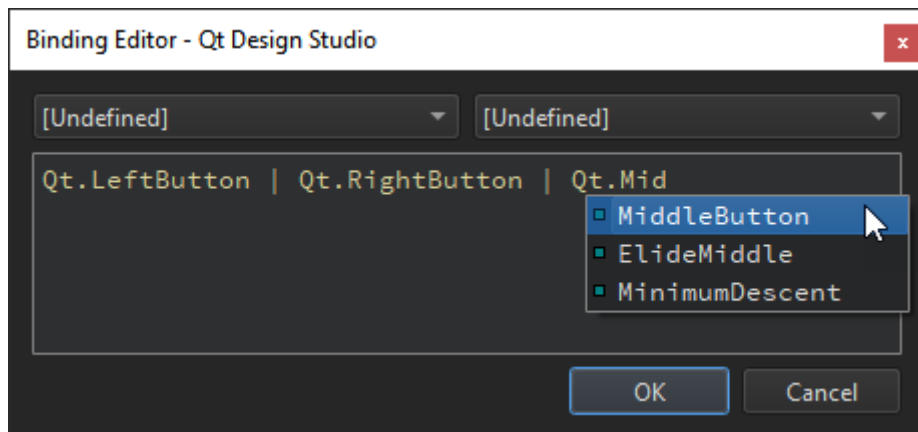
选中“**启用**”复选框以启用代理组件的鼠标处理。禁用后，鼠标区域对鼠标事件将变为透明。



默认情况下，**鼠标区域**组件仅报告鼠标单击次数，而不报告鼠标光标位置的更改。选中“**悬停**”复选框以确保使用适当的处理程序，并且即使不按下鼠标按钮，也会根据需要进行更新其他属性的值。

即使**鼠标区域**是不可见的组件，它也具有**可见**属性。取消选中“**可见性**”部分中的“**可见**”复选框，以使鼠标区域对鼠标事件透明。

在“**接受的按钮**”字段中，选择鼠标区域对其做出反应的鼠标按钮。选择“**所有按钮**”可让鼠标区域对所有鼠标按钮做出反应。可以通过将值与 OR 运算符 (|) 组合在“**代码**”视图或**绑定编辑器**中添加对多个按钮的支持。有关可用值的详细信息，请参阅[已接受按钮](#)的开发人员文档。



In the **Cursor shape** field, select the cursor shape for this mouse area. On platforms that do not display a mouse cursor, this value may have no effect.

In the **Hold interval** field, specify a value to override the elapsed time in milliseconds before the signal is emitted. If you do not explicitly set the value or it is reset, it follows the globally set application style hint. Set this value if you need particular intervals for particular **Mouse Area** instances.`pressAndHold()`

Select the **Scroll gesture** check box to respond to scroll gestures from non-mouse devices, such as the 2-finger flick gesture on a trackpad. If the check box is not selected, the wheel signal is emitted only when the wheel event comes from an actual mouse with a wheel, while scroll gesture events will pass through to any other component that will handle them. For example, the user might perform a flick gesture while the cursor is over a component containing a **Mouse Area** instance, intending to interact with a **Flickable** component which is underneath. Setting this property to will allow the **PinchArea** component to handle the mouse wheel or the pinch gesture, while the **Flickable** component handles the flick gesture.

are defined. If a mouse area overlaps with the area of other instances of the **Mouse Area** components, you can propagate `clicked()`, `doubleClicked()`, and events to these other components by selecting the **Propagate events** check box. Each event is propagated to the next enabled **Mouse Area** beneath it in the stacking order, propagating down this visual hierarchy until a **Mouse Area** accepts the event.

## Advanced Mouse Area Properties

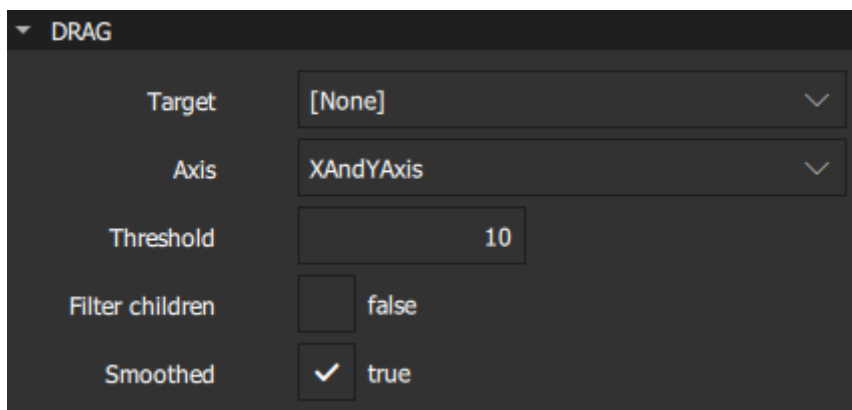
You can place a **Mouse Area** instance within a component that filters child mouse events, such as **Flickable**. However, the mouse events might get stolen from the **Mouse Area** if a gesture, such as a flick, is recognized by the parent component.

