Lors de cette nouvelle étude, je vous propose d'implémenter des applications qui permettent d'écouter de la musique, de lire des vidéos, de prendre des photos et des vidéos, en utilisant encore une fois les compétences de QML. Commençons par la partie audio.

Composant Audio en QML

Notre premier projet dans le domaine du multimédia consiste à lire un fichier audio ou une musique dont le nom du fichier sera directement proposé dans le code **QML**. Nous verrons par la suite comment lire un fichier audio parmi une liste proposée dans un répertoire spécifique.

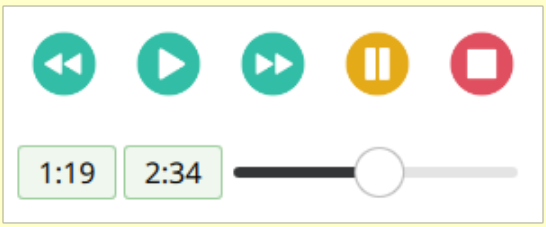
La gestion du flux audio est très simple puisque la classe Audio est prévue pour ce genre d'utilisation. Grâce à cette classe, il est possible de récupérer le fichier audio au moyen de la propriété source, d'écouter le fichier son avec la méthode play(), d'arrêter en cours de lecture avec la méthode pause() ou d'arrêter complètement cette lecture au moyen de la méthode stop(). Enfin, nous pouvons commencer (ou recommencer) la lecture à un endroit spécifique du fichier audio à l'aide de la méthode seek().

Nous pouvons également connaître la durée totale de l'enregistrement avec la propriété duration et aussi le temps écoulé depuis le début de la lecture au moyen de la propriété position.

Par contre, cette classe ne dispose pas d'éléments visuels pour réaliser ces différents traitement. Vous devez composer votre IHM en utilisant des composants annexes qui vont contrôler les différents éléments que nous venons d'évoquer. À noter que Audio peut être remplacé en lieu et place par le composant MediaPlayer qui est plus généraliste mais qui possède exactement les mêmes propriétés et les mêmes méthodes. Faites en l'expérience avec le projet ci-dessous.

Notre projet est découpé en plusieurs parties afin de créer des composants adaptés aux différents éléments visuels tel que les boutons de gestion de lecture et les affichages du temps écoulé et du temps total.

Bouton.qml




```
import QtQuick 2.0
import QtQuick.Controls 2.4

ToolButton {
  id: bouton
  property alias icone: bouton.icon.source

  width: 48
  height: 48
  background: Rectangle { color: "transparent" }

  icon {
    width: parent.width
    height: parent.height
    color: "transparent"
  }
}
```

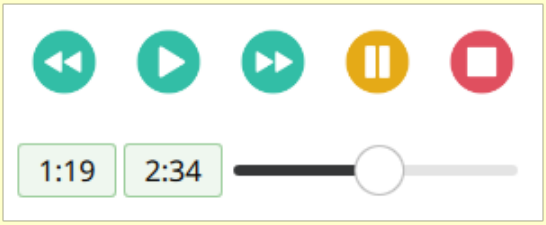
Horloge.qml

```
import QtQuick 2.0

Rectangle {
  property int secondes: 1
  property int minutes
  property int reste

  onSecondesChanged: {
    minutes = secondes/60
    reste = secondes%60
    temps.text = minutes +":"+ (reste < 10 ? "0"+reste : reste)
  }
  color: "#10008000"
  border.color: "#60008000"
  radius: 3;
  height: 30
  width: 55

  Text {
    id: temps
    font.pixelSize: 16
    anchors.centerIn: parent
  }
}
```

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtMultimedia 5.9
import QtQuick.Controls 2.4

Window {
  visible: true
  width: 300
  height: 150
  title: qsTr("Lecteur audio")

  Audio {
    id: lireMusique
    source: "file:///home/manu/Musique/musique.mp3"
```

```
    onDurationChanged: {
      progression.to = duration
      tempsTotal.secondes = duration/1000
    }
    onPositionChanged: {
      progression.value = position
      tempsRestant.secondes = position/1000
    }
  }

  Row {
    anchors {
      top: parent.top
      topMargin: 8
      horizontalCenter: parent.horizontalCenter
    }
    spacing: 10

    Bouton {
      icone: "icones/rembobiner.png"
      onPressed: lireMusique.seek(lireMusique.position-5000)
    }
    Bouton {
      icone: "icones/play.png"
      onPressed: lireMusique.play()
    }
    Bouton {
      icone: "icones/avance-rapide.png"
      onPressed: lireMusique.seek(lireMusique.position+5000)
    }
    Bouton {
      icone: "icones/pause.png"
      onPressed: lireMusique.pause()
    }
    Bouton {
      icone: "icones/arreter.png"
      onPressed: lireMusique.stop()
    }
  }

