BLUESCAPE

From desk to wall… and back

Touch-and-pen-proof Qt applications

Romain Pokrzywka – Native Application Architect

08/09/2020

Overview

1. Touch and pen input support in Qt
2. Mixed input applications
3. Tips & tricks
4. Input stack changes in Qt6

Touch and Pen input support in Qt

BLUESCAPE®

Multi-Touch support (C++)

```
QWindow::touchEvent(QTouchEvent*)
QWidget::event(QEvent*)
```

```
case QEvent::TouchBegin:
case QEvent::TouchUpdate:
case QEvent::TouchEnd:
case QEvent::TouchCancel:

    for (const QTouchEvent::TouchPoint& touchPoint : te->touchpoints())
        qDebug() << touchPoint.id() << touchPoint.pos() << touchPoint.state();
```

```
Qt::TouchPointPressed
Qt::TouchPointMoved
Qt::TouchPointStationary
Qt::TouchPointReleased
```

BLUESCAPE®

Pen support (C++)

```
QWindow::tabletEvent(QTabletEvent*)
QWidget::event(QEvent*)
```

```
case QEvent::TabletPress:
case QEvent::TabletMove:
case QEvent::TabletRelease:
case QEvent::TabletEnterProximity:
case QEvent::TabletLeaveProximity:
    qDebug() << te->uniqueId() << te->pos() << te->device() << te->pointerType();
```

```
QTabletEvent::Pen
QTabletEvent::Eraser
QTabletEvent::Cursor
```

Multi-Touch support (QML)

Area Item API

```
MultiPointTouchArea {
    anchors.fill: parent

    minimumTouchPoints: 1
    maximumTouchPoints: 2

    onTouchUpdated: {
        console.log(touchPoints.length)
    }
}
```

Multi-Touch support (QML)

Pointer handlers (Qt 5.10+)

```
DragHandler {
    target: anotherItem // default is parent
}

Text {
    id: text
    text: handler.translation.x + " " + (handler.scale * 100) + "%; deg=" + handler.rotation

    PinchHandler {
        id: handler
        target: null // to only use the handler properties
    }
}
```

Pen/Touchpad support (QML)

Additional pointer handlers (Qt 5.10+)

`TapHandler`

`WheelHandler`

`HoverHandler`

`PointHandler`

Other useful flags/options

QGuiApplication

`Qt::AA_SynthesizeMouseForUnhandledTouchEvents` (default true)

`Qt::AA_SynthesizeMouseForUnhandledTabletEvents` (default true)

`Qt::AA_SynthesizeTouchForUnhandledMouseEvents` (default false)

QWidget

`Qt::WA_AcceptTouchEvents`

Windows Platform (QPA)

