Fluid UI Creation with Qt Quick Controls

Tino Pyssysalo
Senior Manager, Product Management
The Qt Company

Practicalities

- Trainer Tino Pyssysalo
  - Based in Finland, Nokia
  - Qt guy since 2007
- Timing
  - 10.00-13.00
  - Coffee break @11.00-11.30
- Materials available in https://github.com/tpyssysa/QtDay.git

Agenda

- Introduction
- Application Window and Views
- Controls, Containers, Layouts
- Themes and Styles
- QQC1 vs. QQC2

2

Qt Quick Controls

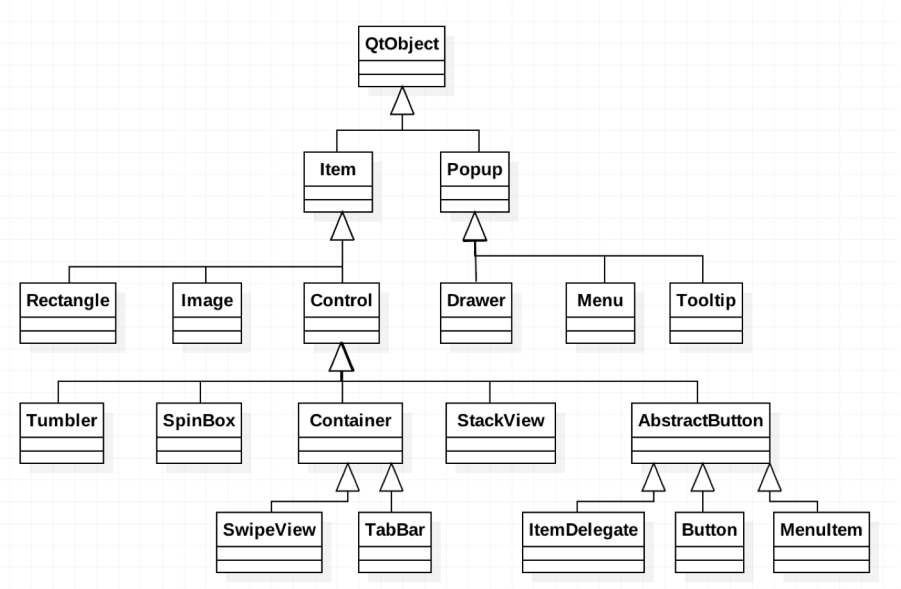
Qt day

- Ready-made UI controls, custom items
  - Several modules and versions
  - QtQuick.Controls 2.3
  - QtQuick.Templates 2.3
  - QtQuick.Controls.<Style name>
  - (QtQuick.Controls 1.4)
  - QtQuick.Extras 1.4
    - Gauge, CircularGauge, PieMenu, DelayButton
- Qt.labs.calendar
- Qt.labs.platform
  - Menus, dialogs, system tray icon
- Layouts
  - QtQuick.Layouts 2.1
  - Dynamically shrink/expand items in the layout

4

SDK demo: Qt Quick Controls 2 Gallery
Qt Quick Extras Gallery

Class Hierarchy



5

Application Window

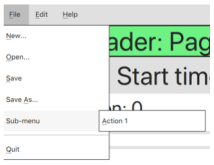
- Header and footer items
- Menubar
  - With menus, sub-menus, separators, and menu items / actions
- Content
  - Window children, children of `contentItem`
  - For example, a view, a container, a control
- Background
  - Any item assigned to `background`
- Overlay
  - Popups, like menus and dialogs
- Window-specific palette
- Locale aware

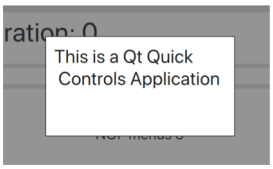
```
ApplicationWindow {
    visible: true; width: 640; height: 480;
    title: qsTr("Application Window")
    minimumHeight: 300; minimumWidth: 330

    header: Label {
        height: 60
        text: qsTr("Header")
    }
    footer: Button {
        text: qsTr("Button")
        onClicked: console.log("Button pressed")
    }
    menuBar: MenuBar {}

    Page {
        title: qsTr("Header: Page title")
        anchors.fill: parent
    }

    background: Rectangle {
        anchors.fill: parent
        color: "lightgreen"
    }
    Popup {} id: popup
}
```

Demo: applicationWindow

6

Popups





- Hierarchy of menus assigned to the `menuBar` property
  - Menus contain buttons (`text`, `icon`, `checkable`, `action`)
  - `MenuItem` is a specialized button with a possible sub-menu and `arrow` item
  - `Actions` available since 5.10
    - Allows defining the key sequence CTRL-L
- Dialogs
  - Modal or non-modal `Popup`
  - Modal popup dims the window by default
  - Note `FileDialog`, `FontDialog` etc. are not part of Qt Quick Controls

```
menuBar: MenuBar {
    id: menuBar
    Menu {
    title: qsTr("&File")
    cascade: true
    Action { text: qsTr("&New...") }
    Menu {
        title: qsTr("Sub-menu")
        overlap: 20.0
        Action { text: qsTr("&Action 1") }
    }
    MenuSeparator { }
}
Popup {
    id: popup
    width: parent.width * 0.4; height: width
    x: (parent.width - width) / 2; y: x
    modal: true
    closePolicy: Popup.CloseOnEscape |
        Popup.CloseOnPressOutsideParent
    Label {
        anchors { fill: parent
        text: qsTr("This is Qt Quick Controls"}
    }
```