  Row {
    id: temps
    spacing: 4

    anchors {
      bottom: parent.bottom
      bottomMargin: 13
      left: parent.left
      leftMargin: 8
    }
    Horloge { id: tempsRestant }
    Horloge { id: tempsTotal; secondes: 0 }
  }

  Slider {
    id: progression
    anchors {
      bottom: parent.bottom
      bottomMargin: 8
      left: temps.right
      right: parent.right
      rightMargin: 8
    }
  }
}
```



Lire plusieurs fichiers audios

Reprenons le projet précédent en permettant cette fois-ci de choisir un fichier audio parmi une liste et en permettant également de naviguer dans les différents répertoires. Pour ce genre de procédé, il existe bien entendu un composant de haut niveau qui est capable de gérer les répertoires et les fichiers associés, il s'agit de la classe ***FolderListModel***.

Il s'agit d'un modèle que vous pouvez configurer à votre convenance, en précisant le répertoire de base grâce à la propriété folder, en proposant un filtre avec nameFilters, en décidant si les dossiers sont visibles ou pas avec les propriétés showDirs, showDirsFirst et showDotAndDotDot, etc.

Pour chaque élément constituant cette liste de fichiers, il est également possible de récupérer les informations concernant le fichier ou le répertoire correspondant, comme : le nom du ficher complet avec fileName, le chemin complet avec filePath, l'URL du fichier avec fileURL, le nom du fichier sans l'extension avec fileBaseName, l'extension du fichier avec fileSuffix, la taille du fichier avec flieSize et en autre s'il s'agit d'un répertoire ou pas avec fileIsDir.

Comme précédemment, nous reprenons les composants personalisés « Bouton.qml » et « Horloge.qml ». Le premier n'est pas modifié par ce nouveau projet, par contre pour le suivant, nous changeons ses dimensions.

Horloge.qml

```
import QtQuick 2.0

Rectangle {
  property int secondes: 1
  property int minutes
  property int reste
```

```
  onSecondesChanged: {
    minutes = secondes/60
    reste = secondes%60
    temps.text = minutes +":"+ (reste < 10 ? "0"+reste : reste)
  }
  color: "#10008000"
  border.color: "#60008000"
  radius: 3;
  height: 35
  width: 40

  Text {
    id: temps
    font.pixelSize: 13
    anchors.centerIn: parent
  }
}
```

Nous rajoutons un nouveau composant personnalisé qui correspond au fichier audio sélectionné :

Choix.qml

```
import QtQuick 2.0

Rectangle {
  property alias nomFichier: nom.text
  color: "#10800000"
  border.color: "#60800000"
  radius: 3;
  height: 35

  Text {
    id: nom
    font.bold: true
    font.pixelSize: 13
    color: "darkred"
    anchors.centerIn: parent
    width: parent.width-20
    wrapMode: Text.Wrap
  }
}
```

Nous rajoutons un autre composant qui représente chaque fichier (ou répertoire) de la liste des fichiers donnée par le modèle vu précédemment.

Musique.qml

```
import QtQuick 2.0
import QtQuick.Controls 2.4

Rectangle {
  property alias nomFichier: nom.text
  property alias icone: icone.icon.source
  property url nomUrl
  property bool estUnRépertoire

  anchors {
    right: parent.right
    left: parent.left
  }
  height: 40
  color: "#30008000"
  border.color: "#60008000"
  radius: 5;

  ToolButton {
    id: icone
    anchors.verticalCenter: parent.verticalCenter
    background: Rectangle { color: "transparent" }
    icon {
      width: 32
      height: 32
      color: "transparent"
    }
  }

