

Développement mobile avec

Meetup #4 : Caméra

Objectif de l'atelier



Une application très simple en QML avec :

- Gestion basique de la caméra (prise de photo, flash, zoom)
- Extension en QML
- Extension en C++
- Lecture QrCode

Les sources sur gitHub : <https://github.com/a-team-fr/MeetupMobileQtQml/tree/master/160229/DemoProject>

Etapes

1. création du projet - (time box : 10mn)
2. ajouter un panneau de contrôle - (time box : 10mn)
3. prendre une photo - (time box : 10mn)
4. gestion du zoom et du flash - (time box : 15mn)
5. ajouter un controleur en C++ - (time box : 30mn)

Etape 1

création du projet

- Création du projet
- Activation du module Multimedia
- Une fenêtre, une caméra et un viewfinder

Activation du module Multimedia

1. Créer un projet de type Application (Qt Quick Application)
2. Modifier le fichier .pro pour activer les modules

```
QT += qml quick multimedia
```

3. Modifier le fichier main.qml pour charger les modules nécessaires

```
import QtMultimedia 5.5  
import QtQuick.Controls 1.4
```

Une fois ces étapes réalisées, les types QML suivants seront disponibles : **Audio, MediaPlayer, Radio, Video** et bien entendu... **Camera !**

Les types multimedia QML qui vont nous intéresser

- **Camera** : Récupérer une frame, prendre une photo, une vidéo
- **VideoOutput** : afficher le contenu du capteur de la camera (le viewfinder)
- **QtMultimedia** : objet global avec des infos
- autres types à utiliser pour contrôler les réglages de la caméra :
 - **CameraCapture** : prendre une photo
 - **CameraRecorder** : prendre un film
 - **CameraExposure** : régler l'exposition
 - **CameraFlash** : gestion du flash
 - **CameraFocus** : une interface pour administrer la mise au point
 - **CameraImageProcessing** : réglage des paramètres pour prendre une photo ou un film

Ajouter Note : camera et un viewfinder

L'objet **Camera** n'est pas visible, on utilise un **VideoOutput** pour afficher les frames de la Camera.

On donne une taille au **VideoOutput** (ici on lui demande de prendre toute la place de la fenêtre parente).

Et on lui indique de s'alimenter avec le contenu de la **Camera** *camera*

```
import QtQuick 2.3
import QtQuick.Window 2.2
import QtMultimedia 5.5

Window {
    visible: true
    width:1024
    height:768

    Camera{
        id:camera
    }
    VideoOutput{
        id:viewfinder
        source:camera
        anchors.fill: parent
    }
}
```

Voilà le meetup est terminé...merci et à la prochaine fois !

Quelques cas d'utilisations fréquents

- sélectionner une caméra
- transformer le viewfinder (rotation, échelle)

Avant d'aller plus loin, voyons ensemble quelques techniques pour répondre à des situations classiques...


```

...
Camera{
    id:camera
}

ListView {
    anchors.fill: parent

    model: QtMultimedia.availableCameras
    delegate: Text {
        text: modelData.displayName

        MouseArea {
            anchors.fill: parent
            onClicked: camera.deviceId = modelData.deviceId
        }
    }
}

```

ou simplement sélectionner la
caméra Avant/Arrière sur mobile :

```

Camera{
    id:camera
    position: Camera.BackFace
    //position:Camera.FrontFace
}

```

```
VideoOutput{
    id: viewfinder
    source: camera
    anchors.fill: parent
    orientation: 90
    autoOrientation: false
    fillMode: VideoOutput.PreserveAspectRatio
    // autres mode : Stretch, PreserveAspectCrop
    scale: height/width
    rotation : 12
    transformOrigin: Item.Center
}
```

Orientation : pas de 90° - vous pouvez aussi récupérer celle du capteur : *camera.orientation*. On peut utiliser *autoOrientation* pour que l'orientation se synchronise avec celle de la fenêtre (portrait vs paysage).