7

Palette

- Basic QML type `palette`
- Defines color of various roles
  - `palette.base`
  - `palette.brightText`
  - `palette.text`
- Application-wide
  - Maintained by the application window
  - Can be set in `QGuiApplication`
  - Propagated to all window children
- Can be customized in controls
  - All control children will use the customized palette

```
ApplicationWindow {
    id: window visible
    // All child controls will have
    // blue foreground text
    palette.text: "blue"
```

8

Window Content

- Anything – window does not specialize its content
  - Good practice
  - Used everywhere in Quick Controls 2
- Views, containers, controls, any items can be used
- Views provide the basic layout and navigation
  - `StackView`, `SwipeView`
- Containers
  - `SwipeView`, `TabBar`

Views – StackView

- Allows user to push, pop, and replace pages in the stack
- Only the top-most item visible
- Several pages may be pushed in one function call, only the topmost created
- Custom animations may be defined for view transitions

```
StackView {
    id: stackView
    initialItem: page
    pushEnter: Transition {
        ParallelAnimation {
            RotationAnimation { from: 0; to: 360 }
            NumberAnimation { properties: "opacity"; from: 0.0; to: 1.0;
                              easing.type: Easing.InOutQuad }
    }
}
Component {
    id: page
```


Demo: stackView

Views – SwipeView

- Swipe trigged page navigation – horizontal or vertical
- Pages may be dynamically added and removed
- As extends `Container`
- Page indicator helps user to see there are multiple pages
- Another control added by the developer


```
SwipeView {
    id: swipeView; anchors.fill: parent
    currentIndex: tabBar.currentIndex
    Page { Label { text: qsTr("Page"); anchors.centerIn: parent } } }
    PageIndicator { id: indicator
        count: swipeView.count; currentIndex: swipeView.currentIndex
        anchors.bottom: swipeView.bottom;
        anchors.horizontalCenter: parent.horizontalCenter
    }
}
```

11

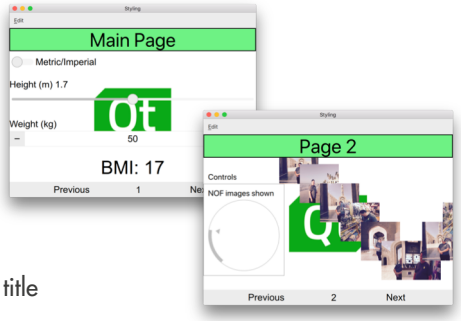
Demo: swipeView

Summary

- Qt Quick Controls provide ready-made UI controls
- Delivered in several modules and versions
  - QQC2 is suitable for all platforms
  - QQC1 is not optimal for mobile and embedded
- Application window extends the window with menus, header, footer, and popups
  - Supports also locale and application-wide palette
- Window content can be completely specific by the developer
  - View, list, control, custom item, container, another window

