Qt Quick for Qt Developers

Training Course

Visit us at `http://qt.digia.com`

Produced by Digia Plc.

Material based on Qt 5.0, created on September 27, 2012

Digia Plc.

- Mouse Input
- Touch Input
- Keyboard Input

Objectives

- Knowledge of ways to receive user input
  - mouse/touch input
  - keyboard input
- Awareness of different mechanisms to process input
  - signal handlers
  - property bindings

Demo $QTDIR/qtdeclarative/examples/quick/touchinteraction/flickable/corkboards.qml

- Mouse Input
- Touch Input
- Keyboard Input

Mouse Areas

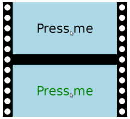
Mouse areas define parts of the screen where cursor input occurs

- Placed and resized like ordinary items
  - using anchors if necessary
- Two ways to monitor mouse input:
  - handle signals
  - dynamic property bindings

See QML MouseArea Element Reference Documentation

Clickable Mouse Area

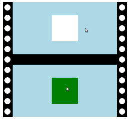
```
import QtQuick 2.0

Rectangle {
    width: 400; height: 200; color: "lightblue"

    Text {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
        text: "Press me"; font.pixelSize: 48

        MouseArea {
            anchors.fill: parent
            onPressed: parent.color = "green"
            onReleased: parent.color = "black"
        }
    }
}
```



Demo qml-user-interaction/ex-mouse-input/mouse-pressed-signals.qml

Mouse Hover and Properties

```
import QtQuick 2.0

Rectangle {
    width: 400; height: 200; color: "lightblue"

    Rectangle {
        x: 150; y: 50; width: 100; height: 100
        color: mouse_area.containsMouse ? "green" : "white"

        MouseArea {
            id: mouse_area
            anchors.fill: parent
            hoverEnabled: true
        }
    }
}
```



Demo qml-user-interaction/ex-mouse-input/hover-property.qml

Mouse Area Hints and Tips

- A mouse area only responds to its `acceptedButtons`
  - the handlers are not called for other buttons, but
  - any click involving an allowed button is reported
  - the `pressedButtons` property contains *all* buttons
  - even non-allowed buttons, if an allowed button is also pressed
- With `hoverEnabled` set to `false`
  - `containsMouse` can be `true` if the mouse area is clicked

Signals vs. Property Bindings

Which to use?

- Signals can be easier to use in some cases
  - when a signal only affects one other item
- Property bindings rely on named elements
  - many items can react to a change by referring to a property
- Use the most intuitive approach for the use case
- Favor simple assignments over complex scripts

- Mouse Input
- Touch Input
- Keyboard Input

Touch Events

- Single-touch (MouseArea)
- Multi-touch (MultiPointTouchArea)
- Gestures
  - Tap and Hold
  - Swipe
  - Pinch

Multi-Touch Events

```
MultiPointTouchArea {
  anchors.fill: parent
  touchPoints: [
      TouchPoint { id: point1 },
      TouchPoint { id: point2 },
      TouchPoint { id: point3 }
  ]
}
```

TouchPoint properties:

- int x
- int y
- bool pressed
- int pointId

MultiPointTouchArea signals

- **onPressed(list<TouchPoint> touchPoints)**
  onReleased(...)
  touchPoints is list of *changed* points.
- **onUpdated(...)**
  Called when points is updated (moved)
  touchPoints is list of *changed* points.
- **onTouchUpdated(...)**
  Called on *any* change
  touchPoints is list of *all* points.

MultiPointTouchArea signals

- **onGestureStarted(GestureEvent gesture)**
  Cancel the gesture using `gesture.cancel()`
- **onCanceled(list<TouchPoint> touchPoints)**
  Called when another element takes over touch handling.
  Useful for undoing what was done on onPressed.

Demo qml-user-interaction/ex-multi-touch/main.qml

Gestures

- Tap and Hold (MouseArea signal onPressAndHold)
- Swipe (ListView)
- Pinch (PinchArea)

Swipe Gestures

- Build in to ListView
- **snapMode: ListView.SnapOneItem**
  The view settles no more than one item away from the first visible item at the time the mouse button is released.
- **orientation: ListView.Horizontal**

Demo $QTDIR/qtdeclarative/examples/quick/touchinteraction/flickable/corkboards.qml

Pinch Gesture

Automatic pinch setup using the target property:

```
Image {
    id: image
    source: "qt-logo.jpg"

    PinchArea {
        anchors.fill: parent
        pinch.target: parent

        pinch.minimumScale: 0.5
        pinch.maximumScale: 2.0

        pinch.minimumRotation: -3600
        pinch.maximumRotation: 3600

        pinch.dragAxis: Pinch.XAxis
    }
}
```

Demo qml-user-interaction/ex-pinch

Pinch Gestures

- Signals for manual pinch handling
  - onPinchStarted(PinchEvent pinch)
  - onPinchUpdated(PinchEvent pinch)
  - onPinchFinished()

- PinchEvent properties:
  - point1, point2, center
  - rotation
  - scale
  - accepted
    set to false in the onPinchStarted handler
    if the gesture should not be handled

- Mouse Input
- Touch Input
- Keyboard Input

Keyboard Input

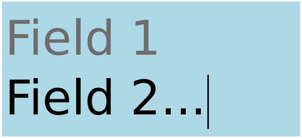
Basic keyboard input is handled in two different use cases:

- Accepting text input
  - `TextInput` and `TextEdit`
- Navigation between elements
  - changing the focused element
  - directional (arrow keys), tab and backtab

On page 28 we will see how to handle raw keyboard input.