Tips : utiliser *Qt.platform.os* si besoin de faire un traitement particulier

Sinon, le **VideoOutput** étant un objet visible, il hérite des propriétés d'un **Item** :

scale : homothétie

rotation : valeur en degré de la rotation autour du point défini par transformOrigin

On pourrait aussi jouer avec un capteur (penser à activer sensors) :

```
OrientationSensor{
    active: true
    if (reading.orientation === OrientationReading.LeftUp)
    { ... }
}
```

Etape 2

ajout d'un panneau de contrôle

...ce sera l'occasion de voir une manière de créer de nouveaux types QML ;-)

- Ajouter un panneau de controle en overlay avec un nouveau type utilisateur
- Ajouter un bouton pour prendre une photo
- Ajouter un sélecteur de mode pour le flash

Si on souhaite créer un nouveau type (organiser son code et se constituer sa bibliothèque), il suffit :

- de créer un nouveau document QML **avec un nom commençant par une majuscule**
- pour faciliter le déploiement, ajouter le nouveau document dans votre fichier de ressource (QRC)
- on peut alors l'utiliser dans un autre document :
 - directement si le nouveau type est au même niveau que le document qui l'utilise
 - avec **import** dans le cas contraire

/Main.qml

```
import "../myLibs"
```

```
MonSuperNouveauType{  
    isSomethingImportant : true  
    anchors.fill : parent  
}
```

/myLibs/MonSuperNouveauType.qml

```
import QtQuick 2.0
```

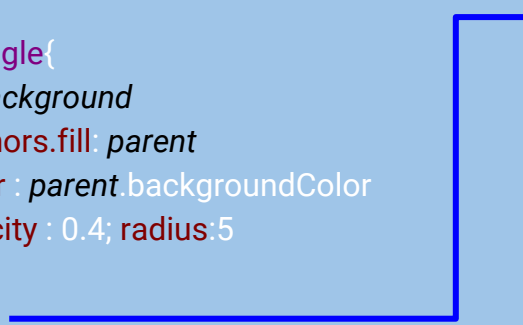
```
Item {  
    property bool isSomethingImportant : true  
    property alias text : label.text  
    Text{  
        id:label  
        color: parent.isSomethingImportant ? "red" : "black"  
    }  
}
```

Overlay.qml

```
import QtQuick 2.0
import QtMultimedia 5.5

Item {
    id: root
    property color backgroundColor : "darkblue"
    signal takePhoto()
    property int flashMode : Camera.FlashAuto

    Rectangle{
        id: background
        anchors.fill: parent
        color : parent.backgroundColor
        opacity : 0.4; radius:5
    }
    Row{
    }
}
```



```
Row{
    Image{
        source:"qrc:/photo.png"
        MouseArea{ anchors.fill: parent; onClicked: root.takePhoto(); }
    }
    Image{
        id: flashMode
        source: root.flashMode === Camera.FlashAuto ?
            "qrc:/camera_flash_auto.png" : "qrc:/camera_flash_off.png"
        MouseArea{
            anchors.fill: parent; onClicked: {
                if (root.flashMode === Camera.FlashAuto)
                    root.flashMode = Camera.FlashOff;
                else root.flashMode = Camera.FlashAuto;
            }
        }
    }
}
```

```
Overlay{  
    //our control panel as custom type  
    anchors.bottom: parent.bottom  
    anchors.horizontalCenter: parent.horizontalCenter  
    width : 100; height:50  
    onTakePhoto: console.log("takePhoto")  
    onFlashModeChanged: console.log("FlashMode:"+flashMode)  
}
```

Notes :

Toutes les propriétés d'un type QML envoient un signal
<NomPropriété>Changed()

Questions :

- Je n'ai pas mis le contenu du panneau de contrôle (**Row**) directement dans *Background* (**Rectangle**)... pourquoi à votre avis ?
- Quelle est la différence entre :

```
Item{  
    //mon super truc  
}
```

```
Rectangle{  
    //mon super truc  
    color: "transparent"  
}
```

- Quelle est la différence entre `visible:false` et `opacity:0`

Réponses :

- *Le fond est transparent mais je ne voulais pas que les éléments du Row le soit !*
- *chaque pixel du rectangle est rendu en OpenGL, ce qui pourrait dégrader inutilement la performance sur certaines plate-formes*
- *un objet qui n'est pas visible n'est pas présent dans le scene graph openGL et ne sera pas actif. Alors qu'un objet complètement transparent se comporte normalement même s'il ne se voit pas...*

