Future of pointer events in Qt Quick

Qt Contributors' Summit 2015

Shawn Rutledge
shawn.rutledge@theqtcompany.com
ecloud on #qt-labs

Promise that we might break

- Plan for last 3 years was to handle touch directly in MouseArea, Flickable etc.
- problem: parallel delivery paths (even worse if we try to support QTabletEvent too)
- problem: hard to get all existing tests passing
- problem: nobody wants to give +2 on the patches
- problem: some people insist on making it opt-in
- problem: MouseArea doesn't even sound like it should handle touch

Stuff we'd really better fix

- make it easy to handle mouse, touch and tablet agnostically
- guarantee that events always have velocity
- plan on unified QPointingEvent delivery in Qt 6: make sure current changes are compatible
- plan on multiple seats/users and support for multiple mice etc. in Qt 6

Idea: attached handlers

- make mouse/touch more like the Keys attached prop
- no need to bind the geometry: proxy for the attachee Item
- Tap.onFinished is a one-liner (e.g. for Button)
- lots of small, understandable handlers instead of monolithic MouseArea etc.
- preferred: handlers for gestures (tap, drag, pinch etc.)
- but also: some for mouse only, some for touch only, just in case
- it's completely different, so minimizes compatibility risks
- problems: only one attached per Item, repetitive syntax, inefficiency

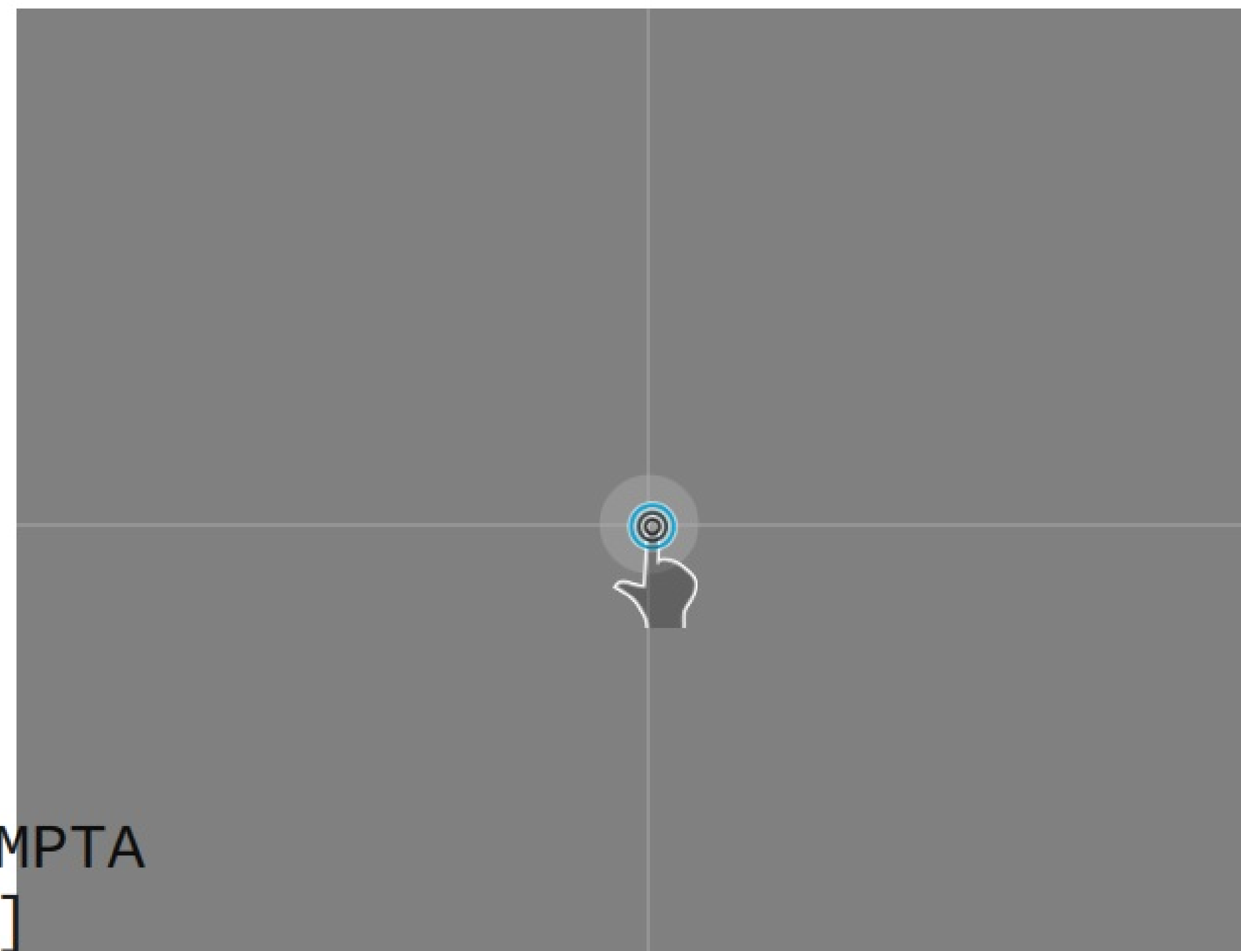
```
Item {
   Mouse.wheel.angle
   Mouse.hoverEnabled
   Mouse.pos (with x and y sub-properties)
   Mouse.velocity (with x and y sub-properties)
   Mouse.clickCount
   Mouse.pressedButtons
      probably can do without mouse-specific drag.target
   Mouse.cursor

   Touch.points
      TouchPoint.pos (with x and y sub-properties)
      TouchPoint.velocity (with x and y sub-properties)
      TouchPoint.id
   Touch.tapCount

   Tap.onStarted:
   Tap.onUpdated:
   Tap.onFinished:
   Tap.onCanceled:

   Carry.onStarted:
   Carry.onUpdated:
   Carry.onFinished:
   Carry.onCanceled:
   Carry.target (or not?  it's implied by what item it's attached to)
   Carry.pos (with x and y sub-properties)
   Carry.velocity (with x and y sub-properties)
   Carry.momentum, mass, friction, something like that
      ...make flicking/scrolling the same as carrying by putting a Carry on an explicitly-created content Item which contains other stuff, and which is inside a clipping Item or underneath masking items

   Pinch.onStarted:
   Pinch.onUpdated:
   Pinch.onFinished:
   Pinch.onCanceled:
}
```

