Développement mobile avec Qt QML

Meetup #5 : Qt Labs controls

Guillaume Charbonnier
gcharbonnier@a-team.fr

Développement mobile avec Qt/QML
La Cantine Numérique le 30 mars 2016

Objectif de l’atelier

Revue de code de la gallerie d’exemples fournie avec le sdk.

Les sources :
http://code.qt.io/cgit/qt/qtquickcontrols2.git/tree/examples/controls/gallery
%QTDIR%/Examples/Qt-5.6/qtquickcontrols2/controls/gallery

Contexte

Qt supporte plusieurs méthodes pour faire de l'IHM :

- historique : totalement en C++ en utilisant les QWidget
- en QML :
  - à "la mano" à partir des Item
  - en utilisant les Controls
  - en utilisant les nouveaux Controls arrivés en technology preview avec Qt 5.6
- Canvas3D, Qt3D

L'objectif de cet atelier est de découvrir ces nouveaux contrôles en QML

Comparaison avec les QtQuick Controls 1.0

- couverture fonctionnelle quasi-identique (mais pas de Hover en Desktop)
- plus performant (temps de chargement et mémoire)
- plus simple (ex : ScrollView vs Flickable avec indicator/scroolbar)
- Style basique universel ultra simple + Universal + Material (un style Qt en préparation)
- basé sur le concept de template (comportement : tout ce qui n'est pas visuel)
- de nouveaux éléments (Drawer, RangeSlider, SwipeView) et certains controles importés de Qt.Extra
- DPIAware

	Qt Quick Controls	Qt Labs Controls
==	ApplicationWindow, BusyIndicator, Button, CheckBox, ComboBox, Label, Menu, ProgressBar, RadioButton, Slider, SpinBox, Switch, TextArea, TextField, ToolBar, ToolButton	
	StatusBar, TreeView, TableView	pas d'équivalent
	ColorDialog, FileDialog, FontDialog	pas d'équivalent
!=	MessageDialog, Dialog	Popup
!=	Action	Shortcut
!=	Calendar	MonthGrid, DayOfWeekRow, WeekNumberColumn
!=	GroupBox	GroupBox, Frame
!=	ScrollView	ScrollBar, ScrollIndicator
!=	Stack, StackView, StackViewDelegate	StackView
!=	Tab, TabView	TabBar, SwipeView

Style

- sélection d’un style
- paramétrage
- customization

Par défaut, 3 styles sont disponibles :

- **default** : universel, simpliste et très léger
- **Universal** : style suivant les recommandations de Microsoft
- **Materiel** : style suivant les recommandations de Google

On peut sélectionner le style (par ordre de précédance) :

- avec un argument en ligne de commande : *-style STYLE*
- avec la variable d'environnement QT_LABS_CONTROLS_STYLE
- avec un fichier de configuration :/qtlabscontrols.conf dans le QRC

Si le style n'est pas disponible, le style se replie sur Default

Pour le style Material :

- Material.theme : Material.Light / Material.Dark
- Material.primary : n'importe-quelle couleur (Material.BlueGray par défaut)
- Material.accent : n'importe-quelle couleur (Material.Teal par défaut)

Pour le style Universal :

- Universal.theme : Universal.Light / Universal.Dark
- Universal.accent : n'importe-quelle couleur (Universal.Teal par défaut)

Créer son style

- Créer une copie d'un style existant (Material par exemple)
- Le modifier en s'inspirant des guidelines : https://doc-snapshots.qt.io/qt5-5.6/qtlabscontrols-customize.html
- Dans le QML, importer le nouveau style (éventuellement avec un alias si l'on souhaite utilise les controles originaux en même temps)

```
import QtQuick 2.6
import Qt.labs.controls 1.0
import "../MyStyle/" as C

Pane {
    C.Label {
        width: parent.width
        text: "My Label"
    }
}
```

High-DPI support

Méthode 1 : en definissant l'attribut AA_EnambleHighDpiScaling avant la construction de QGuiApplication