`nomousefromtouch`

QScroller

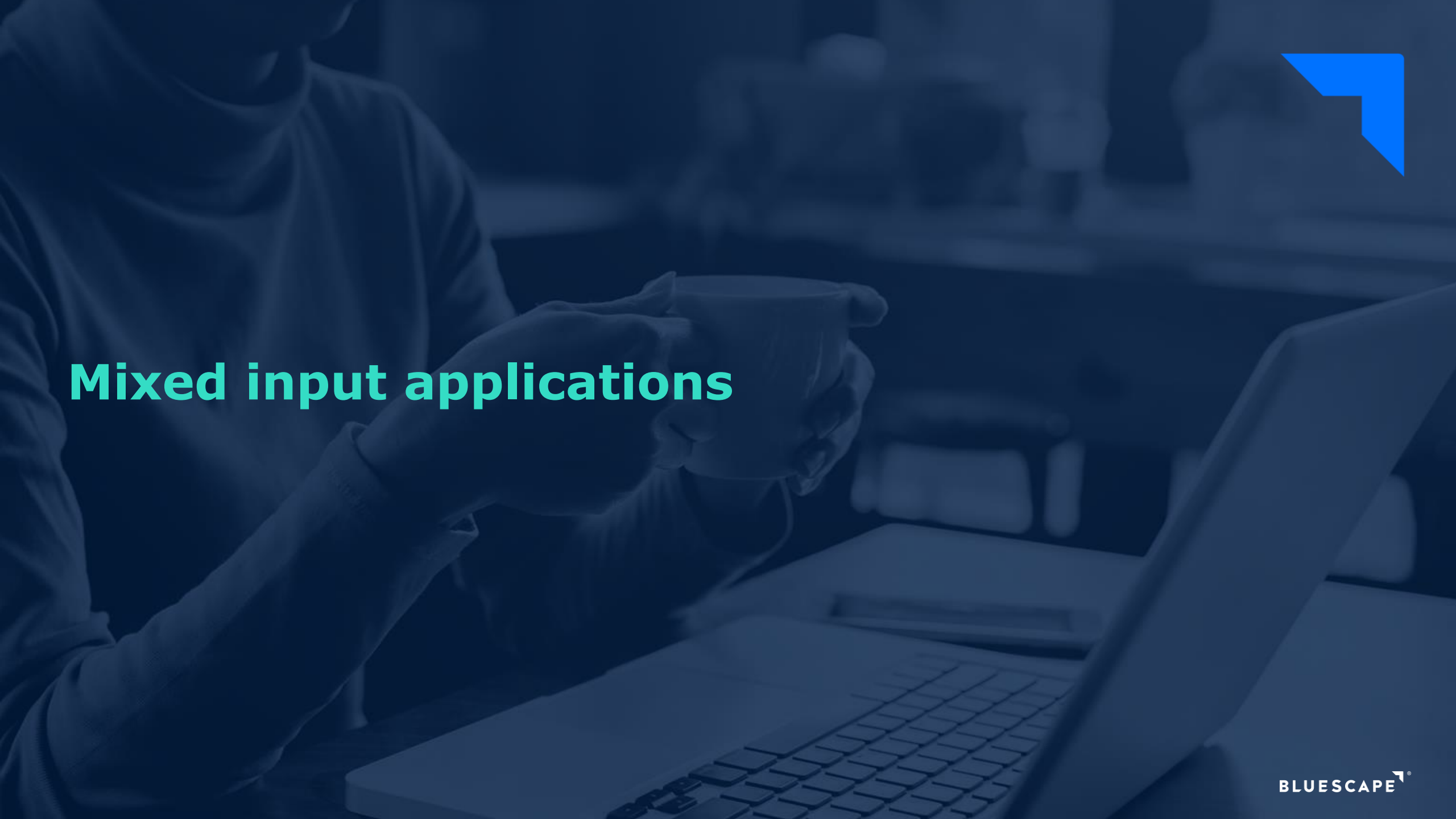
Off-the-shelf flick support for scrollable widgets

```
QScroller::grabGesture(scrollAreaWidget, QScroller::TouchGesture);

QScroller::scroller(textEdit)->scrollTo(QPointF(0, 100));

QScroller::scroller(textEdit)->ensureVisible(QRectF(0, 100, 200, 200), 0, 0);
```

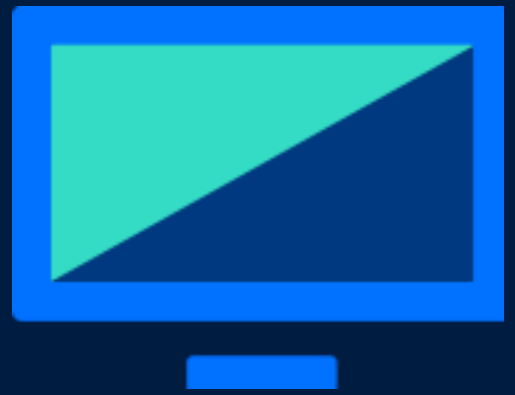
… but grabGesture(TouchGesture) should be avoided due to non-standard behavior!



Mixed input applications

BLUESCAPE®

Bluescape: A visual collaboration platform



Native app
(Windows/Linux/Mac)

Web browser

Mobile

Mixed input strategies

Mouse + Keyboard as primary input

- Touchpad as alternate mouse with 2D wheel scrolling

- Touch/Pen optional, with default as mouse cursor (single-touch only)

Finger + Pen as primary input

- Mouse as single finger, wheel as two-finger gestures (w/ keyboard modifiers)

- Touchpad as one-finger and two-finger gestures

- Pen as extra tool and/or as single finger

Caveat: Mouse wheel support

Mouse wheel behavior is inconsistent

- Sometimes used for zooming, sometimes for scrolling
- Most mice only support 1D scrolling
- Scrolling often implemented as mouse press+drag, but can conflict with content interactions

Touchpads behavior is consistent

- One-finger gestures map to the mouse cursor: move, press, release
- Two-finger gestures map to the mouse wheel: pan for 2D scrolling, pinch-zoom for zooming (w/ Ctrl keyboard modifier)
- Consistent on all OSes, and closely matches touchscreen behavior

Caveat: Mouse wheel support

Recommendation:

- Map the mouse wheel to **scrolling**, following the touchpad scrolling direction
- Use the **Shift key modifier** to switch scrolling direction (if applicable)
- Use the **Ctrl key modifier** to map the wheel to **zooming** (if applicable)