Select the **Prevent stealing** check box to stop mouse events from being stolen from the **Mouse Area** instance. This value will take no effect until the next event if it is set once a component has started stealing events.

For more information, see the developer documentation for the [Mouse Area](#) component.

## Drag Properties

You can specify properties for dragging components in the **Drag** section. Select the component to drag in the **Target** field. Keep in mind that anchored components cannot be dragged.



DRAG	
Target	[None]
Axis	XAndYAxis
Threshold	10
Filter children	<input type="checkbox"/> false
Smoothed	<input checked="" type="checkbox"/> true

In the **Axis** field, specify whether dragging can be done horizontally, vertically, or both.

In the **Threshold** field, set the threshold in pixels of when the drag operation should start. By default, this value is bound to a platform dependent value.

Select the **Filter children** check box to enable dragging to override descendant **Mouse Area** instances. This enables a parent **Mouse Area** instance to handle drags, for example, while the descendant areas handle clicks.

Select the **Smoothed** check box to move the target component only after the drag operation has started. If this check box is not selected, the target component is moved straight to the current mouse position.

## Focus Scope

When a key is pressed or released, a key event is generated and delivered to the focused component. If no component has active focus, the key event is ignored. If the component with active focus accepts the key event, propagation stops. Otherwise the event is sent to the component's parent until the event is accepted, or the root component is reached and the event is ignored.

A component has focus when the **Focus** property in the **Advanced** section is set to `true`. However, for reusable or imported components, this is not sufficient, and you should use a **Focus Scope** component.

Within each focus scope, one object may have focus enabled. If more than one component have it enabled, the last component to enable it will have the focus and the others are unresponsive, similarly to when there are no focus scopes.

focus. If this component is also a focus scope, both the focus scope and the sub-focused component will have active focus.

The **Focus Scope** component is not a visual component and therefore the properties of its children need to be exposed to the parent component of the focus scope. **Layouts** and **positioners** will use these visual and styling properties to create the layout.

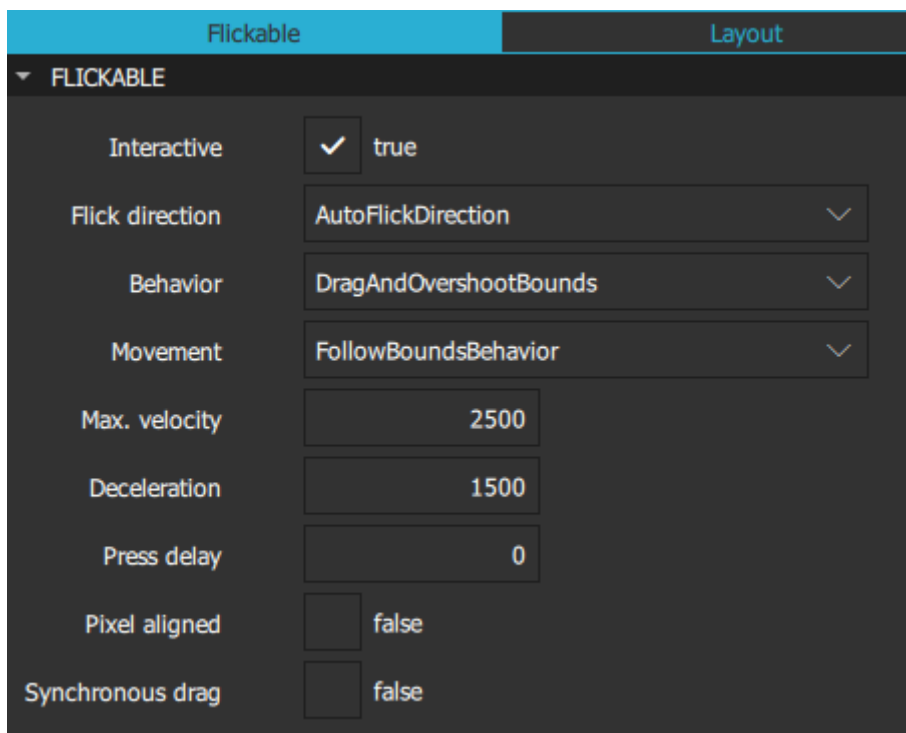
For more information, see [Keyboard Focus in Qt Quick](#).