  Text {
    id: nom
    anchors {
      verticalCenter: parent.verticalCenter
      left: parent.left
      leftMargin: 45
    }
    color: "darkgreen"
    font.bold: true
    font.pixelSize: 14
    width: parent.width-50
    wrapMode: Text.Wrap
  }
}
```

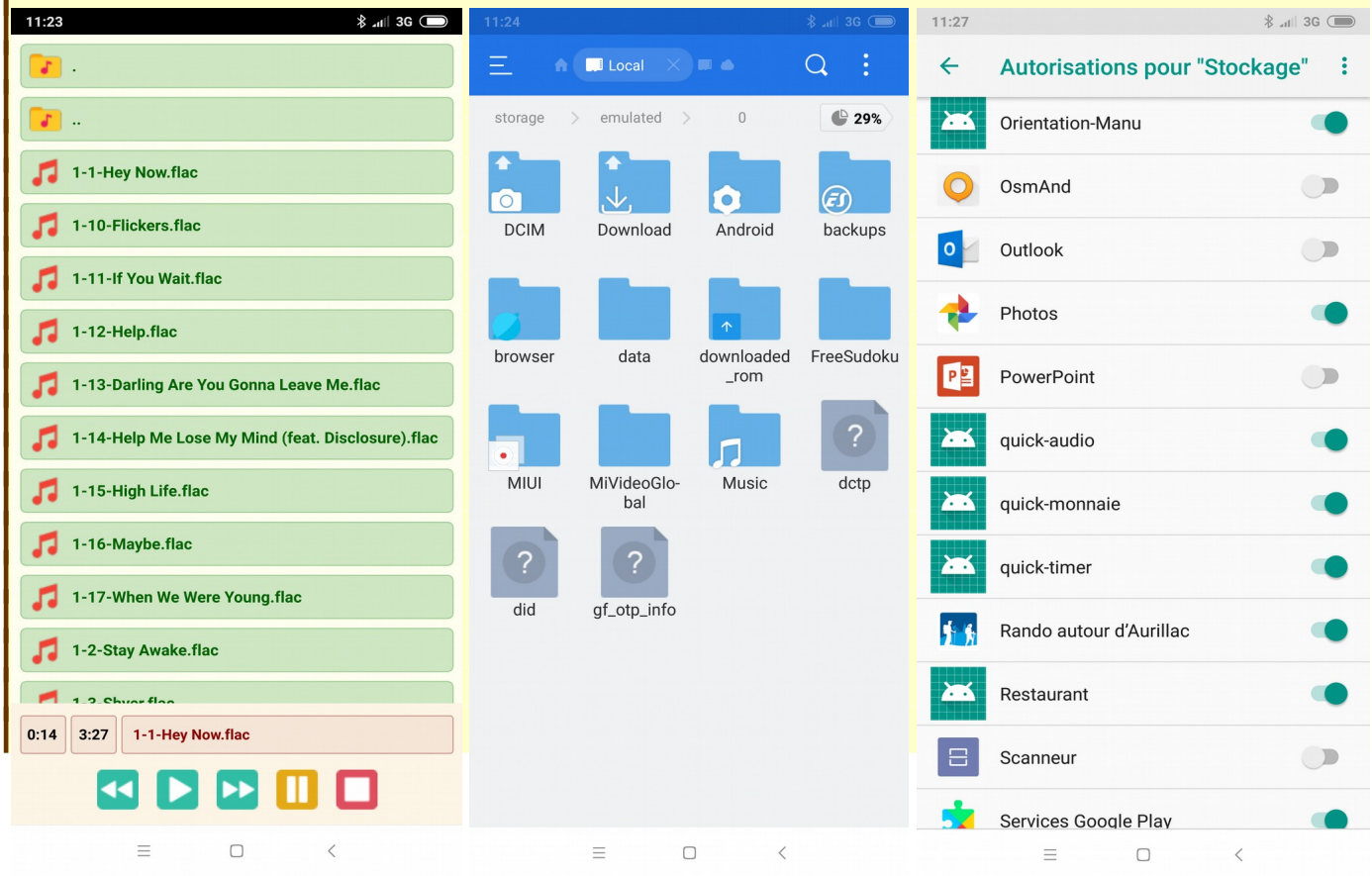
Et enfin notre vue principale qui factorise l'ensemble de ces composants.

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtMultimedia 5.9
import Qt.labs.folderlistmodel 2.2

Window {
  visible: true
  width: 320
  height: 480
  title: qsTr("Lecteur audio")
  color: "ivory"

  MediaPlayer {
    id: lireMusique
    onDurationChanged: tempsTotal.secondes = duration/1000
    onPositionChanged: tempsRestant.secondes = position/1000
  }

  FolderListModel {
    id: fichiers
    nameFilters: ["*.flac", "*.mp3", "*.wav"]
    showDirs: true
    showDirsFirst: true
    showDotAndDotDot: true
    folder: Qt.platform.os=="android" ? "file:///storage/emulated/0/Music" : "file:///home/manu/Musique"
  }

  ListView {
    spacing: 8
    anchors {
      top: parent.top
      margins: 8
      left: parent.left
      right: parent.right
      bottom: temps.top
    }
    model: fichiers
    delegate: Musique {
      id: musique
      nomFichier: fileName
      nomUrl: fileURL
      estUnRépertoire: fileIsDir
      icone: estUnRépertoire ? "icones/dossier-musique.png" : "icones/note-de-musique.png"
      MouseArea {
        anchors.fill: parent
        onClicked: {
          if (musique.estUnRépertoire)  fichiers.folder = musique.nomUrl
          else {
            choixMusique.nomFichier = musique.nomFichier
            lireMusique.source = musique.nomUrl
            lireMusique.play()
          }
        }
      }
    }
  }

  Rectangle {
    color: "oldlace"// "#F3FFFF"
    height: 110
    anchors {
      bottom: parent.bottom
      right: parent.right
      left: parent.left
    }
  }

  Row {
    id: boutons
    anchors {
      bottom: parent.bottom
      bottomMargin: 8
      horizontalCenter: parent.horizontalCenter
    }
    spacing: 4
    Bouton { icone: "icones/rembobiner.png"; onPressed: lireMusique.seek(lireMusique.position-5000) }
    Bouton { icone: "icones/play.png"; onPressed: lireMusique.play() }
    Bouton { icone: "icones/avance-rapide.png"; onPressed: lireMusique.seek(lireMusique.position+5000) }
    Bouton { icone: "icones/pause.png"; onPressed: lireMusique.pause() }
    Bouton { icone: "icones/arreter.png"; onPressed: lireMusique.stop()  }
  }

  Row {
    id: temps
    spacing: 4
    anchors {
      bottom: boutons.top
      bottomMargin: 8
      left: parent.left
      leftMargin: 8
```

```
    }
    Horloge { id: tempsRestant }
    Horloge { id: tempsTotal; secondes: 0 }
  }

  Choix {
    id: choixMusique
    anchors {
      bottom: boutons.top
      bottomMargin: 8
      left: temps.right
      leftMargin: 4
      right: parent.right
      rightMargin: 8
    }
  }
}
```