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>

int main(int argc, char *argv[])
{
    QGuiApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();
}
```

Méthode 2 : en définissant la variable d'environnement **QT_AUTO_SCREEN_SCALE_FACTOR** à 1

informations complémentaires : https://doc-snapshots.qt.io/qt5-5.6/highdpi.html

Mise en route

création de l’application

sélection du module

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>

int main(int argc, char *argv[])
{
    QGuiApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    return app.exec();
}
```

main.cpp

```
import QtQuick 2.6
import Qt.labs.controls 1.0

ApplicationWindow {
        title: "My Application"
        width: 640; height: 480
        visible: true

        Button {
                text: "Hello World!"
                anchors.centerIn: parent
                onClicked: Qt.quit()
        }
}
```

Présentation des controles

- les containers
- les boutons
- les autres controles d’entrée
- les menus
- les indicateurs
- divers...

Les conteneurs

- ApplicationWindow
- Container
- Drawer
- Frame
- GroupBox
- Page
- Pane
- StackView
- SwipeView
- TabBar
- ToolBar

Ces controles servent à regrouper d'autres contrôles enfants. Ces conteneurs héritent de **Container**.

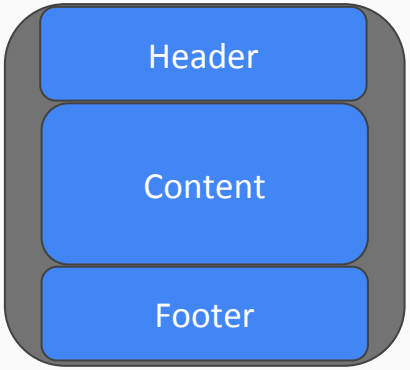
Le conteneur racine est une **ApplicationWindow**, elle définit un en-tête, une zone de contenu et un pied de page.

Les **Frame** et les **GroupBox** permettent de regrouper visuellement des controles. Le **GroupBox** est un **Frame** avec un titre.

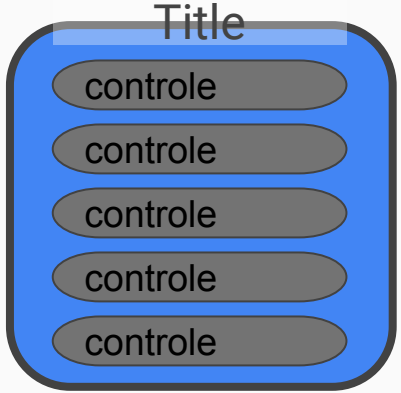
Pour une page d'écran, on pourra utiliser un **Pane** ou un **Page** (**Pane** avec un en-tête et un pied de page)

En fonction de l'organisation souhaitée, on se tournera vers un **StackView** pour une pile et vers un **SwipeView** pour un défilement par glissement.

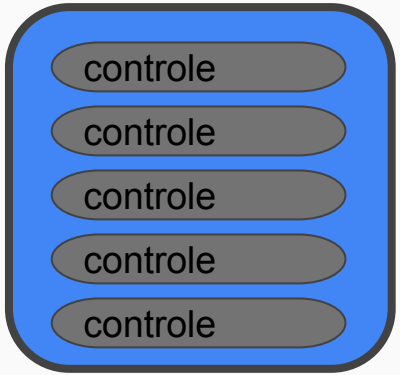
Enfin la **TabBar** permet une gestion avec des onglets tandis que la ToolBar est une barre d'outils.