12

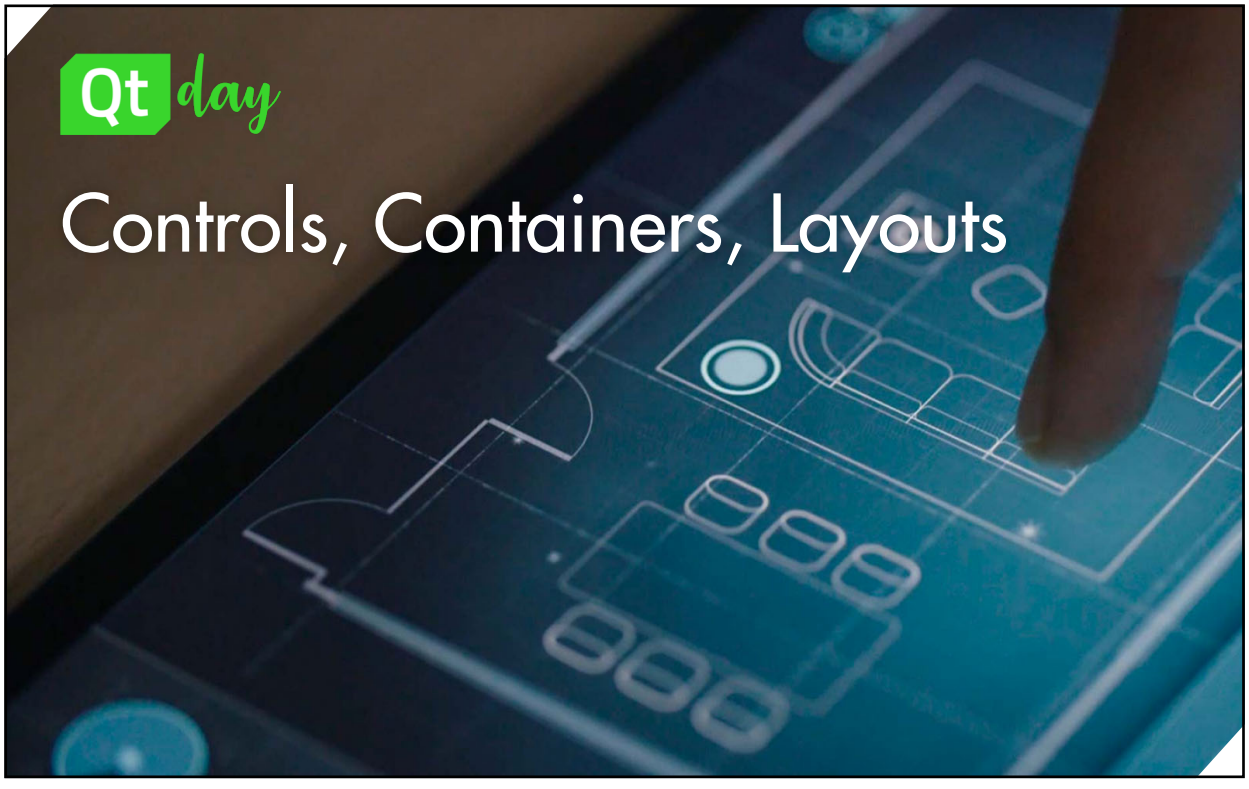
Lab 1 – Application Window



Let's implement a simple application with Qt Quick Controls
- The first page is a BMI (body-mass index) calculator
- The second page shows some images in the V-shape path

1. Create a stack view -based QML application in Qt Creator
2. Application window header is a simple label, showing the page title
   - Each page actually extends the Page tQML type
   - No need to touch the footer yet
3. Add a MessageDialog to give notifications to the user
4. Add a file dialog and a ListModel to pick-up and choose image file urls
   - Add the selected files to the list model `imageModel.append({"anyPropertyName": fileUrls[index]});`
5. Add at least one menu item to open the file dialog
6. If time permits, customize the stack view transitions

13



Best Practices – General Purpose Components

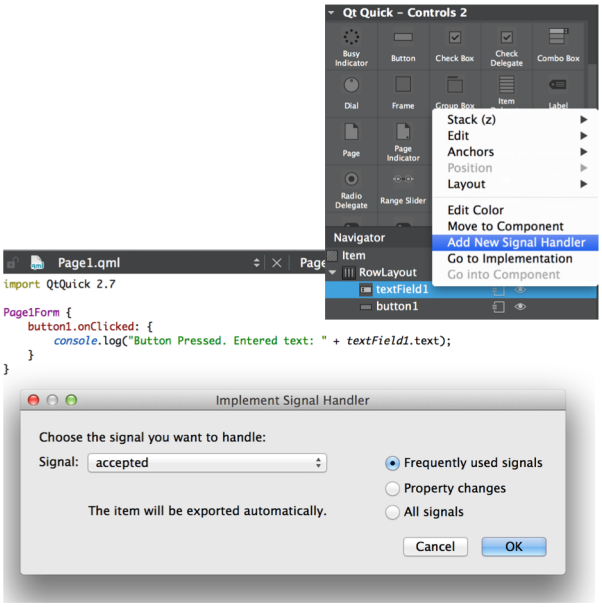
- Possible to refer to a window and its children with an `id`
- Avoid creating dependencies like this in components
- Prefer using window attached properties
  - E.g. `ApplicationWindow.menuBar` vs. `id menuBar`

```
import QtQuick 2.4
import QtQuick.Controls 2.3

Item {
    id: item1
    Label {
        id: label
        text: qsTr("NOF menus ") + ApplicationWindow.menuBar.menus.length
        font.pointSize: 20
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: parent.verticalCenter
     }
```

15

UI Component Design with Qt Quick Designer

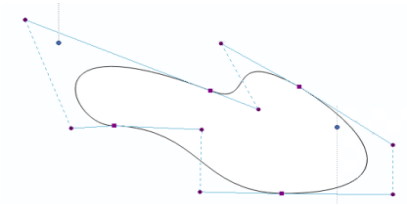
- Create UI items
  - Form – `MyComponent.ui.qml`
  - Component – `MyComponent.qml`
- Drag'n' drop controls and other items
- Add layouts
- Define item properties
  - Using the text editor, if necessary
- Add states, if needed
  - Each state defines different property values for one or more items
  - Transitions animate property value changes
- Implement the business logic into the corresponding component