Attached Carry gesture handler (because the name Drag is taken)

```
 1: import QtQuick 2.6
 2:
 3: Item {
 4:     id: root
 5:     width: 640
 6:     height: 480
 7:
 8:     Rectangle {
 9:         id: rect
10:         width: 80; height: 80; x: 280; y: 200; radius: 5
11: //          anchors.centerIn: parent // Carry doesn't break anchors, but maybe it should (?)
12:
13:         Carry.enabled: true
14:         color: Carry.dragging ? "red" : "green"
15:         Carry.onPressed: chip.createObject(root, {"x": point.sceneX, "y": point.sceneY})
16:         Carry.onReleased: {
17:             console.log("released @ " + point)
18:             xanim.end = rect.x + point.velocity.x / 8
19:             yanim.end = rect.y + point.velocity.y / 8
20:             xanim.start()
21:             yanim.start()
22:         }
23:         // I just had to write "Carry" 4 times.  That's attached properties for you...
24:
25:         Image {
26:             source: "resources/tooth.png"
```

Attached Touch gesture handler

```
 1: import QtQuick 2.6
 2: import QtQuick.Window 2.0
 3:
 4: Rectangle {
 5:     id: root
 6:
 7:     Repeater {
 8:         model: 10
 9:         Item {
10:             anchors.fill: parent
11:
12:             // Have to give IDs to touchpoints just as in MPTA
13:             Touch.points: [ TouchPoint { id: touchPoint } ]
14:             Touch.maximumTouchPoints: 1
15:             Touch.enabled: true
16:
17:             Item {
18:                 visible: touchPoint.pressed
19:                 anchors.fill: parent
20:                 Rectangle {
21:                     color: root.colors[index]
22:                     anchors.top: parent.top
23:                     anchors.bottom: parent.bottom
24:                     width: 2
25:                     x: touchPoint.x - 1
26:                 }
```