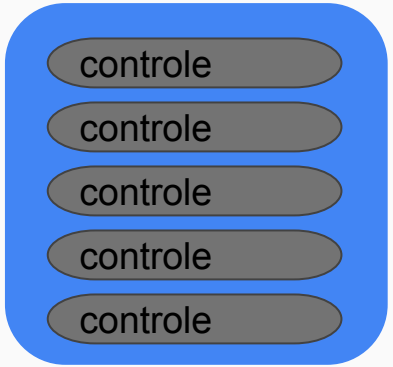
ApplicationWindow
Page



GroupBox

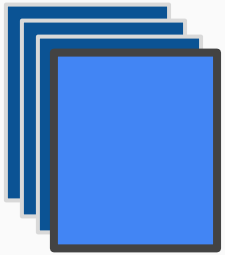


Frame

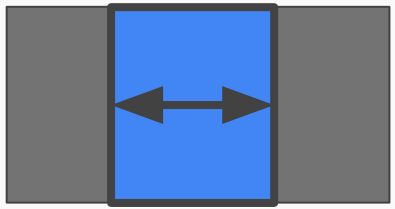


Pane

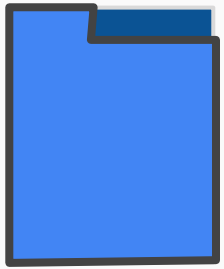
ToolBar



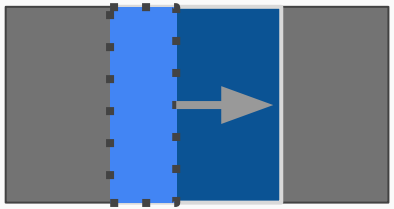
StackView



SwipeView



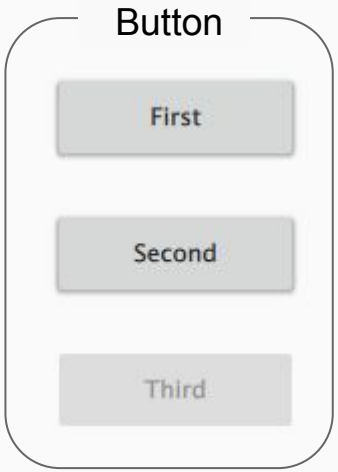
TabBar

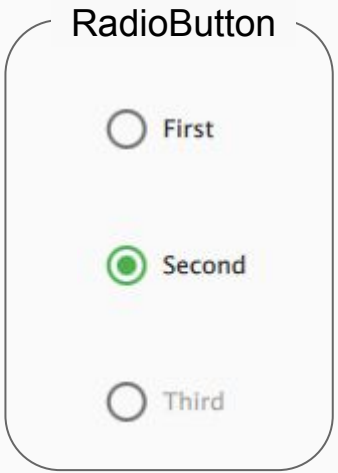


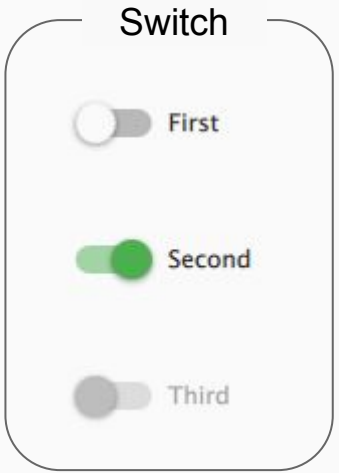
Drawer

Button Controls

- AbstractButton
- Button
- CheckBox
- RadioButton
- Switch
- ToolButton

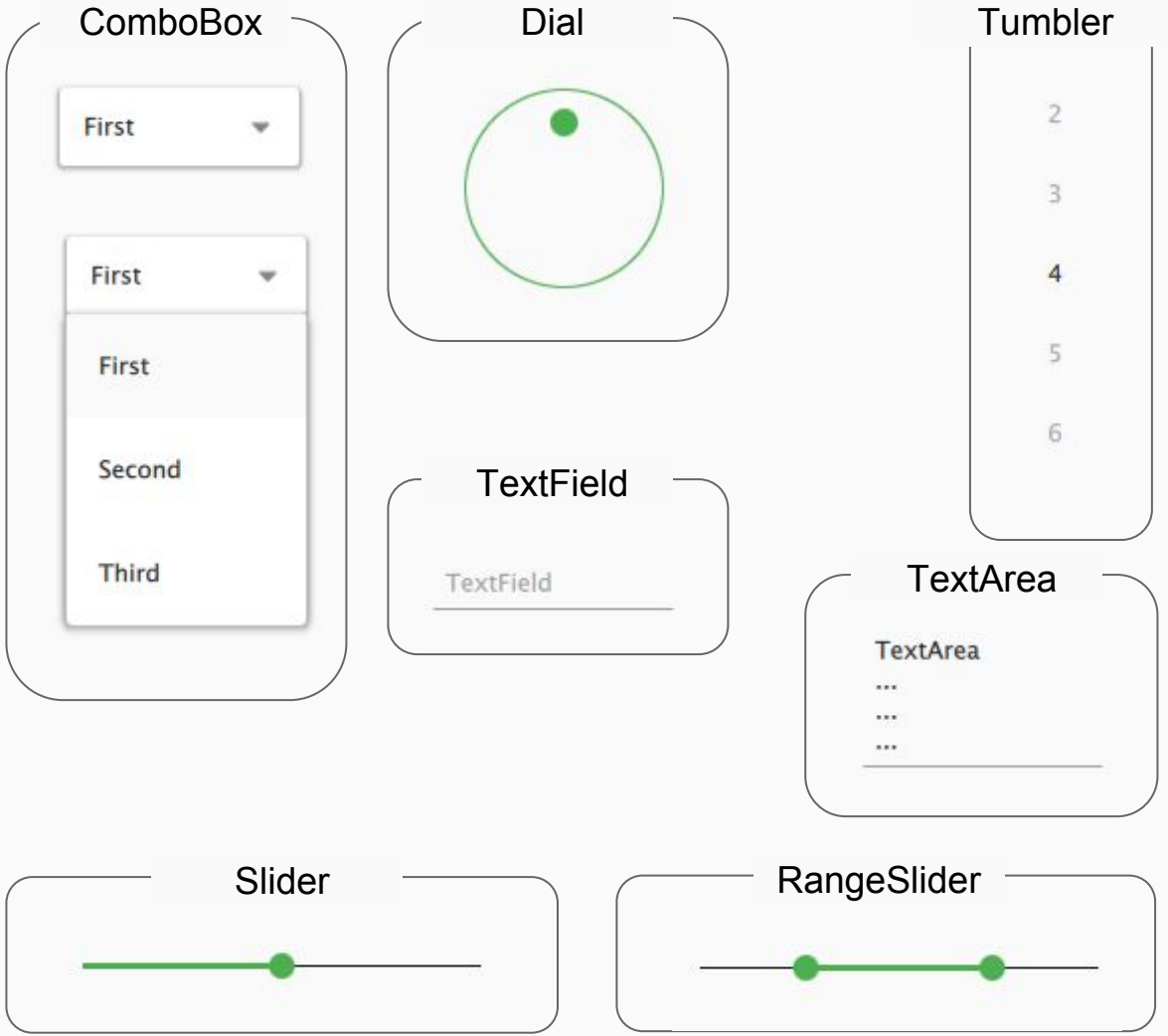