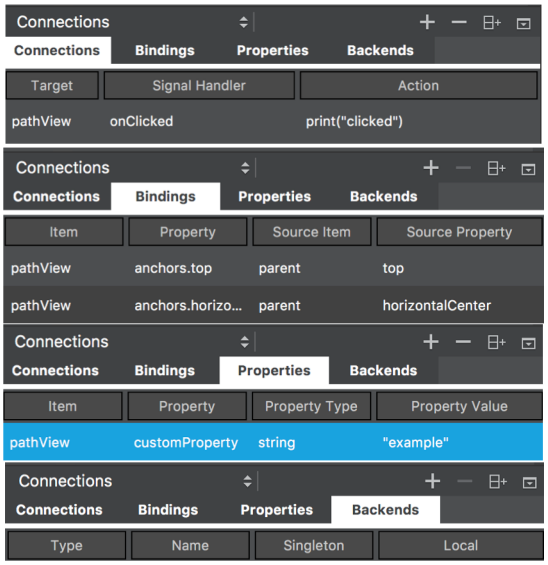


16

Qt Quick Designer Editors

- Drag'n'Drop Editor
- Connections Editor
- Bindings Editor
- Property Editor
- Backend Editor
- Spline Editor for PathView



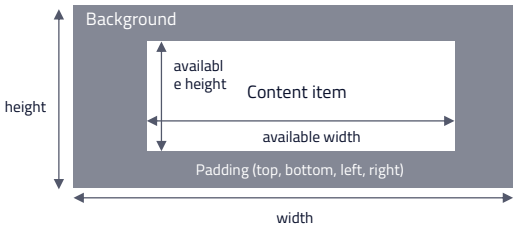


17

Controls

- Extend `Control`
- Content not specialized
  - Property `contentItem`
- Background item
- Padding dimensions
- Layout
  - Control's implicit size is the background's implicit size by default
- Palette
- Focus
  - Any item may request to get active focus (property `focus`)

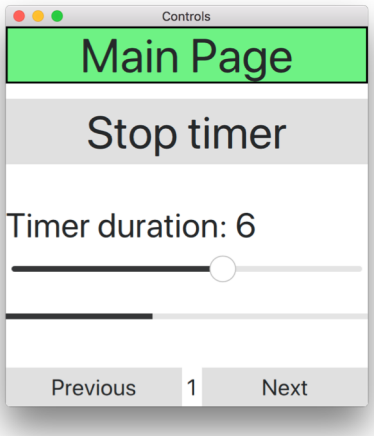
- Event handling
  - In C++, no area objects needed
  - Interactive controls do not pass mouse click and touch events to the item below them
- Locale aware



18

Control Examples

- `Button`
  - Can be clicked or touched
  - Properties: action with shortcuts, autoExclusive, display (icon, text or both)
  - Multi-touch support, like in `Slider`
- `Slider`
  - Used to select a value by sliding a handle
  - Properties: `from`, `to`, `position`, `value`, `handle`, `snapMode`
  - Live value update (default), like in `RangeSlider` and `Dial`
- `ComboBox`
  - Combined button and popup list for selection options
  - Can be editable

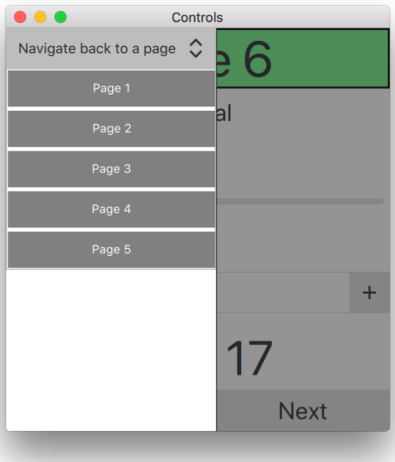


19

Demo: controls
https://doc.qt.io/qt-5.10/qtquickcontrols2-guidelines.html

Control Examples

- `Page`
  - Simple control with header and footer
  - Title
- `Drawer`
  - Side panel with any content
  - Can be positioned into any window edge
  - Open and closed by swiping
- `ItemDelegate`
  - Convenient delegate for list views, combo boxes, menus
  - Check, radio, switch, swipe delegates



20

Demo: controls

Container Controls

- Support adding, inserting, moving, and removing items
  - Additional properties: `currentIndex` and `currentItem`
- No visual presentation
  - Defined by the `contentItem` property
- Container items are defined using `contentModel` default property
  - All children are assigned to `contentModel`
- `Page` is another container (not `Container` sub-type) having a header and a footer

```
Container {
    id: container
    contentItem: ListView {
        model: container.contentModel
    }
    Image { source: "qrc:/images/page1_image" }
    Image { source: "qrc:/images/page2_image" }
```

21

Dynamic Management of Container Items

```
footer: TabBar {
    id: tabBar
    currentIndex: container.currentIndex
    TabButton {
        text: qsTr("+")
        onClicked: tabBar.addItem(tabButton.createObject(tabBar));
    }
    Component {
        id: tabButton
        TabButton {
            text: qsTr("I'm removed by clicking")
            onClicked: tabBar.removeItem(tabBar.currentIndex);
        }
    }
}
```

22

Demo: applicationWindow

Qt Quick Layouts

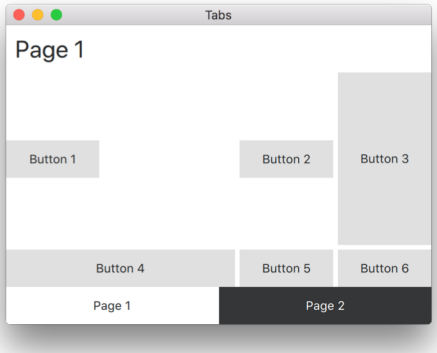
- `RowLayout, ColumnLayout, GridLayout`
- Default behavior is similar to basic positioners
  - `Row, Column, Grid, Flow`
- Possible to let the layout automatically determine the item dimensions
  - No anchors or explicit width/height needed
- Set the `Layout.fillHeight` or `Layout.fillWidth` attached properties to
  - `false` – if you do not want the layout to use all extra space for the item
  - `true` – if you want the extra space to be used to expand the item