Etape 3

prendre une photo

- ajouter un élément image pour visualiser la photo
- prendre la photo

```
Image{  
    id:previewImage  
    anchors.fill: parent  
    fillMode: Image.PreserveAspectFit  
    visible: false  
    MouseArea{  
        anchors.fill: parent; onClicked:parent.visible = false  
    }  
}
```

```
Overlay{  
    onTakePhoto: camera.imageCapture.capture();  
}
```

```
Camera{  
    id:camera  
    imageCapture {  
        onImageCaptured: {  
            previewImage.source = preview;  
            previewImage.visible = true;  
        }  
        onImageSaved: console.log("picture saved to :"+path)  
    }  
}
```

Avec l'élément ***imageCapture*** (du type **CameraCapture**), on peut :

- connaître le chemin de la dernière photo enregistrée avec ***capturedImagePath*** (plutôt que de le récupérer dans le slot ***onImageSaved()***)
- définir la résolution avec...***resolution***
- tester si la camera est prête avant de prendre une photo avec la propriété ***ready***
- enregistrer des métadonnées (on peut utiliser certaines données comme les coordonnées GPS, le mode portrait/paysage...fournies par la propriété ***metaData*** de la **Camera**) dans la photo avec la méthode **setMetadataKey**
- changer le chemin où les photos s'enregistrent avec **captureToLocation()** plutôt que **capture()**

On peut enregistrer une vidéo de manière similaire avec l'élément *videoRecorder de la caméra* (du type **CameraRecorder**) en appelant les méthodes **record()** et **stop()**.

Il faudra mettre la caméra en mode prise de video avec
`camera.captureMode = Camera.CaptureVideo`

On utilisera alors un élément du type **CameraPlayer** (au lieu de **Image**) pour afficher la vidéo prise en l'alimentant à travers un **MediaPlayer** dont la source sera celle du **videoRecorder** de la caméra (**camera.videoRecorder.actualLocation**).

Etape 4

gérer le zoom et le flash

- gérer le flash
- gérer le zoom

On a déjà fait le plus difficile, il ne reste qu'à modifier notre Caméra pour utiliser le sélecteur de mode de flash fourni par notre panneau de controle :

```
Camera{  
    //our Camera to play with  
    id:camera  
    flash.mode: overlay.flashMode  
    ..  
}
```

```
PinchArea{
    anchors.fill: parent
    enabled: true
    pinch.minimumScale: 1
    pinch.maximumScale: camera.maximumDigitalZoom
    scale: camera.digitalZoom
    onPinchStarted: {
        scale = camera.digitalZoom;
        zoom.visible = true;
    }
    onPinchFinished: zoom.visible = false;
    onPinchUpdated: {
        camera.digitalZoom = pinch.scale;
    }
}
```


Etape 5

Intégrer un contrôleur en C++

...ce sera l'occasion de voir une manière de créer de nouveaux types QML mais cette fois en C++

- créer une classe C++ “QML compliant”
- rendre disponible un objet C++ en tant qu'élément QML (ou une classe C++ en tant que type QML)
- définir des propriétés, des signaux...

```
#ifndef CAMERACONTROLLER_H  
#define CAMERACONTROLLER_H  
#include <QObject>
```

```
class CameraController : public QObject
```

→ hérite de QObject

```
{
```

```
    Q_OBJECT
```

→ utilise la macro Q_OBJECT

```
public:
```

```
    explicit CameraController(QObject *parent = 0);
```

```
signals:
```

```
public slots:
```