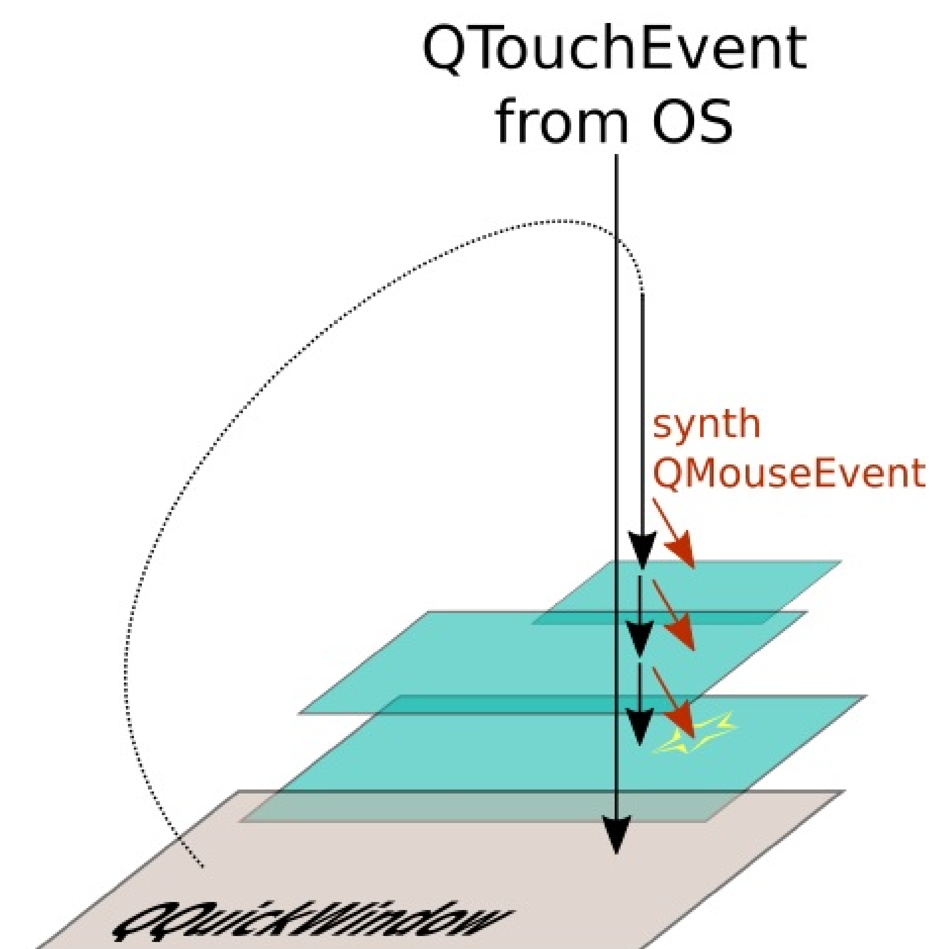


Touch Event Delivery in Qt <= 5.5

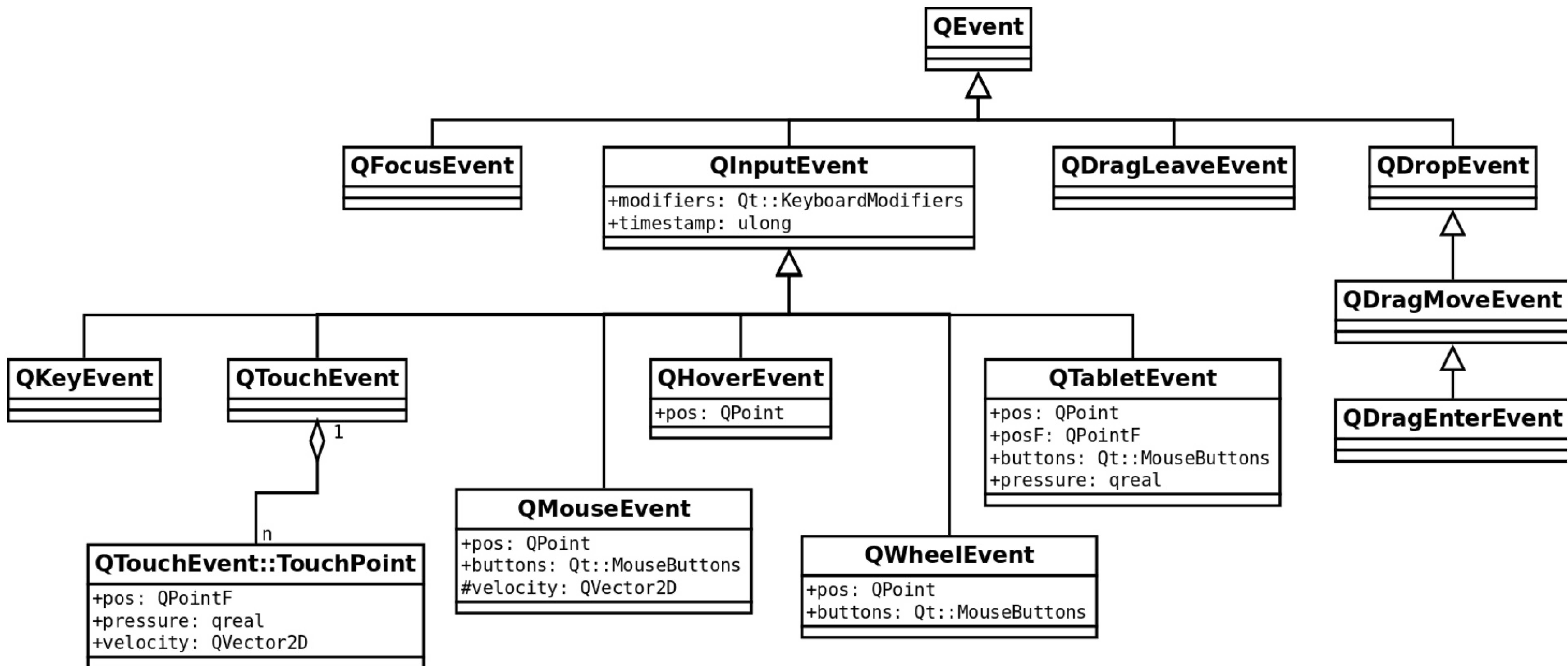
- each item under the touchpoint gets a touch event (and ignores by default)

- if not accepted, the item gets a synthetic mouse event

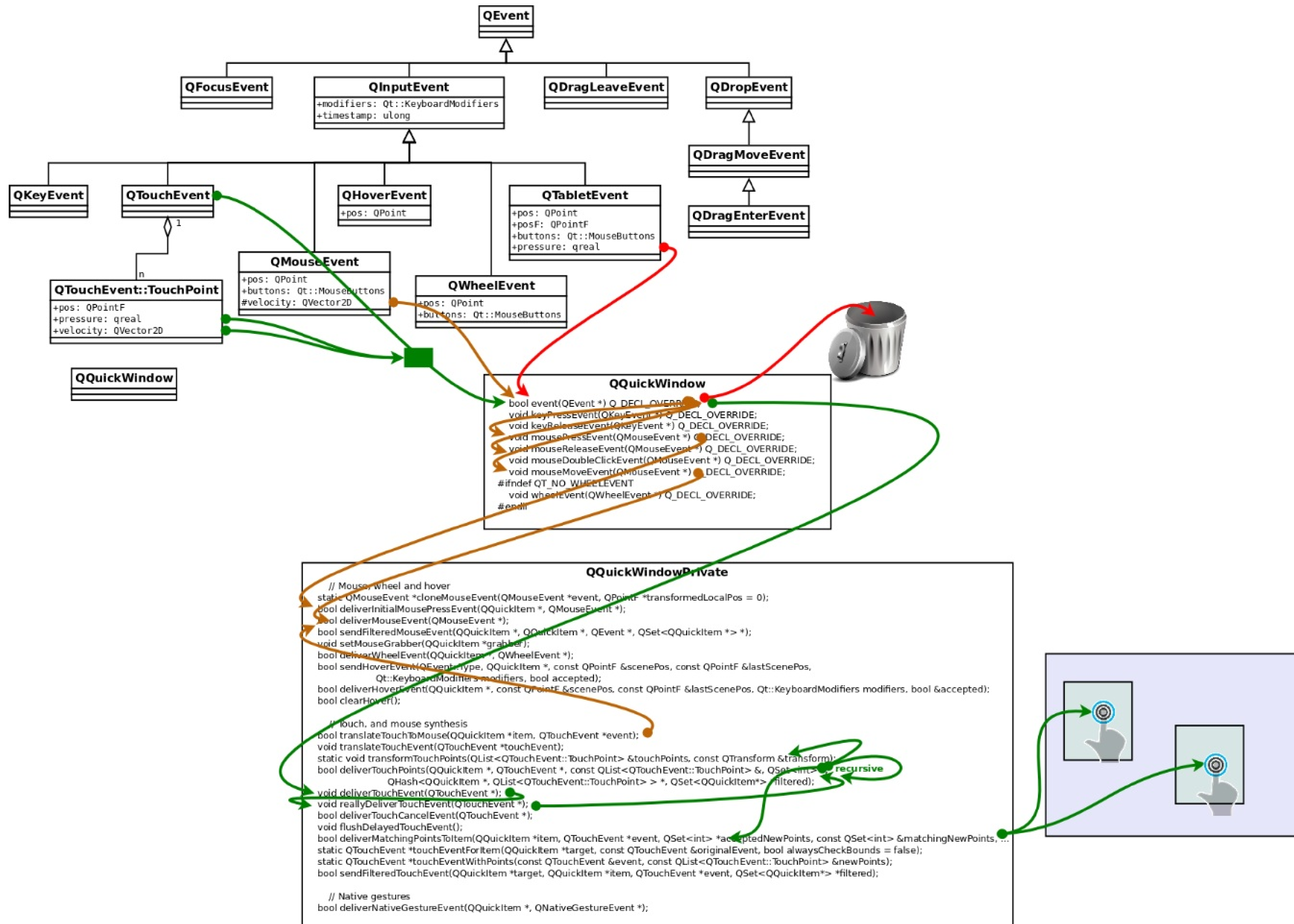
- continue with next item (in reverse paint order) if still not accepted