Input controls

- ComboBox
- Dial
- RangeSlider
- Slider
- TextArea
- TextField
- Tumbler



Menu controls

- Menu
- MenuItem

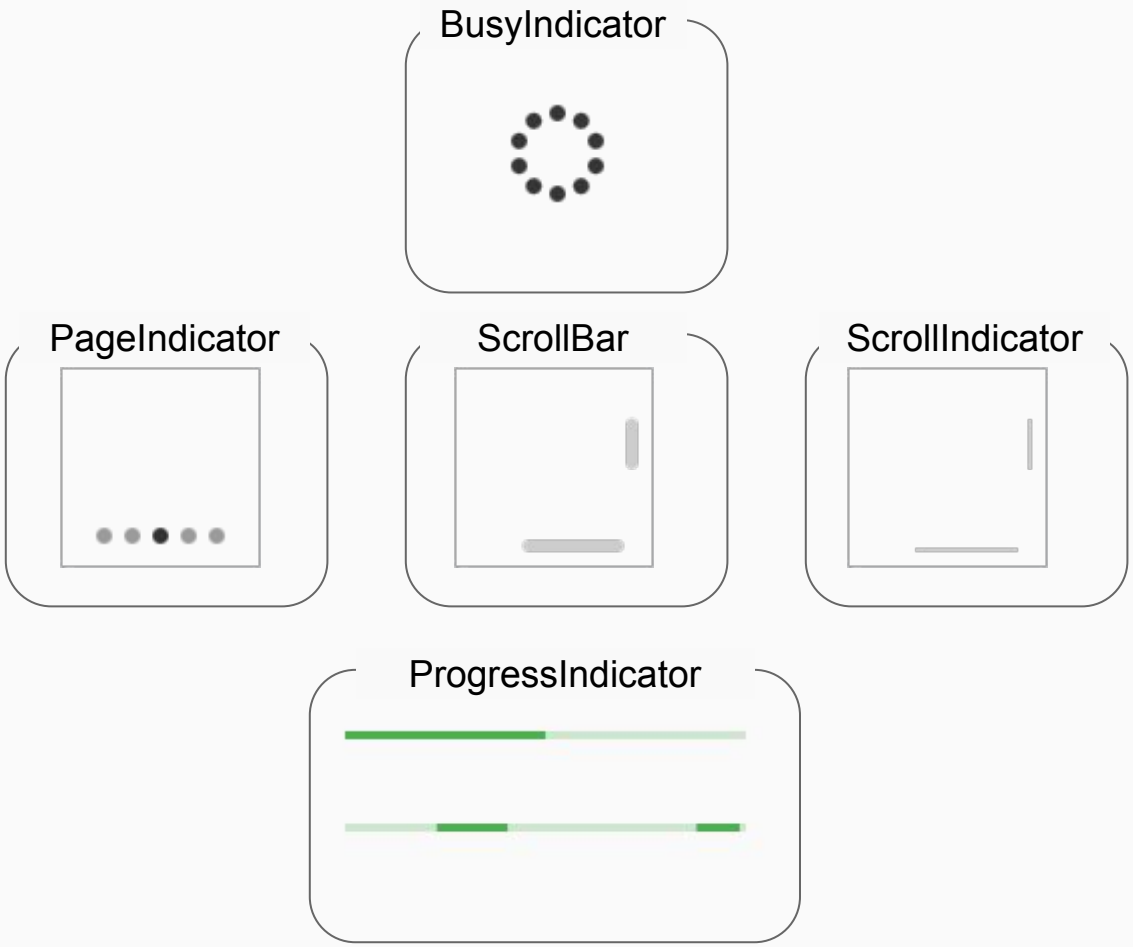
```
Button {
        id: button
        onClicked: menu.open()

        Menu {
                id: menu
                y: button.height

                MenuItem {
                        text: "New..."
                        onTriggerred:console.log(“triggerred”)
                }
        }
}
```

Indicator controls

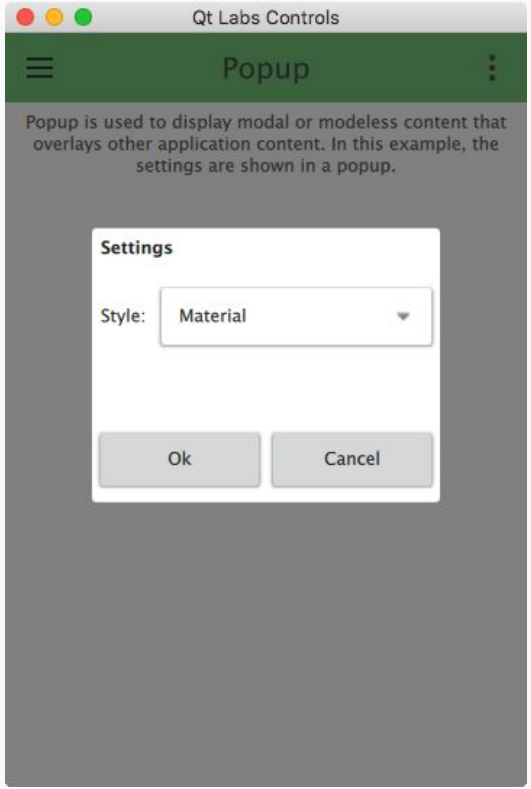
- BusyIndicator
- PageIndicator
- ProgressBar
- ScrollBar
- ScrollIndicator



Divers

- Popup
- Label
- Control
- ItemDelegate
- ButtonGroup
- SpinBox
- TabButton

- Les **TabButtons** s'utilisent avec les TabBar
- Les **Label** permettent d'afficher du texte
- **Control** est l'élément de base dont les autres controles héritent.
- Un **ButtonGroup** n'est pas visible mais permet de définir un groupe de controles intérragissants.
- **ItemDelegage** est utilisable avec les **ListView** et les **ComboBox** et permet d'afficher un élément d'un modèle avec un comportement par défaut (un text, sélectionnable..)