→ Keeps consistency with touchpad and finger/pen gestures

If a different wheel behavior is desired, give the user an option

Tips & tricks

BLUESCAPE®

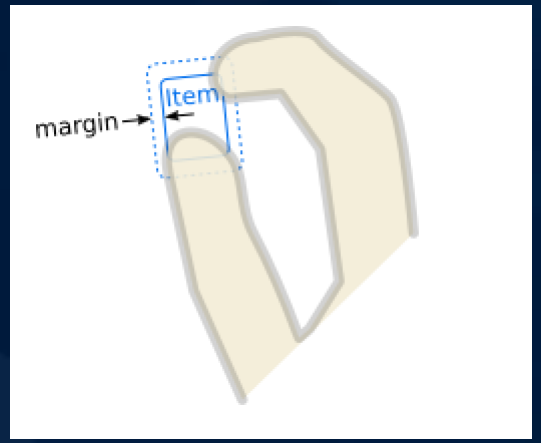
Tips & tricks

Pens make great mice too

- But careful about OS-generated mouse events if you need to handle both on a same window/widget!

Use extended tap/click areas for small QML items

- Easy with MouseArea, even easier with the new Pointer Handlers margin property



Use custom style/stylesheet for small widgets

- Not as flexible as margins, but a simple way to adjust the UI to a touch screen

Qt6 Input API changes

Qt6 Input API changes

New QPointerEvent base class

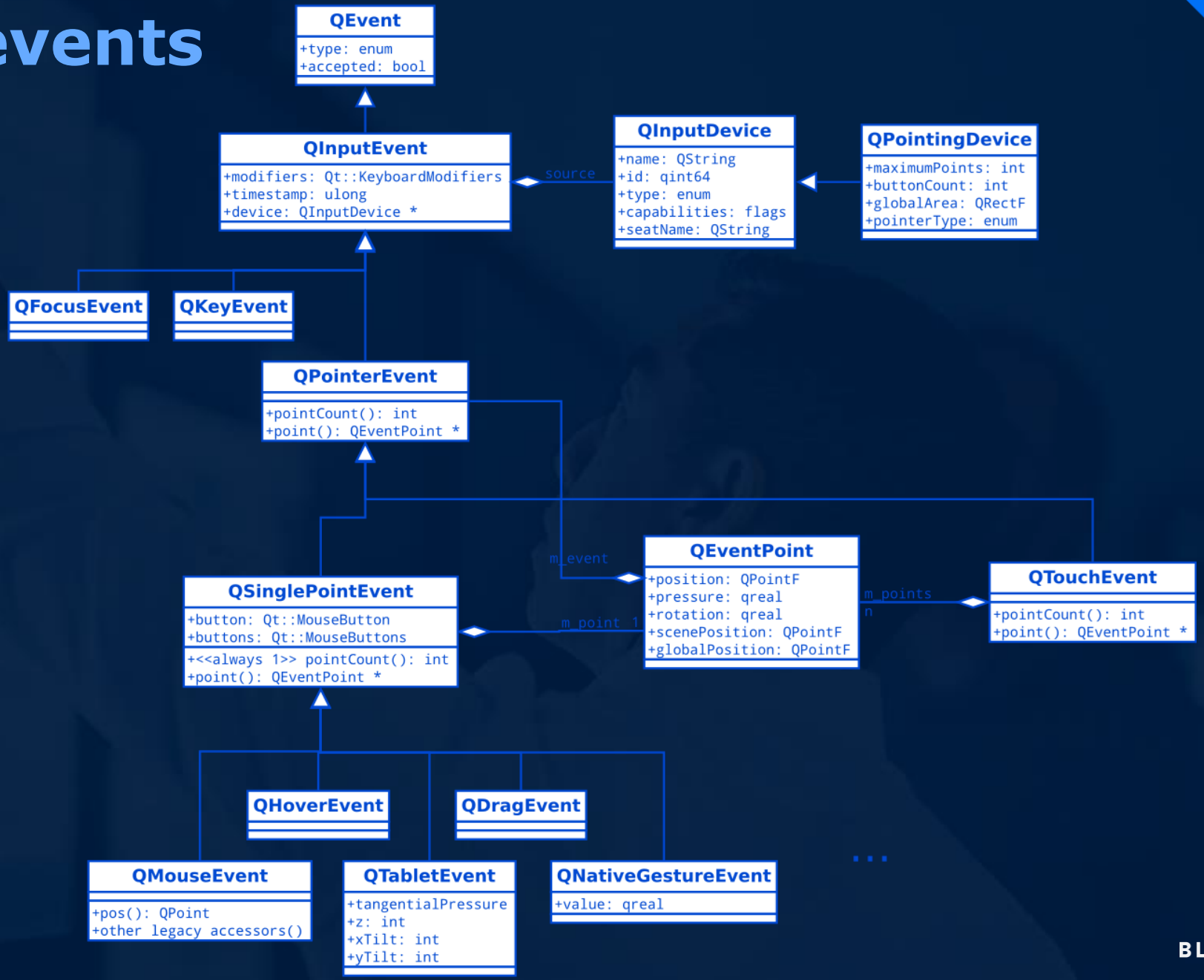
- C++ equivalent to the QML Pointer handlers (and same underlying events)