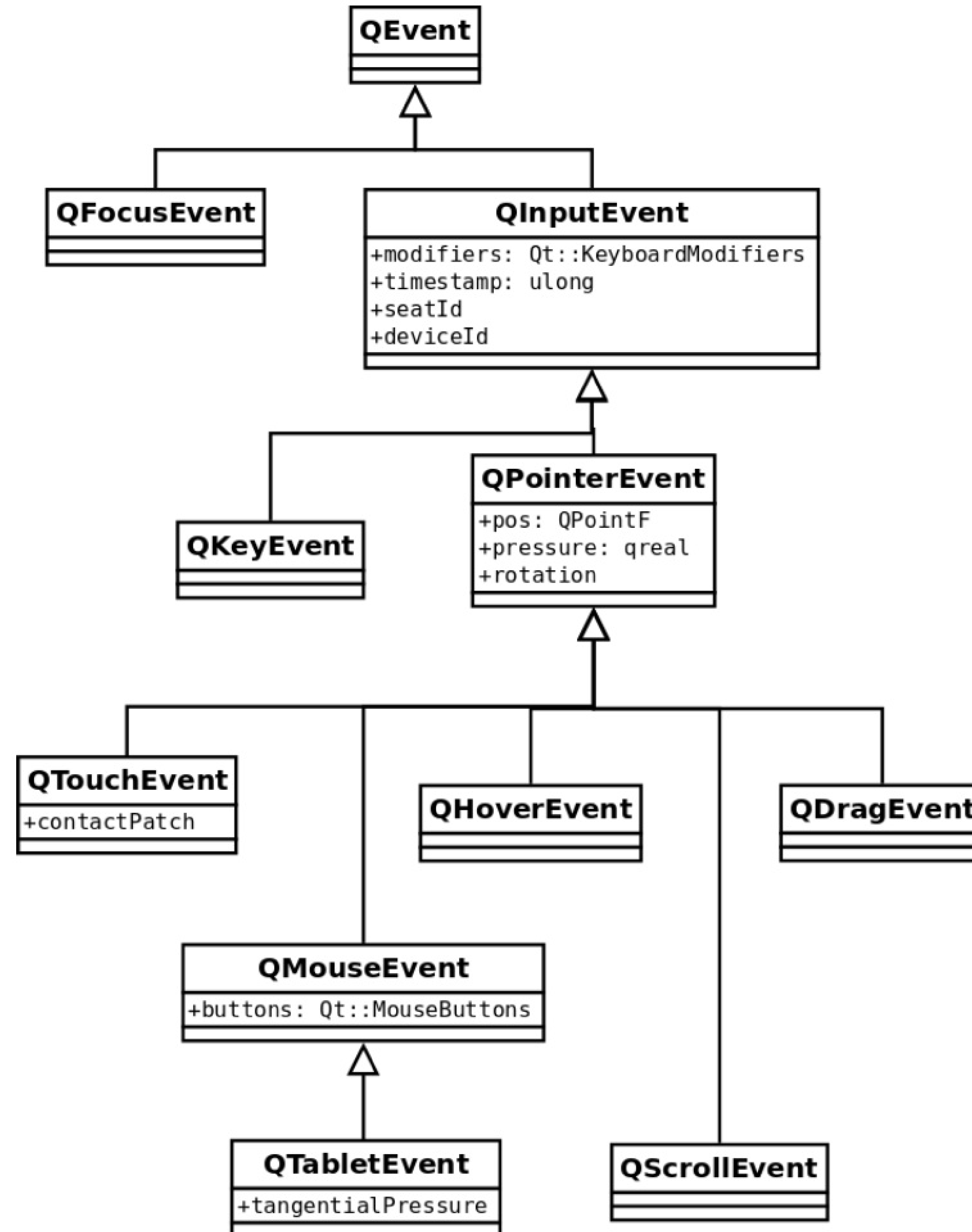
Existing QEvent hierarchy



Many parallel event delivery paths (and some missing)