23

Layouts Example

- Two buttons expand vertically and horizontally
- Other button dimensions are based on the text and font properties

```
GridLayout {
    columns: 3
    …
    Button {
        text: qsTr("Btn 2")
        Layout.fillHeight: true
        …
    }
    Button {
        text: qsTr("Btn 4")
        Layout.fillWidth: true
        …
    }
```



24

Demo: layouts

Summary

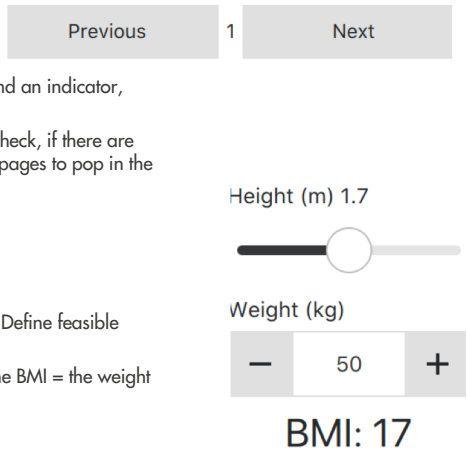
- Create generic UI components
  - Avoid designing too specialized controls for each use case
  - Avoid making dependencies to root items, such as window
- Controls
  - May have background, padding, and content
  - Background implicit size defines control's implicit size
  - May request for the focus like any item
  - May have their own palette
- Content item should be scalable
  - Qt Quick Layouts often used to make contentItem children shrinkable/expandable

25

Lab 2 – Controls (1/2)

Let's add the footer and some pages to the application

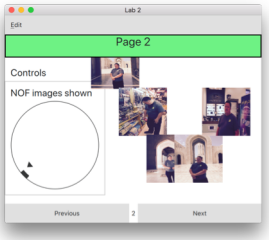
1. Add a footer to the application window
   - It should contain two buttons to navigate to the previous and next pages and an indicator, showing the stack view depth
   - Pop a page from the stack view, if the user clicks on the previous button. Check, if there are any pages to pop and notify the user with MessageDialog, if there are no pages to pop in the stack
   - Similarly, push a page (either of the Page1/Page2) to the stack
2. BMI page (Implement e.g. to HomeForm)
   - Drag'n'drop and layout the UI using Quick Controls (e.g. slider, spinbox). Define feasible ranges, values etc.
   - Use binding to change the BMI value, when the height/weight changes. The BMI = the weight / the square of the height



26

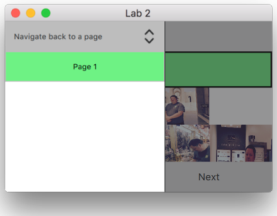
Lab 2 – Controls (2/2)

3. Image View page (Implement to the page you pushed into the stackview in Item 1)
   - Lay out a group box and path view into a row
   - Add a label and dial to the group box
   - The path view model is the model, you created in Lab 1
   - The path view delegate may be simply an image with a hard-coded size, e.g. 100x100
   - The dial value should determine the number of items shown in the path view
   - To have a V shape path in path view, you may use something like




```
path: Path {
    startX: pathView.width * 0.05; startY: pathView.height * 0.05
    PathLine { x: pathView.width / 2; y: pathView.height * 0.85 }
    PathLine { x: pathView.width * 0.95; y: pathView.height * 0.05 } }
```

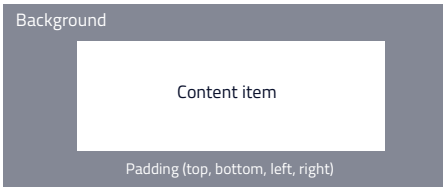
4. If time permits, change the drawer to contain a ComboBox, which lists all the pages
   - Selecting a combo box item, should result the UI to navigate to that page
   - You need to have a model with page names or integers
   - ComboBox delegate may be a simple framed text



27

Styling Qt Quick Controls

- Cross-platform icon themes
- Stylable control properties
  - `background`
  - `contentItem`
- Available styles
- Configuration file
  - Define, which style and which style-specific property values used
- Qt Quick templates
  - Stylable control-specific properties



29

Icon Themes

- Buttons, item delegates, and menu items support icons
- Default icon size and color defined in a style
  - Can be overridden with icon properties

```
icon.name: "File-open"; icon.source: "open.png"
```

- Icon themes allow using the same icons everywhere in the application
  - Set the theme name before loading the QML file

```
QIcon::setThemeName("clusterTheme");
```

```
[Icon Theme]
Name=clusterTheme
Comment=Cluester Icon Theme
Directories=32x32,32x32@2
[32x32]
Size=32
Type=Fixed
[32x32@2]
Size=32
```

```
<RCC>
  <qresource prefix="/">
    <file>icons/desktopTheme/index.theme</file>
    <file>icons/desktopTheme/32x32/open.png</file>
    <file>icons/desktopTheme/32x32@2/open.png</file>
  </qresource>
</RCC>
```