Assigning Focus

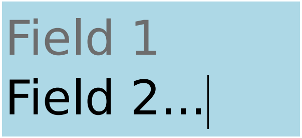
- UIs with just one `TextInput`
  - focus assigned automatically
- More than one `TextInput`
  - need to change focus by clicking
- What happens if a `TextInput` has no text?
  - no way to click on it
  - unless it has a `width` or uses anchors
- Set the `focus` property to assign focus



Using TextInputs

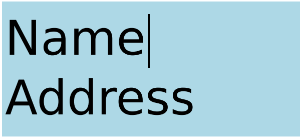
```
import QtQuick 2.0

Rectangle {
    width: 200; height: 112; color: "lightblue"

    TextInput {
        anchors.left: parent.left; y: 16
        anchors.right: parent.right
        text: "Field 1"; font.pixelSize: 32
        color: focus ? "black" : "gray"
        focus: true
    }
    TextInput {
        anchors.left: parent.left; y: 64
        anchors.right: parent.right
        text: "Field 2"; font.pixelSize: 32
        color: focus ? "black" : "gray"
    }
}
```

Demo qml-user-interaction/ex-key-input/textinputs.qml


Field 1
Field 2...

Focus Navigation

```
TextInput {
    id: name_field
    ...
    focus: true
    KeyNavigation.tab: address_field
}
TextInput {
    id: address_field
    ...
    KeyNavigation.backtab: name_field
}
```

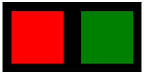


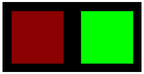
- The `name_field` item defines `KeyNavigation.tab`
  - pressing **Tab** moves focus to the `address_field` item
- The `address_field` item defines `KeyNavigation.backtab`
  - pressing **Shift+Tab** moves focus to the `name_field` item

Demo qml-user-interaction/ex-key-input/tab-navigation.qml

Key Navigation

```
import QtQuick 2.0

Rectangle {
    width: 400; height: 200; color: "black"

    Rectangle { id: left_rect
                x: 25; y: 25; width: 150; height: 150
                color: focus ? "red" : "darkred"
                KeyNavigation.right: right_rect
                focus: true }

    Rectangle { id: right_rect
                x: 225; y: 25; width: 150; height: 150
                color: focus ? "#00ff00" : "green"
                KeyNavigation.left: left_rect }
}
```





- Using cursor keys with non-text items
- Non-text items can have focus, too

Demo qml-user-interaction/ex-key-input/key-navigation.qml

Mouse and cursor input handling:

- `MouseArea` receives clicks and other events
- Use anchors to fill objects and make them clickable
- Respond to user input:
  - give the area a name and refer to its properties, or
  - use handlers in the area and change other named items

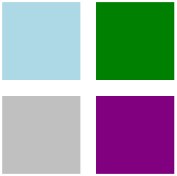
Key handling:

- `TextInput` and `TextEdit` provide text entry features
- Set the `focus` property to start receiving key input
- Use anchors to make items clickable
  - lets the user set the focus
- `KeyNavigation` defines relationships between items
  - enables focus to be moved
  - using cursor keys, tab and backtab
  - works with non-text-input items

Exercise – User Input

- Which element is used to receive mouse clicks?
- Name two ways `TextInput` can obtain the input focus?
- How do you define keyboard navigation between items?

Lab – Menu Screen



- Using the partial solution as a starting point, create a user interface similar to the one shown above with these features:
  - items that change color when they have the focus
  - clicking an item gives it the focus
  - the current focus can be moved using the cursor keys

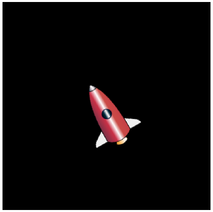
Lab qml-user-interaction/lab-menu-screen

Raw Keyboard Input

- Raw key input can be handled by items
  - with predefined handlers for commonly used keys
  - full key event information is also available
- The same focus mechanism is used as for ordinary text input
  - enabled by setting the `focus` property
- Key handling is not an inherited property of items
  - enabled using the `Keys` attached property
- Key events can be forwarded to other objects
  - enabled using the `Keys.forwardTo` attached property
  - accepts a list of objects

Raw Keyboard Input

```
import QtQuick 2.0

Rectangle {
    width: 400; height: 400; color: "black"

    Image {
        id: rocket
        x: 150; y: 150
        source: "../images/rocket.svg"
        transformOrigin: Item.Center
    }

    Keys.onLeftPressed:
        rocket.rotation = (rocket.rotation - 10) % 360
    Keys.onRightPressed:
        rocket.rotation = (rocket.rotation + 10) % 360

    focus: true
}
```



Raw Keyboard Input

- Can use predefined handlers for arrow keys:

```
Keys.onLeftPressed:
    rocket.rotation = (rocket.rotation - 10) % 360
Keys.onRightPressed:

    rocket.rotation = (rocket.rotation + 10) % 360
```

- Or inspect events from all key presses:

```
Keys.onPressed: {
    if (event.key == Qt.Key_Left)
        rocket.rotation = (rocket.rotation - 10) % 360;
    else if (event.key == Qt.Key_Right)
        rocket.rotation = (rocket.rotation + 10) % 360;

}
```

Focus Scopes

- Focus scopes are used to manage focus for items
- `FocusScope` delegates focus to one of its children
- When the focus scope loses focus
  - remembers which one has the focus
- When the focus scope gains focus again
  - restores focus to the previously active item