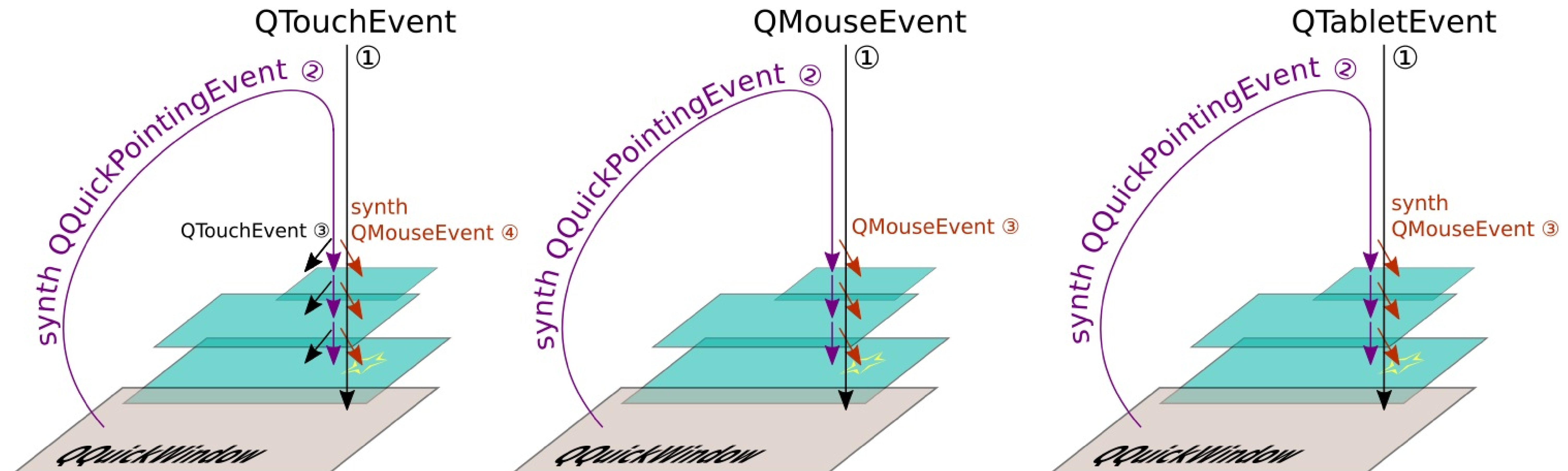


Possible hierarchy for Qt 6



One of the proposals for Qt 5.6 or 5.7

- preferred event for Items is QQuickPointerEvent (synthetic for now)
- multi-touch, tablet and mouse handling are agnostic until you need to distinguish them



QQuickPointerEvent

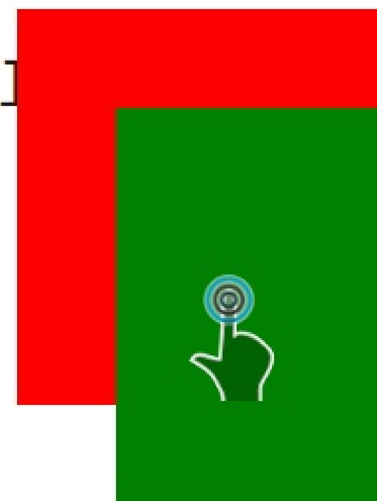
```
// (barely?) W3C-inspired pointer event
// Template maybe?
class Q_QUICK_PRIVATE_EXPORT QQuickPointerEvent
{
    Q_GADGET
    Q_PROPERTY(qreal x READ x)
    Q_PROPERTY(qreal y READ y)
//    Q_PROPERTY(qreal sceneX READ sceneX)
//    Q_PROPERTY(qreal sceneY READ sceneY)
    Q_PROPERTY(QPointF center READ center)
    // TODO elliptical contact point: major, minor, rotation
    Q_PROPERTY(qreal rotation READ rotation)
    Q_PROPERTY(Qt::MouseButton button READ button)
    Q_PROPERTY(Qt::MouseButtons buttons READ buttons)
    Q_PROPERTY(Qt::KeyboardModifiers modifiers READ modifiers)
    Q_PROPERTY(qreal timeHeld READ timeHeld)
    Q_PROPERTY(bool isTap READ isTap)
    Q_PROPERTY(bool accepted READ isAccepted WRITE setAccepted)

public:
    QQuickPointerEvent(QMouseEvent *ev, QMouseEvent *pressEv = 0) : m_event(ev) {
        if (pressEv)
            _pressTimestamp = pressEv->timestamp();
        else
            _pressTimestamp = ev->timestamp();
    }
    QQuickPointerEvent(QWheelEvent *ev) : m_event(ev) { }
    QQuickPointerEvent(QNativeGestureEvent *ev) : m_event(ev) { }
    QQuickPointerEvent(QTouchEvent *ev) : m_event(ev) { }
    QQuickPointerEvent(QTabletEvent *ev) : m_event(ev) { }

    qreal x() const { return _x; }
    qreal y() const { return _y; }
    Qt::MouseButton button() const { return _button; }
    Qt::MouseButtons buttons() const { return _buttons; }
    Qt::KeyboardModifiers modifiers() const { return m_event->modifiers(); }
    // Only correct if QInputEvent::timestamp() is always in ms
    qreal timeHeld() const { return qreal(m_event->timestamp() - _pressTimestamp) / 1000.0; }
    bool isTap() const { return _isTap; }
```

Alternatives to attached handlers

```
Item {

    // Child object rather than attached object
    MouseArea {
        // drag
        drag.target: parent
    }

    MultiPointTouchArea {
        // drag
    }

    // ----------------------------------

    // Supplemental, optional attached property
    Pointer.onPressed:
    Pointer.onReleased:

    PointerArea {
        id: mouse
        sourceFilter: Pointer.MouseSource
        onPressed:
        onReleased:
    }
    Pointer {
        id: touch
        sourceFilter: Pointer.TouchSource
        onPressed:
        onReleased:
    }

    ScrollArea

    Touch {
        points:
    }

    Touch.points: [
        TouchPoint { id: touchPoint1; onPositionChanged: },
```

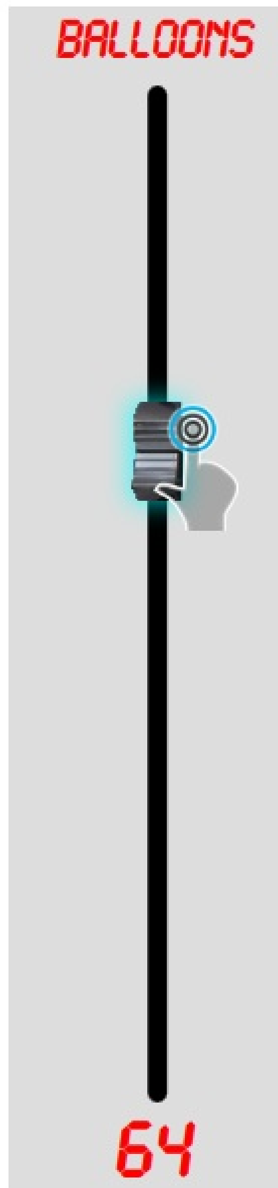
Events, Pressed and Hover

```
 1: import QtQuick 2.0
 2:
 3: Item {
 4:     x: 50; y: 200;  width: 250; height: 250;
 5:     Rectangle {
 6:         width: 200; height: 200;
 7:         color: mouse.pressed ? "red" : (mouse.containsMouse ? "blue" : "lightsteel
 8:         MouseArea {
 9:             id: mouse
10:             anchors.fill: parent
11:             hoverEnabled: true
12:         }
13:     }
14:     Rectangle {
15:         x: 50; y: 50; width: 200; height: 200;
16:         color: mouse2.containsMouse ? "green" : "aquamarine"
17:
18:         MouseArea {
19:             id: mouse2
20:             anchors.fill: parent
21:             hoverEnabled: true
22:             // Should it be possible to reject hover too?
23:             onPressed: mouse.accepted = false
24:         }
25:     }
26: }
```