Vous remarquez que cette application peut fonctionner sur votre smartphone android. Remarquez alors l'arborescence particulière pour retrouver vos musiques préférées. Enfin, toujours sur votre smartphone, il peut être nécessaire d'autoriser l'utilisation de la zone de stockage. Voir les différentes captures ci-dessous :



Lire des vidéos

Dans la suite logique de ces composants propres au multimédia, et toujours en utilisant la classe ***MediaPlayer***, il est tout à fait possible de visualiser des vidéos. La visualisation se fait toutefois grâce à un composant spécifique associé au ***MediaPlayer*** et qui s'appelle ***VideoOutput***.

Encore une fois, il s'agit d'une simple zone rectangulaire où le film apparaîtra sans boutons de gestions associés. C'est à vous de concevoir la vue à votre convenance avec les composants nécessaires à ce genre de lecture.

Bouton.qml

```
import QtQuick 2.0
import QtQuick.Controls 2.4

ToolButton {
  id: bouton
  property alias icone: bouton.icon.source

  width: 40
  height: 40
  checkable: true

  background: Rectangle { color: "transparent" }
  icon {
    width: parent.width
    height: parent.height
    color: "transparent"
```

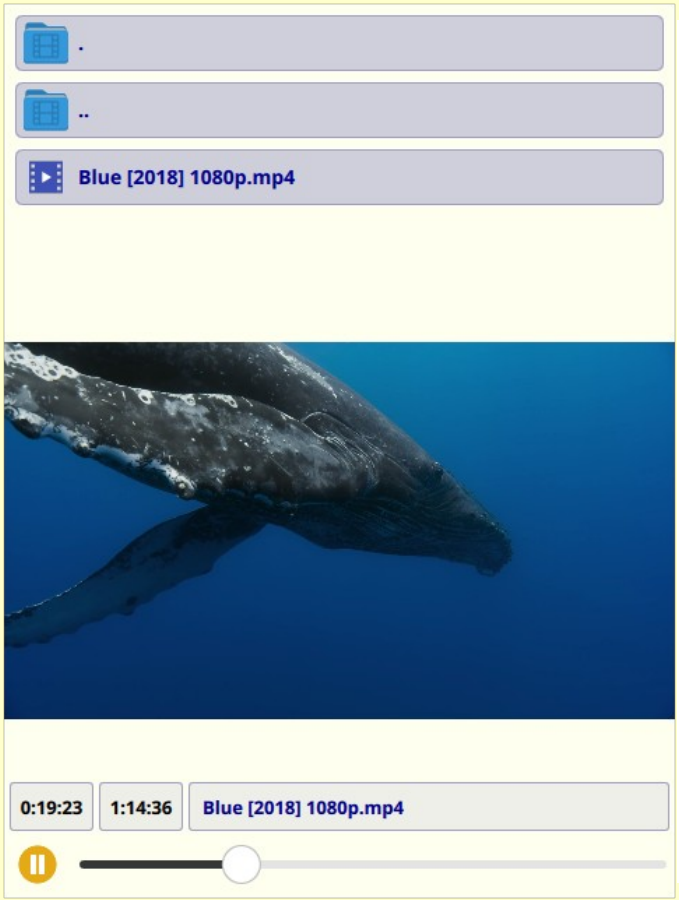
```
    source: "icones/play.png";
  }
}
```