30

Available Styles

Define attached properties, used in controls styling

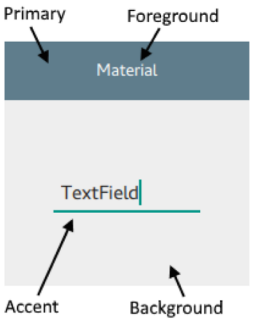
- Default style
  - Light-weight default style
  - Used as a fallback, if the style does not implement a control
- Imaging style
  - Property `path` defines the image assets location from PS
  - Images must follow the imaging naming convention
- Material and Universal styles
  - Google Material and MS Universal Design Guidelines –based styles
  - Properties: accent, background, foreground, theme, primary (only in Material style)
- Fusion style
  - Platform-agnostic style, providing desktop look'n'feel

31

Using Styles

- Refer directly to style-specific attached properties
- Recommended to use style-specific colors

```
import QtQuick.Controls.Material 2.0

Button {
    text: "Stop"; highlighted: true
    Material.accent: Material.Red
    Material.theme: Material.Dark
}
```



32

Configuration File

- Deployed in resources similarly to QML files
- Configure style and style properties
- Configure default font properties
- Configure palette

```
[Controls]
Style=Custom

[Universal]
Theme=Dark

Font\Family=Courier New
Font\PointSize=24
Font\Style=StyleItalic

[Custom]
Palette\Text=#abcdef
```

33

Deploying Platform, Locale, and Style Variants

- Built-in support for selecting different variants of QML files
  - Based on the file selectors
- Platform

  `:/CustomControl.qml`

  `:/+linux/CustomControl.qml`
- Locale

  `:/+fi_FI/CustomControl.qml`
- Style

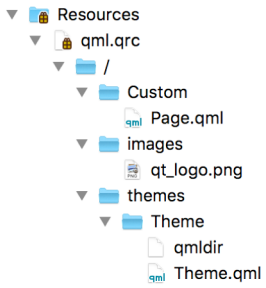
  `:/+custom/CustomControl.qml`
- Variants may be combined as well

  `:/+custom/+fi_FI/CustomControl.qml`

34

Qt Quick Templates

- Non-visual implementation of control's logic and behavior
- Style agnostic
- Style properties defined in `$QTDIR/qml/QtQuick/Controls.2/ControlType.qml`
  - Implicit dimension, `padding`, `spacing`, icon properties, `contentItem`, `background`

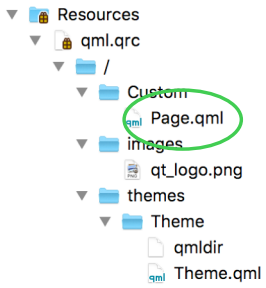


- To create a custom style
  1. Create a style folder
  2. Copy controls, you want to style, from `$QTDIR/qml/QtQuick/Controls.2`
  3. Apply existing style property values, if applicable
  4. Implement customizations
  5. Apply the style using the configuration file, a command line switch `-style`, an environment variable or set with `QQuickStyle::setStyle("CustomStyle")`

35

Custom Page Control

```
import QtQuick 2.9
import QtQuick.Templates 2.2 as T
import QtQuick.Controls.Universal 2.2
import Theme 1.0

T.Page {
    id: control
    property alias backgroundVisible: backgroundImg.visible

    background: Rectangle {
        color: control.Universal.background
        Image {
            id: backgroundImg
            anchors.centerIn: parent
            // All stylable properties in the same file
            source: Theme.backgroundImage
        }
    }
}
```



36

Demo: styling

Theme

- Custom set of stylable properties as `QtObject` children
- Define as singleton
- Typically deployed as a module
  - `module Theme`
  - `singleton Theme 1.0 Theme.qml`

```
pragma Singleton
import QtQuick 2.9
QtObject {
    // Window
    readonly property int windowWidth: 640
    readonly property int windowHeight: 480
    readonly property int windowMinWidth: 380
    readonly property int windowMinHeight: 400
    // Font
    property font defaultFont
    defaultFont.pointSize: 24
```

37

Summary

Controls may be styled using

- Custom icon themes
- Palette
- Stylable properties, such as palette
- Existing styles/themes and style properties
- Configuring styles in the configuration file
- Templates to create a custom style