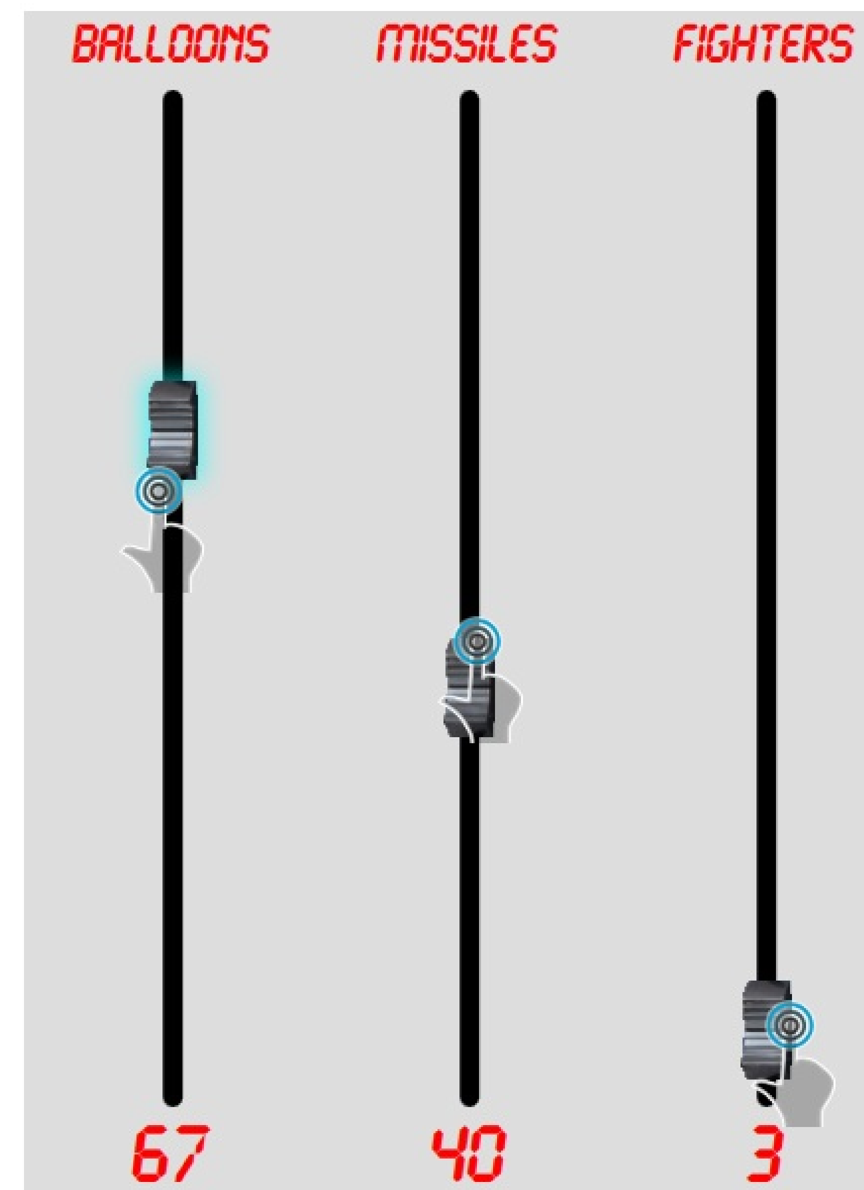
Dragging: Slider

```
 1:  import QtQuick 2.6
 2:  import QtGraphicalEffects 1.0
 3:
 4:  Rectangle {
 5:      id: slider
 6:      width: 150; height: 600; color: "#ddd"
 7:      property int value: 50
 8:      property int minimumValue: 0
 9:      property int maximumValue: 99
10:      property alias label: label.text
11:
12:      Image {
13:          id: knob; source: "../images/fader-knob.png"
14:          x: slot.x - width / 2 + slot.width / 2; z: 2
15:          transformOrigin: Item.Center
16:          MouseArea {
17:              id: dragArea
18:              anchors.fill: parent
19:              anchors.margins: -20
20:              drag.target: parent
21:              drag.axis: Drag.YAxis
22:              drag.minimumY: slot.y
23:              drag.maximumY: slot.height + slot.y - parent.height
24:              // multiPointTouchEnabled: true // Qt 5.6 - planned
25:          }
26:          property real multiplier: slider.maximumValue / (dragArea.drag.maximumY - dragArea.
```



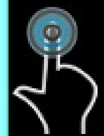
Multiple Sliders

```
1:  import QtQuick 2.6
2:
3:  Row {
4:      Slider { label: "Balloons"; value: 99 }
5:      Slider { label: "Missiles"; value: 40 }
6:      Slider { label: "Fighters"; value: 3 }
7:  }
```



Slide panel with MouseArea

```
1:  import QtQuick 2.0
2:  import QtGraphicalEffects 1.0
3:
4:  Item {
5:      id: root
6:      property real xOpen: xClosed - 150
7:      property real xClosed: 200
8:      x: xClosed; y: 10; width: 300; height: 480
9:
10:     MouseArea {
11:         id: ma
12:         anchors.fill: parent
13:         property real lastWinX: 0
14:         property real lastMouseX: 0
15:         onPressed: {
16:             state = ""
17:             lastWinX = root.x
18:             lastMouseX = mouseX
19:         }
20:         onMouseXChanged: root.x = Math.max(xOpen, root.x + (mouseX - lastMouseX))
21:         onReleased: {
22:             if (root.x - lastWinX > 20) state = "stateClosed"
23:             else state = "stateOpen"
24:         }
25:         state: "stateClosed"; Component.onCompleted: state = "stateOpen"
26:         states: [
```