## Flickable

**Flickable** places its children on a surface that can be dragged and flicked, causing the view onto the child components to scroll. This behavior forms the basis of components that are designed to show large numbers of child components, such as **List View** and **Grid View**. For more information, see [List and Grid Views](#).

In traditional user interfaces, views can be scrolled using standard controls, such as scroll bars and arrow buttons. In some situations, it is also possible to drag the view directly by pressing and holding a mouse button while moving the cursor. In touch-based user interfaces, this dragging action is often complemented with a flicking action, where scrolling continues after the user has stopped touching the view.

The contents of a **Flickable** component are not automatically clipped. If the component is not used as a full-screen component, consider selecting the **Clip** check box in the **Visibility** section.



Users can interact with a flickable component if the **Interactive** check box is set to **true**. Set it to **false** to temporarily disable flicking. This enables special interaction with the component's children. For example, you might want to freeze a flickable map while scrolling through a pop-up that is a child of the **Flickable** component.

The **Flick direction** field determines whether the view can be flicked horizontally or vertically. Select **AutoFlickDirection** to enable flicking vertically if the content height is not equal to height of the flickable and horizontally if the content width is not equal to the width of the flickable. Select **AutoFlickIfNeeded** if the content height or width is greater than that of the flickable.

Specify the maximum velocity for flicking the view in pixels per second in the **Max. velocity** field. Specify the rate at which a flick will decelerate in the **Deceleration** field.

drags or flicks beyond the bounds of the flickable depending on the value of the **Behavior** field.

In the **Press delay** field, specify the time in milliseconds to delay delivering a press to children of a flickable. This can be useful when reacting to a press before a flicking action has undesirable effects. If the flickable is dragged or flicked before the delay times out, the press event will not be delivered. If the button is released within the timeout, both the press and release will be delivered.

**Note:** For nested flickables with press delay set, the press delay of outer flickables is overridden by the innermost flickable. If the drag exceeds the platform drag threshold, the press event will be delivered regardless of this property.

The **Pixel aligned** check box sets the unit of alignment set in the **Content X** and **Y** fields to pixels ( ) or subpixels ( ). Set it to to optimize for still content or moving content with high contrast edges, such as one-pixel-wide lines, text, or vector graphics. Set it to when optimizing for animation quality.`truefalse`

If **Synchronous drag** is set to , then when the mouse or touchpoint moves far enough to begin dragging the content, the content will jump, so that the content pixel which was under the cursor or touchpoint when pressed remains under that point. The default is , which provides a smoother experience (no jump) at the cost of losing some of the drag distance at the beginning.`truefalse`

## Flickable Geometry

The **Content size** field specifies the dimensions of the surface controlled by a flickable. Typically, set the values of the **W** and **H** fields to the combined size of the components placed in the flickable. You can set additional margins around the content in the **Left margin**, **Right margin**, **Top margin**, and **Bottom margin** fields.

FLICKABLE GEOMETRY




Content size	<input type="text" value="0"/>	W	<input type="text" value="0"/>	H
Content	<input type="text" value="0"/>	X	<input type="text" value="0"/>	Y
Origin	<input type="text" value="0"/>	X	<input type="text" value="0"/>	Y
Left margin	<input type="text" value="0"/>			
Right margin	<input type="text" value="0"/>			
Top margin	<input type="text" value="0"/>			
Bottom margin	<input type="text" value="0"/>			

The **Origin** field specifies the origin of the content. It refers to the top-left position of the content regardless of layout direction. Usually, the **X** and **Y** values are set to 0. However, a **List View** and **Grid View** may have an arbitrary origin due to delegate size variation, or component insertion or removal outside the visible region.

## Summary of Basic Interaction Methods

The following table lists the components that you can use to add basic interaction methods to UIs with links to their developer documentation. They are available in **Components > Default Components > Basic**. The *MCU* column indicates which components are supported on MCUs.

Icon	Name	MCU	Purpose Purpose
------	------	-----	--------------------

	Focus Scope		Assists in keyboard focus handling when building reusable components.
	Mouse Area		Enables simple mouse handling.

< Images

UI Controls >



Contact Us

Company

- About Us
- Investors
- Newsroom
- Careers
- Office Locations

Licensing

- Terms & Conditions
- Open Source
- FAQ

Support

- Support Services
- Professional Services
- Partners
- Training

For Customers

- Support Center
- Downloads
- Qt Login
- Contact Us
- Customer Success

Community

- Contribute to Qt
- Forum
- Wiki
- Downloads
- Marketplace