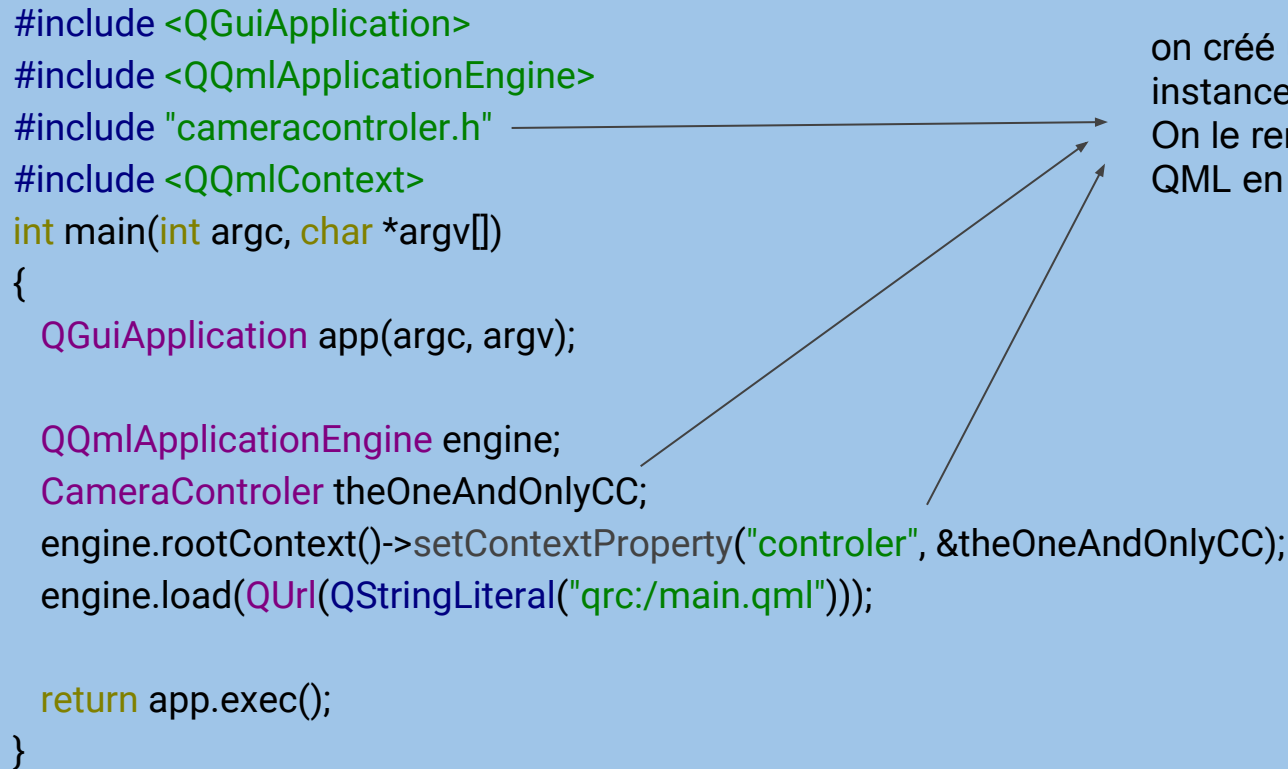
```
};
```

```
#endif // CAMERACONTROLLER_H
```

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "cameracontroller.h"
#include <QQmlContext>
int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    CameraController theOneAndOnlyCC;
    engine.rootContext()->setContextProperty("controller", &theOneAndOnlyCC);
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

    return app.exec();
}
```



on crée un objet “theOneAndOnlyCC”
instance de **CameraController**.
On le rend disponible à notre contexte
QML en tant que “**controller**”

Si on avait eu besoin de plusieurs contrôleurs, il aurait été pratique d'enregistrer la classe `CameraController` en tant que nouveau type QML afin de pouvoir créer des éléments QML de ce nouveau type directement QML.

On aurait procédé ainsi :

```
qmlRegisterType<CameraController>("Controller", 1, 0, "Controller");
```

cameracontroler.h

```
class CameraControler : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString greeting READ getGreeting NOTIFY greetingChanged)

public:
    explicit CameraControler(QObject *parent = 0);
    QString getGreeting(){
        return "Hi, I am "+name;
    }
    Q_INVOKABLE void setName(QString newName){
        name = newName; emit greetingChanged();
    }

signals:
    void greetingChanged();

private:
    QString name="the Big controler";
};
```

main.qml

```
Text{
    anchors.centerIn: parent
    width:200; height:50
    text : controler.greeting
    color:"red"
    MouseArea{
        anchors.fill: parent
        onClicked: controler.setName("the TINY
controler")
    }
}
```



Guillaume Charbonnier
gcharbonnier@a-team.fr