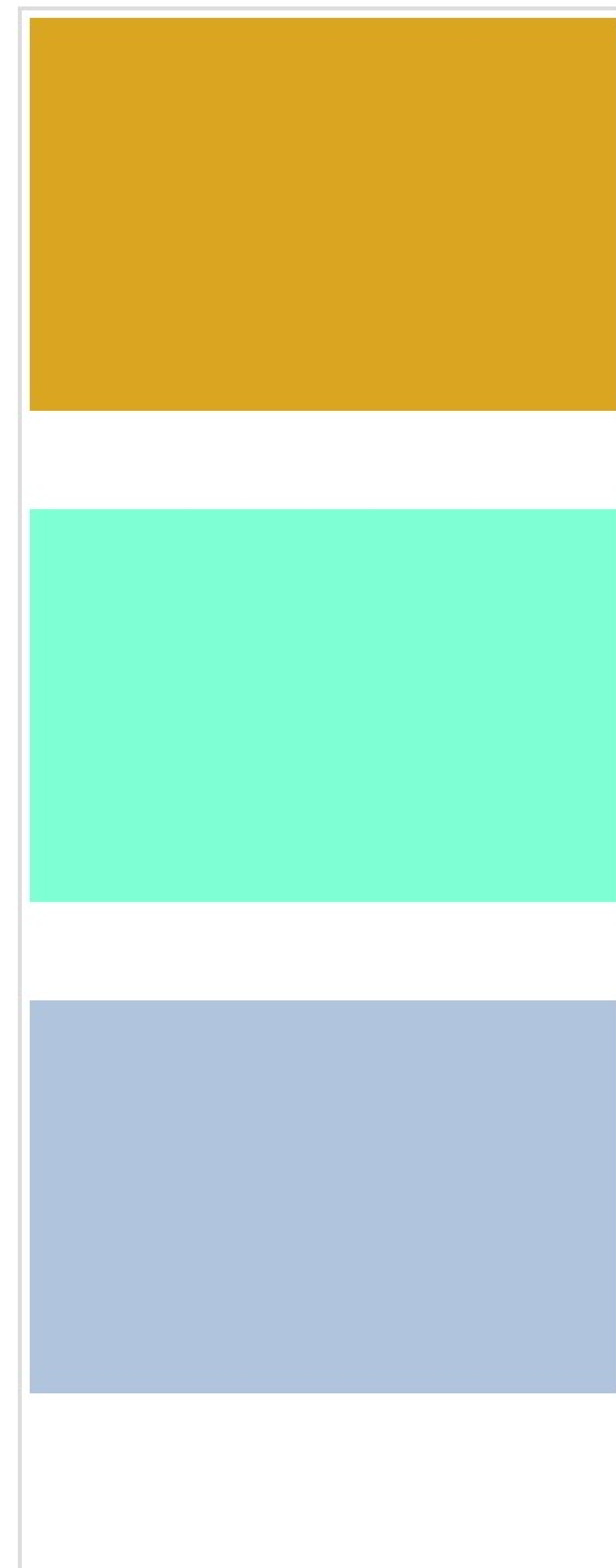


Slide panel with Flickable

```
 1:  import QtQuick 2.0
 2:  import QtGraphicalEffects 1.0
 3:
 4:  Flickable {
 5:      id: root
 6:      default property alias __content: rect.data
 7:
 8:      onDraggingChanged: if (!dragging) {
 9:         if (horizontalVelocity > 0) flick(-1000, 0)
10:         else flick(1000, 0)
11:      }
12:      flickableDirection: Flickable.HorizontalFlick
13:      rebound: Transition { NumberAnimation {
14:              properties: "x"
15:              duration: 300
16:              easing.type: Easing.OutBounce } }
17:
18:      width: 100; height: 480
19:      contentWidth: width * 2; contentHeight: height
20:
21:      RectangularGlow {
22:          x: width - 10; width: root.width; height: root.height
23:          glowRadius: 10; spread: 0.2; color: "cyan"
24:          cornerRadius: rect.radius + glowRadius
25:
26:          Rectangle {
```

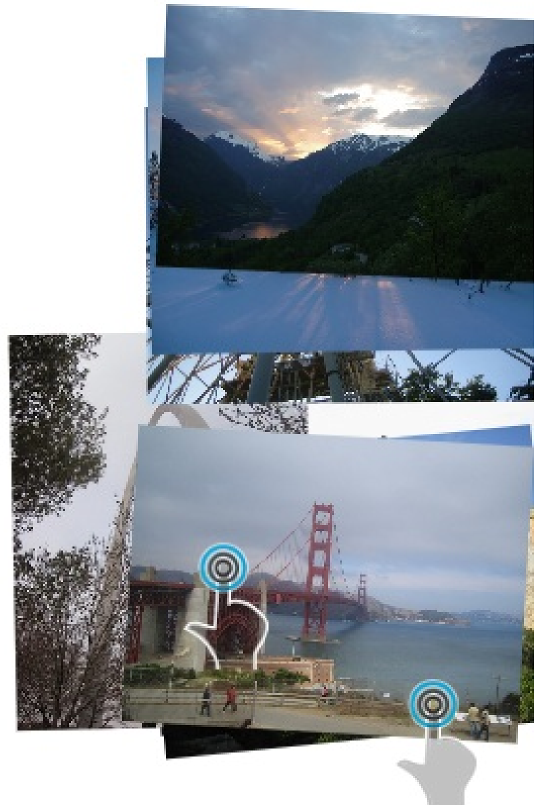
Prevent Stealing

```
1:  import QtQuick 2.0
2:
3:  Rectangle {
4:      width: 312; height: 800; color: "transparent"; border.width: 2
5:      border.color: flick.dragging ? "green" : "#ddd"
6:      Flickable {
7:          id: flick
8:          anchors.fill: parent
9:          contentHeight: col.implicitHeight * 2
10:         Column {
11:             id: col; x: 6; y: 6; spacing: 50
12:             Rectangle {
13:                 width: 300; height: 200
14:                 color: mouse1.pressed ? "yellow" : "goldenrod"
15:                 MouseArea { id: mouse1; anchors.fill: parent }
16:             }
17:             Rectangle {
18:                 width: 300; height: 200
19:                 color: mouse2.pressed ? "green" : "aquamarine"
20:                 MouseArea {
21:                     id: mouse2; anchors.fill: parent;
22:                     onPressed: mouse.accepted = false
23:                 }
24:             }
25:             Rectangle {
26:                 width: 300; height: 200
```