QTouchDevice replaced with QPointingDevice

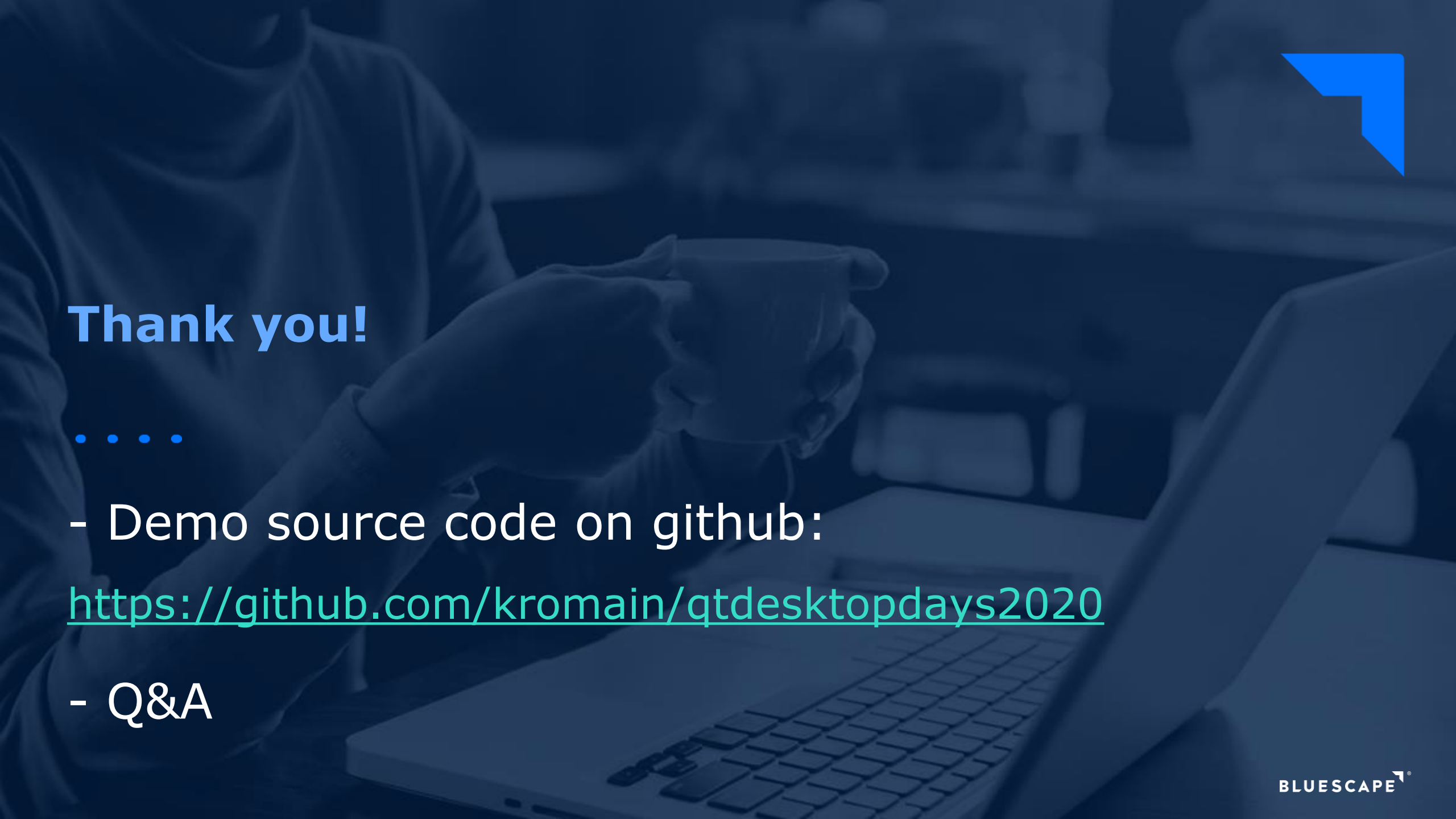
- Not limited to touch events anymore
- Proper multiple-device support and assignment in events

Event delivery cleanup

Qt6 Input events



BLUESCAPE



Thank you!

. . . .

- Demo source code on github:

https://github.com/kromain/qtdesktopdays2020

- Q&A

BLUESCAPE®