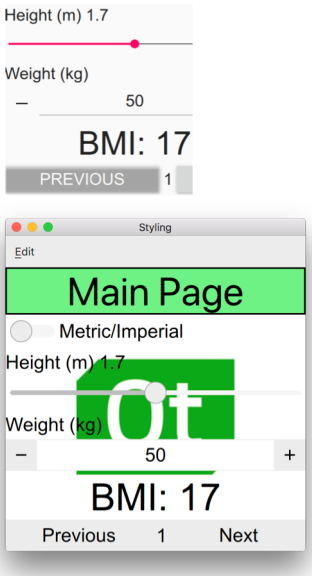
38

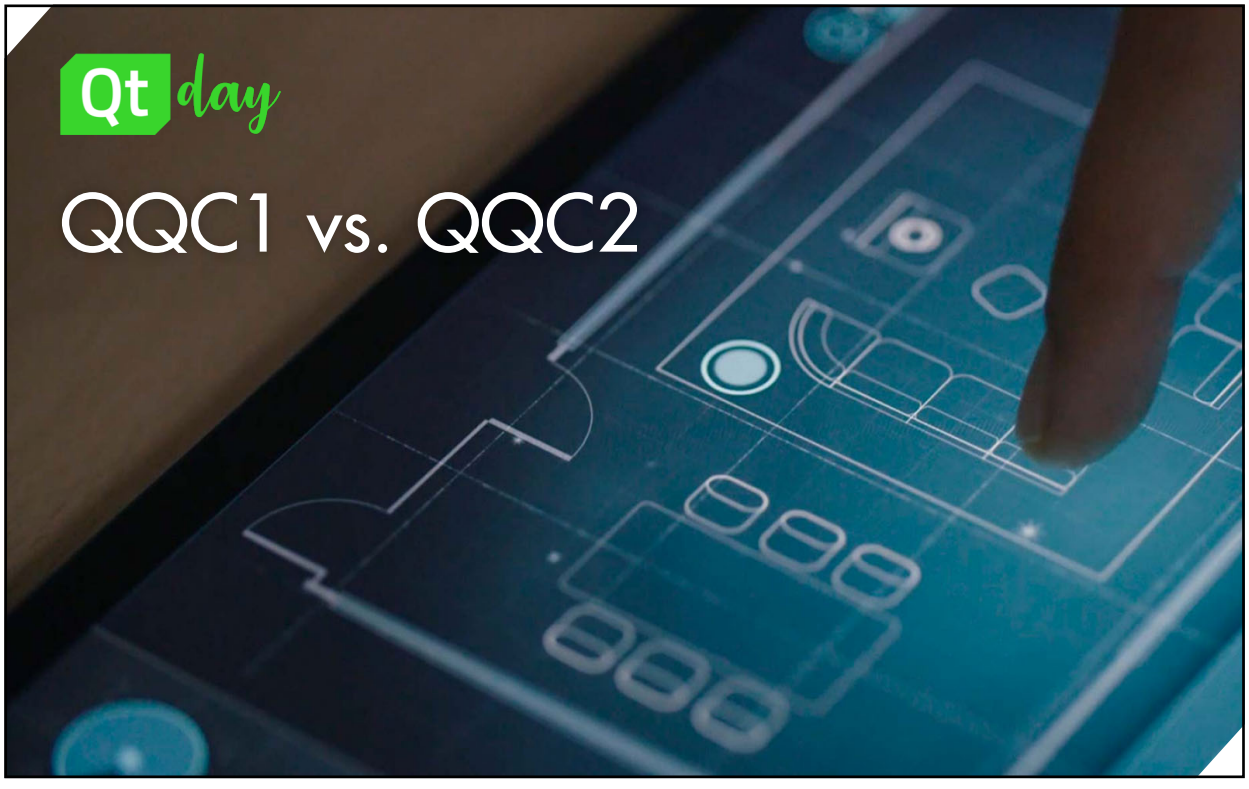
Lab 3 – Styling

1. Change the application style in the qtquickcontrols2.conf file
2. Change style properties for some of your controls (Material.accent, Material.theme)
3. Create a custom control, based on some of the controls you used (e.g. a page)
   - Add, e.g. an image to the control background by default
   - Add a property to the control to determine, whether the background is visible or not
   - Use your custom style (use qtquickcontrols2.conf)

3. If time permits, put all magic numbers to a custom Theme singleton and use it to replace magic numbers

39





Differences

- Implemented in QML
- Extending existing QML types
  - `Button` -> `FocusScope`
- Specialization
  - "Too" specialized
  - Challenging to customize
- Control-specific, platform-dependent style objects
  - Dynamically allocated
  - Native styles

- Logic in C++, UI in QML
  - UI and template separation
- Extends `QQuickItem`
- Specialization
  - More general
  - Property `contentItem`
- Styling with application-global, platform agnostic style
  - Style is part of the control
  - Style guidelines

41

Differences

- Pointer handling with `Area` objects
- Key handling with `Keys`
- Mandatory QML objects
  - Accessibility framework objects