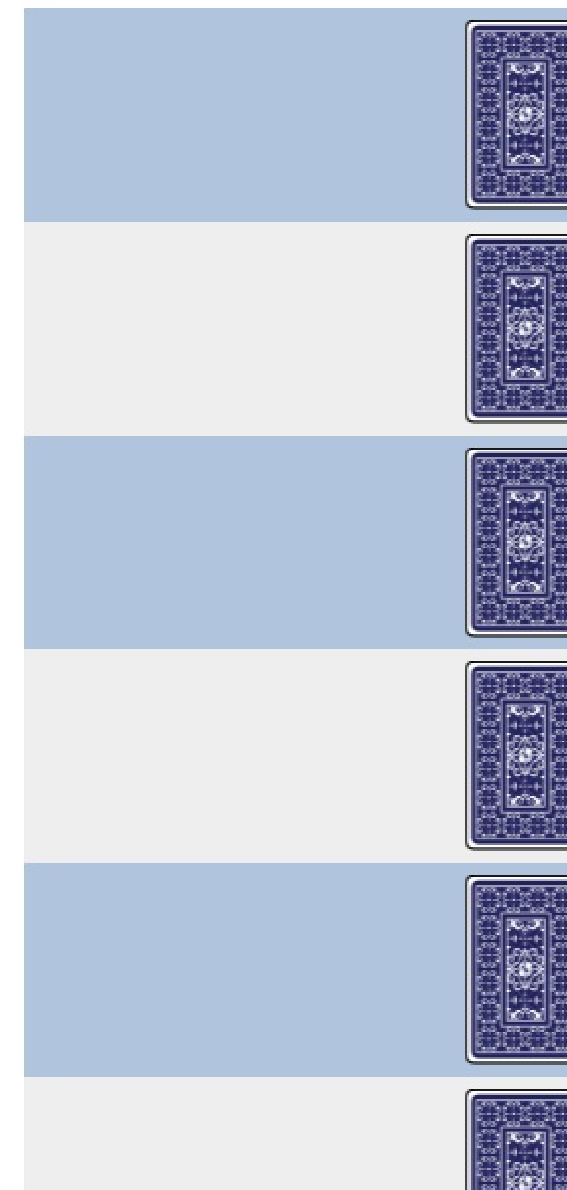
PinchArea with nested MouseArea for single-finger dragging

```
1:  import QtQuick 2.0
2:  import Qt.labs.folderlistmodel 1.0
3:
4:  Item { width: 800; height: 800
5:      Repeater {
6:          id: root; anchors.fill: parent
7:
8:          Image {
9:              source: folderModel.folder + fileName
10:
11:             PinchArea {
12:                 anchors.fill: parent
13:                 pinch.target: parent
14:                 pinch.minimumRotation: -360
15:                 pinch.maximumRotation: 360
16:                 pinch.minimumScale: 0.1
17:                 pinch.maximumScale: 10
18:                 pinch.dragAxis: Pinch.XAndYAxis
19:                 MouseArea {
20:                     anchors.fill: parent
21:                     drag.target: parent.parent
22:                     onPressed: parent.parent.z = ++root.zmax
23:                 }
24:             }
25:
26:             scale: root.defaultSize / Math.max(sourceSize.width, sourceSize.height)
```



MultiPointTouchArea: recognize custom gestures (so we say)

```
 1: import QtQuick 2.0
 2: import QtQuick.Particles 2.0
 3:
 4: ListView {
 5:     id: list; model: 13
 6:     height: 600; width: 300; clip: true
 7:     delegate: Rectangle {
 8:         color: index % 2 == 0 ? "lightsteelblue" : "#eee"
 9:         width: list.width
10:         height: 108
11:         MultiPointTouchArea {
12:             anchors.fill: parent
13:             minimumTouchPoints: 1; maximumTouchPoints: 1
14:             property bool interested: false
15:             onGestureStarted: {
16:                 var tp = gesture.touchPoints[0]
17:                 if (Math.abs(tp.startY - tp.y) < gesture.dragThreshold)
18:                     interested = true
19:             }
20:             onUpdated: if (interested) {
21:                 var tp = touchPoints[0]
22:                 rotation.angle = 2 * (tp.startX - tp.x)
23:             }
24:             function finish() {
25:                 if (rotation.angle > 90 || rotation.angle < -90)
26:                     rotation.angle = 180
```



MultiPointTouchArea substituting for MouseArea

```
 1: import QtQuick 2.0
 2:
 3: Rectangle {
 4:     id: root
 5:     width: label.implicitWidth * 1.5; height: label.implicitHeight * 2.0
 6:     border.color: "#9f9d9a"; border.width: 1; radius: height / 4; antialiasing: true
 7:     property alias label: label.text
 8:     property alias pressed: touch1.pressed
 9:     signal clicked
10:
11:     gradient: Gradient {
12:         GradientStop { position: 0.0; color: touch1.pressed ? "#b8b5b2" : "#efebe7" }
13:         GradientStop { position: 1.0; color: "#b8b5b2" }
14:     }
15:
16:     MultiPointTouchArea {
17:         anchors.fill: parent
18:         touchPoints: [
19:             TouchPoint {
20:                 id: touch1
21:                 onPressedChanged: if (!pressed) root.clicked
22:             } ]
23:     }
24:
25:     Text {
26:         id: label
```