Horloge.qml

```
import QtQuick 2.0

Rectangle {
  property int secondes: 1
  property int minutes
  property int heures
  property int reste

  onSecondesChanged: {
    heures = secondes/3600
    minutes = secondes/60 % 60
    reste = secondes%60
    temps.text = heures +":"+ (minutes<10 ? "0"+minutes : minutes) +":"+ (reste<10 ? "0"+reste : reste)
  }
  color: "#10000080"
  border.color: "#60000080"
  radius: 3;
  height: 35
  width: 60

  Text {
    id: temps
    font.pixelSize: 13
    font.bold: true
    anchors.centerIn: parent
  }
}
```

FichierVideo

```
import QtQuick 2.0
import QtQuick.Controls 2.4

Rectangle {
  property alias nomFichier: nom.text
  property alias icone: icone.icon.source
  property url nomUrl
  property bool estUnRépertoire

  anchors {
    right: parent.right
    left: parent.left
  }
  height: 40
  color: "#30000080"
  border.color: "#60000080"
  radius: 5;

  ToolButton {
    id: icone
    anchors.verticalCenter: parent.verticalCenter
    background: Rectangle { color: "transparent" }
    icon {
      width: 32
      height: 32
      color: "transparent"
    }
  }

  Text {
    id: nom
    anchors {
      verticalCenter: parent.verticalCenter
      left: parent.left
      leftMargin: 45
    }
    color: "darkblue"
    font.bold: true
    font.pixelSize: 14
    width: parent.width-50
    wrapMode: Text.Wrap
  }
}
```



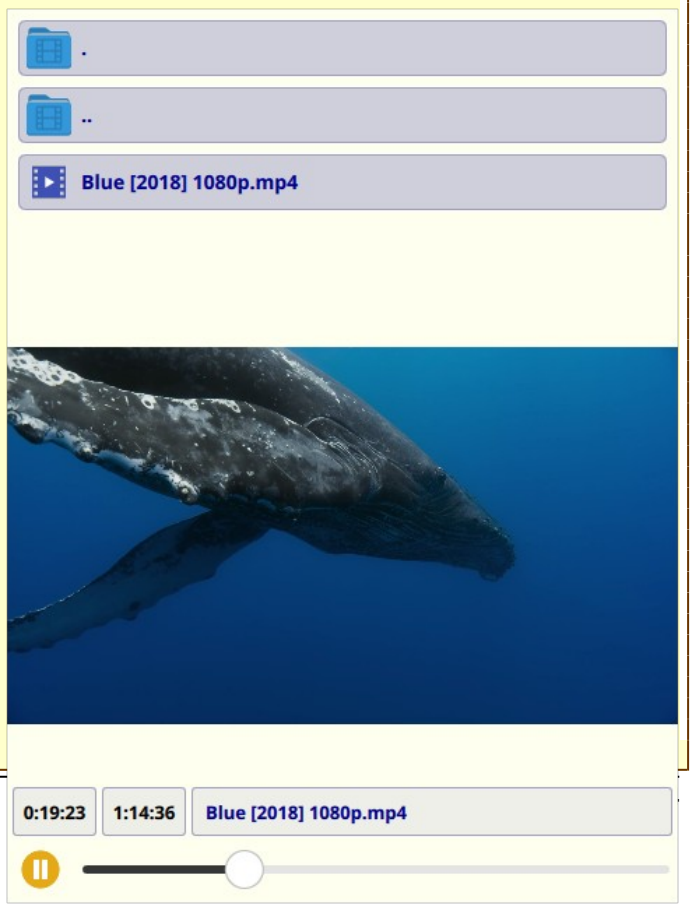
Choix.qml

```
import QtQuick 2.0

Rectangle {
  property alias nomFichier: nom.text
  color: "#10000080"
  border.color: "#60000080"
  radius: 3;
  height: 35
  Text {
    id: nom
```

```
    font.bold: true
    font.pixelSize: 13
    color: "darkblue"
    anchors.centerIn: parent
    width: parent.width-20
    wrapMode: Text.Wrap
  }
}
```