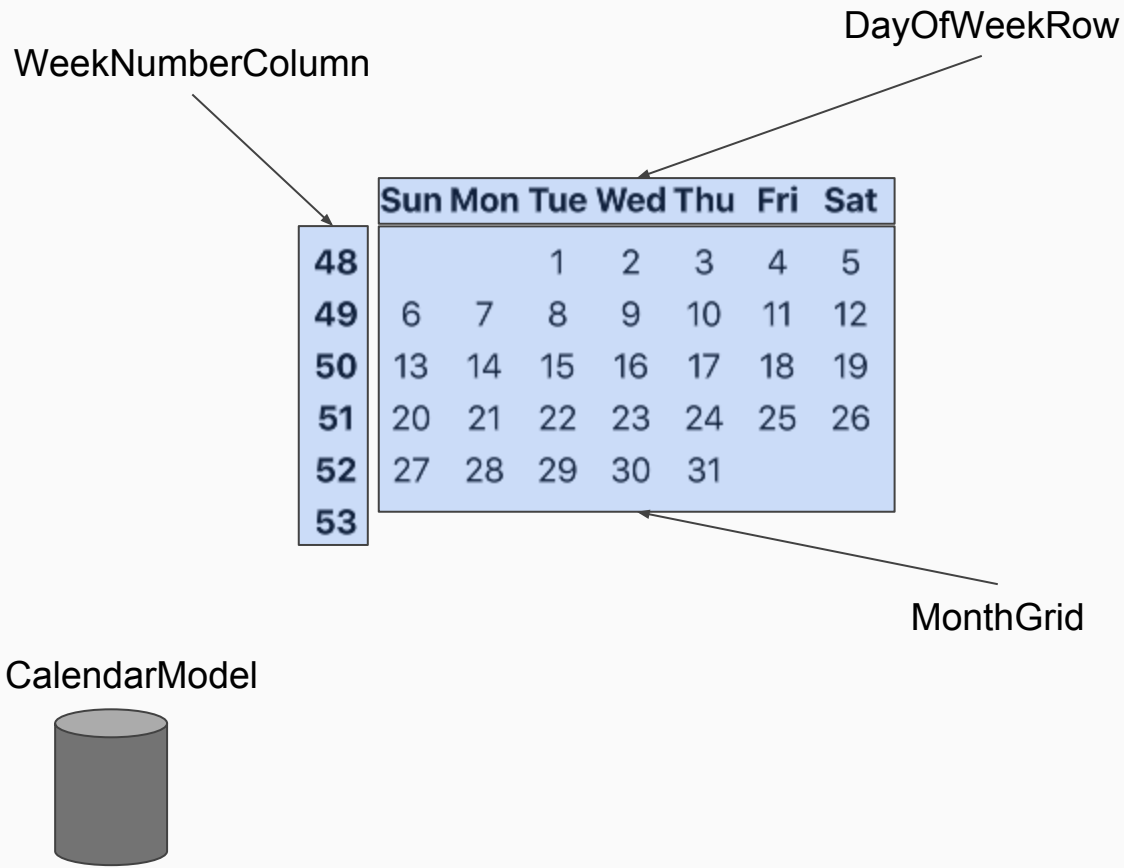


Popup

SpinBox

Qt.labs.calendar

- Calendar
- CalendarModel
- DayOfWeekRow
- MonthGrid
- WeekNumberColumn



Revue de code

Guillaume Charbonnier
gcharbonnier@a-team.fr