- Event handling in C++

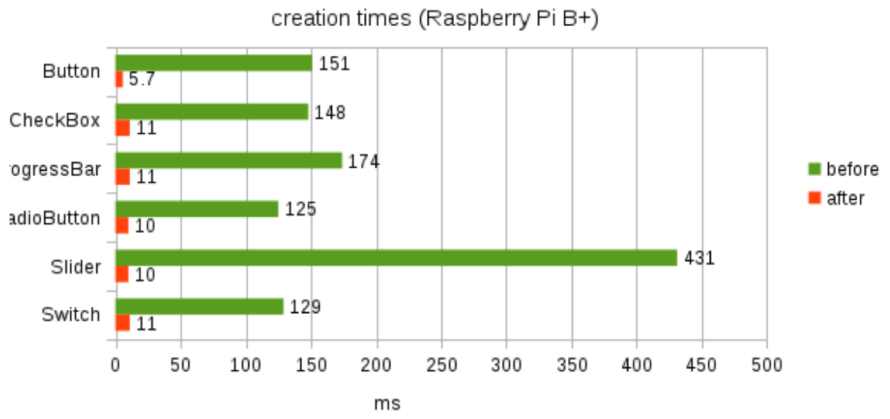
- No mandatory QML objects

- High-DPI support
- Lightweight, more modular controls
  - Increases re-usability
  - `ScrollView` -> `ScrollBar`, `ScrollIndicator`

42

Memory Consumption and Performance

- QQC1 Button allocates 60 `QObjects`, including 4 `Loaders` and 15 `QQuickItems`
- QQC2 Button allocates 7 `QObjects`, including 4 `QQuickItems`



43

QQC1 vs. QQC2 Comparison Table

Calendar	Qt.labs.calendar
ExclusiveGroup	ActionGroup, ButtonGroup
SplitView	NA
StatusBar	ApplicationWindow header/footer
TabView	TabBar
TableView	NA
TreeView	NA
Native dialogs (color, file, font)	Qt.labs.platform
NA	Control, Container, Drawer, Page, Pane, Popup, Delegates, ToolTip

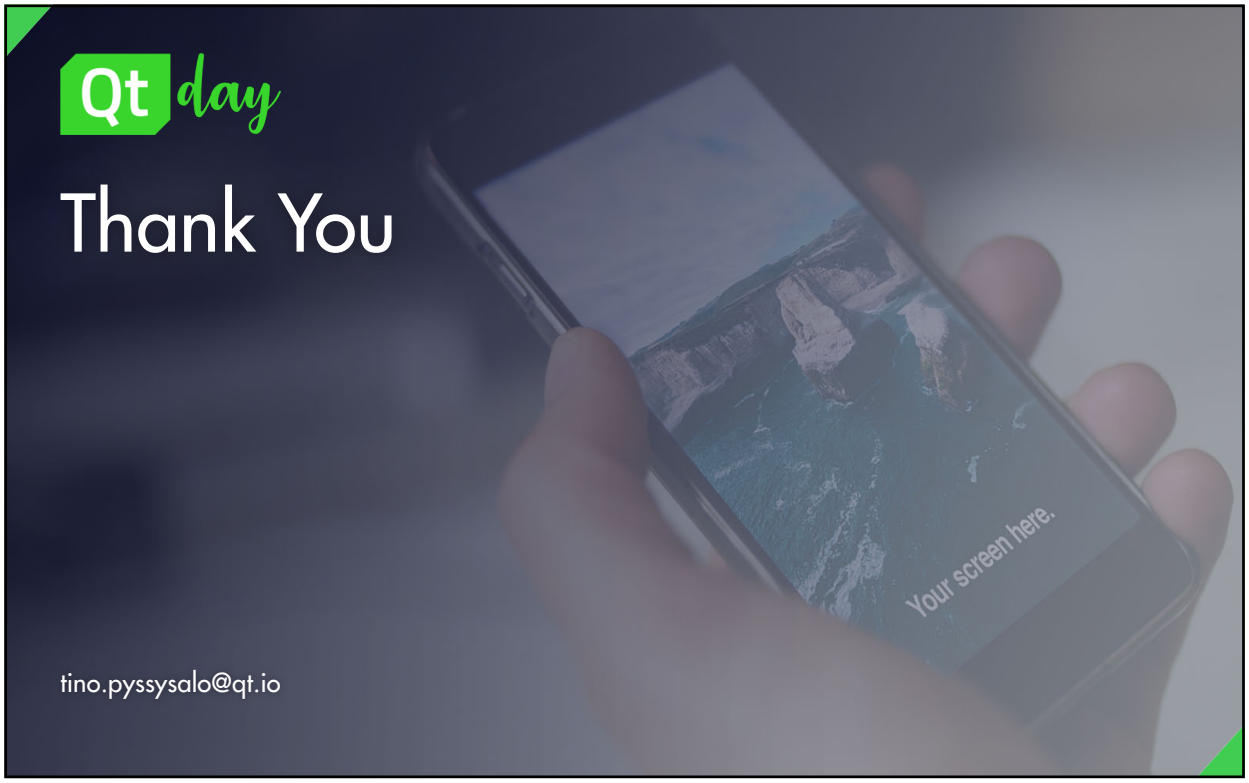
44

Summary

- QQC provide ready-made UI building blocks
  - Application window, page, container, views, actual controls
- QQC2 was created to improve QQC1 memory consumption and performance
  - Can be mixed if needed
  - E.g. there is no TableView or TreeView in QQC2
  - Prefer using QQC2 anyway
- Controls provide good examples how to create custom components
  - Logic in C++, UI in QML
  - No specialized content
  - No style objects in each and every control object

45

Thank You



tino.pyssysalo@qt.io