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtMultimedia 5.9
import Qt.labs.folderlistmodel 2.2
import QtQuick.Controls 2.4

Window {
  visible: true
  width: 480
  height: 640
  title: qsTr("Lecteur vidéo")
  color: "ivory"

  MediaPlayer {
    id: lireMédia
    onDurationChanged: {
      tempsTotal.secondes = duration/1000
      progression.to = duration
    }
    onPositionChanged: {
      tempsRestant.secondes = position/1000
      progression.value = position
    }
  }

  FolderListModel {
    id: fichiers
    nameFilters: ["*.mp4", "*.mkv"]
    showDirs: true
    showDirsFirst: true
    showDotAndDotDot: true
    folder: Qt.platform.os=="android" ? "file:///storage/emulated/0/DCIM/Camera" : "file:///home/manu/Vidéos"
  }

  ListView {
    spacing: 8
    anchors {
      top: parent.top
      margins: 8
      left: parent.left
      right: parent.right
      bottom: lecteurVidéo.top
    }
    model: fichiers
    delegate: FichierVideo {
      id: video
      nomFichier: fileName
      nomUrl: fileURL
      estUnRépertoire: fileIsDir
        icone: estUnRépertoire ? "icones/dossier-video.png" :
"icones/fichier-video.png"

      MouseArea {
        anchors.fill: parent
        onClicked: {
              if (video.estUnRépertoire)   fichiers.folder =
video.nomUrl
          else {
            choixVidéo.nomFichier = video.nomFichier
            lireMédia.source = video.nomUrl
            lecture.toggle()
          }
        }
      }
    }
  }

  VideoOutput {
    id: lecteurVidéo
    source: lireMédia
    height: 320
    anchors {
      bottom: choixVidéo.top
      bottomMargin: 20
      right: parent.right
      left: parent.left
    }
  }

  Row {
    id: boutons
```



```
    anchors {
      bottom: parent.bottom
      bottomMargin: 4
      right: parent.right
      rightMargin: 4
      left: parent.left
      leftMargin: 4
    }
    spacing: 4

    Bouton {
      id: lecture
      onCheckedChanged: {
        if (checked) {
          lireMédia.play()
          icone = "icones/pause.png"
        }
        else {
          lireMédia.pause()
          icone = "icones/play.png"
        }
      }
    }

    Slider {
      id: progression
      property bool changement: false
      from: 0
      width: boutons.width - lecture.width
      onMoved: {
        lireMédia.pause()
        changement = true
      }
      onPressedChanged: {
        if (changement) {
          lireMédia.seek(value)
          lireMédia.play()
          changement = false
        }
      }
    }
  }

  Row {
    id: temps
    spacing: 4
    anchors {
      bottom: boutons.top
      bottomMargin: 4
      left: parent.left
      leftMargin: 4
    }
    Horloge { id: tempsRestant }
    Horloge { id: tempsTotal; secondes: 0 }
  }

  Choix {
    id: choixVidéo
    anchors {
      bottom: boutons.top
      bottomMargin: 4
      left: temps.right
      leftMargin: 4
      right: parent.right
      rightMargin: 4
    }
  }
}
```

Visionneuse

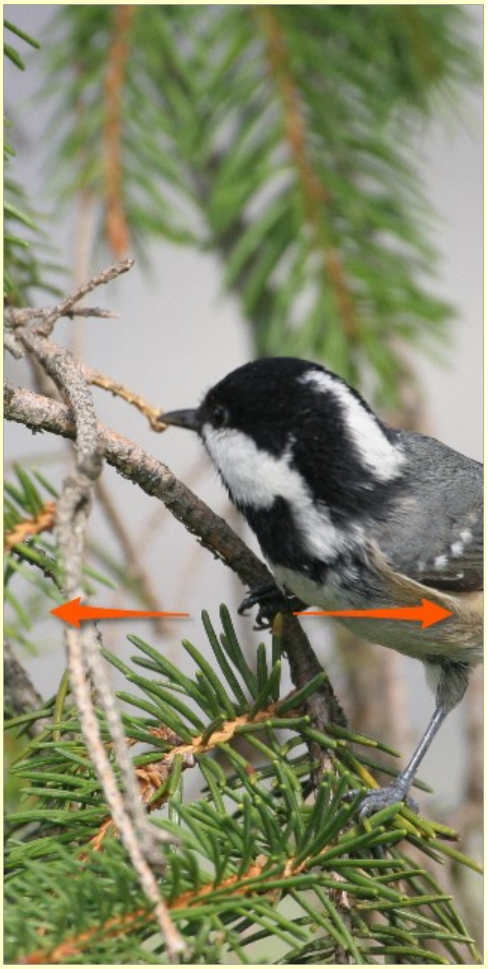
Je vous propose de reprendre un des projets que nous avons déjà mis en œuvre lors de notre première étude et qui permet de consulter les photos que nous avons prises à l'aide notre smartphone. Nous pouvons naviguer d'une photo à l'autre par le système « ***faire-glisser*** » de la droite vers la gauche (et vice-versa) et de permettre en plus un grossissement de la photo en faisant un double-clic sur celle-ci.

> ***L'ergonomie choisie nous permet, dans le même projet, de prendre un composant de type SwipeView (pour le « faire-glisser ») et un composant de type StackView afin de prévoir une vue supplémentaire qui s'affichera automatiquement (qui par réglage viendra de la partie haute) pour montrer un grossissement de la photo choisie.***

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import Qt.labs.folderlistmodel 2.2
import QtQuick.Controls 2.4

Window {
  visible: true
  width: 320
```

```
  height: 640
  title: qsTr("Visionneuse")

  FolderListModel {
    id: fichiers
    nameFilters: ["*.jpg", "*.png", "*.jpeg"]
    folder: Qt.platform.os=="android" ? "file:///storage/emulated/0/DCIM/Camera" : "file:///home/manu/Images"
  }

  StackView {
    anchors.fill: parent
    initialItem: SwipeView {
      Repeater {
        model: fichiers
        Page {
          Image {
            anchors.fill: parent
            source: fileURL
            fillMode: Image.PreserveAspectCrop
            MouseArea {
              anchors.fill: parent
              onDoubleClicked: {
                photo.source = parent.source
                agrandissement.open()
              }
            }
          }
        }
      }
    }
  }

  Drawer {
    id: agrandissement
    width: parent.width
    height: parent.height
    edge: Qt.TopEdge

    Image {
      id: photo
      onSourceSizeChanged: {
        width = sourceSize.width/2
        height = sourceSize.height/2
      }

      MouseArea {
        id: souris
        anchors.fill: parent
        drag.target: parent
        cursorShape: drag.active ? Qt.SizeAllCursor : Qt.ArrowCursor
        onDoubleClicked: agrandissement.close()
      }
    }
  }
}
```



Appareil photo intégré

Dans ce chapitre, je vous invite à utiliser l'appareil photo (la caméra) intégré à votre smartphone et de pouvoir ainsi prendre des photos avec quelques réglages possibles. Encore une fois, il existe des classes qui implémentent les différentes fonctionnalités d'un appareil photo.

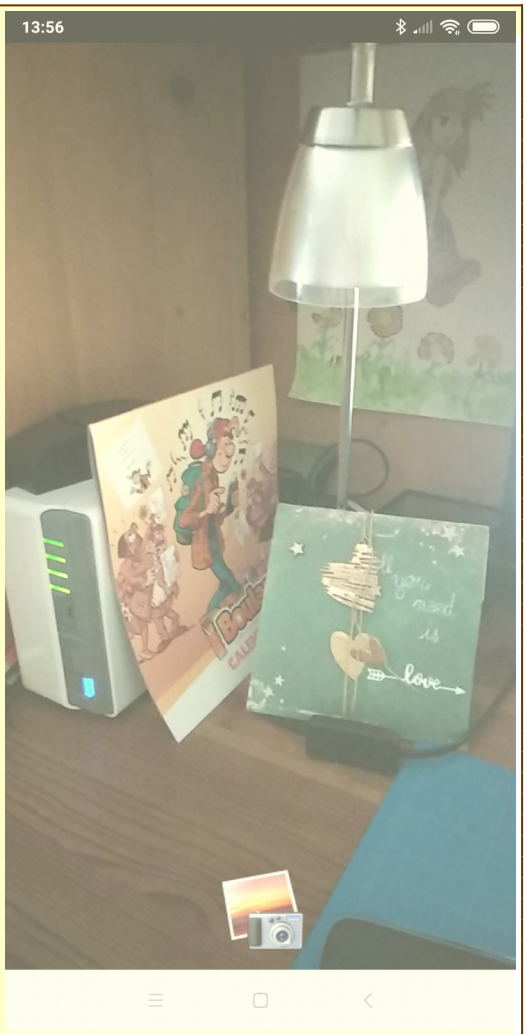
La classe principale se nomme tout simplement « Camera » qui gère l'appareil photo dans sa globalité, mais il existe également des classes annexes qui s'occupent de réglages plus spécifiques, comme l'exposition « CameraExposure », la gestion de l'autofocus « CameraFocus », le post-traitement « CameraImageProcessing », la capture de l'image correspondante « CameraCapture », etc.

Pour visualiser ce que voit l'objectif de la caméra, vous devez utiliser la classe « VideoOutput », la même que précédemment, ensuite vous devez enregistrer la prise de vue dans un simple composant « Image ». Voici ci-dessous un exemple rudimentaire mais qui permet toutefois de voir les différents réglages qui sont à notre disposition.

Main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtMultimedia 5.9
import QtQuick.Controls 2.4

Window {
  visible: true
  width: 320
  height: 480
  color: "black"
  title: qsTr("Appareil photo")

  Camera {
    id: appareil
    exposure {
      exposureMode: Camera.ExposureAction
```

```
      manualIso: 800
    }
    focus {
      focusMode: CameraFocus.FocusContinuous
      focusPointMode: Camera.FocusPointAuto
    }
    imageProcessing {
      saturation: -1
      contrast: 1
    }
    flash.mode: Camera.FlashAuto
    imageCapture.onImageCaptured: visu.source = preview
    captureMode: Camera.CaptureStillImage
  }

  VideoOutput {
    id: vidéo
    source: appareil
    anchors.fill: parent
    orientation: appareil.orientation
  }

  Timer {
    id: temps
    interval: 5000
    onTriggered: visu.visible = false
  }
  Image {
    id: visu
    fillMode: Image.PreserveAspectCrop
    anchors.fill: parent
  }

  ToolButton {
    anchors {
      bottom: parent.bottom
      bottomMargin: 8
      horizontalCenter: parent.horizontalCenter
    }
    icon {
      source: "qrc:/icone/photo_camera.png"
      color: "transparent"
      width: 64
      height: 64
    }
    onClicked: {
      visu.visible = true
      temps.start()
      appareil.imageCapture.capture()
    }
    background: Rectangle { color: "transparent" }
  }
}
```



Code-barres – QR Code – DataMatrix

Il est souvent intéressant de pouvoir décoder à l'aide de son smartphone des **code-barres**, des **QR Code** des codes **DataMatrix**, etc. Il existe une bibliothèque tierce **QZXing** qui ne fait pas encore partie de la librairie **Qt** mais qui peut s'intégrer facilement, avec même la possibilité d'utiliser la technologie **QML**.

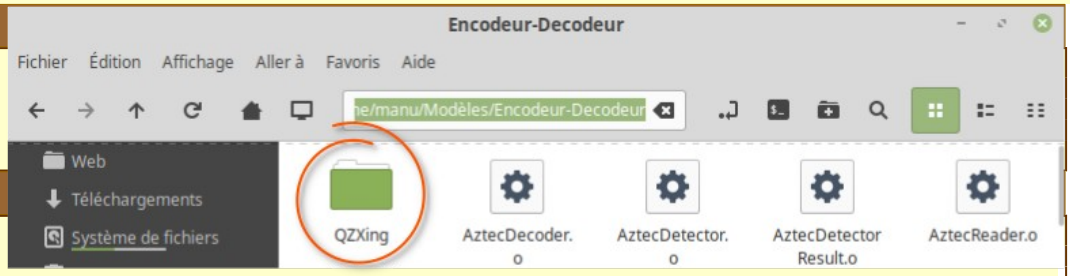
Une fois, que vous avez récupéré la bibliothèque complète, il suffit de prendre tous les sources dans le répertoire « src » que vous devez placer dans vos différents projets. Pour que cela soit plus pratique, je vous invite à renommer ce dossier avec le nom plus évocateur « QXZing ». L'avantage de cette démarche, c'est que vous pourrez exploiter cette bibliothèque aussi bien sur votre PC que sur un système Android, par contre, l'inconvénient, c'est vous aurez beaucoup de fichiers compilés dans votre répertoire de projet.

Cette librairie permet de décoder n'importe quel type de code, par contre elle ne permet pour l'instant que de générer que des QRCode. Cela n'est pas un gros problème puisque sur Internet, vous disposez de générateurs en ligne gratuits. Pour que ces sources soient pris en compte dans votre projet, vous devez rajouter la ligne include(QZXing/QZXing.pri) dans votre fichier de projet.

Pour commencer, je vous propose de prendre un projet sans librairie Qt, en simple C++, qui permet de générer un QRCode, sous forme de fichier image de type PNG, à partir d'un texte codé en dur, directement dans le source.

Encodeur-Decodeur.pro

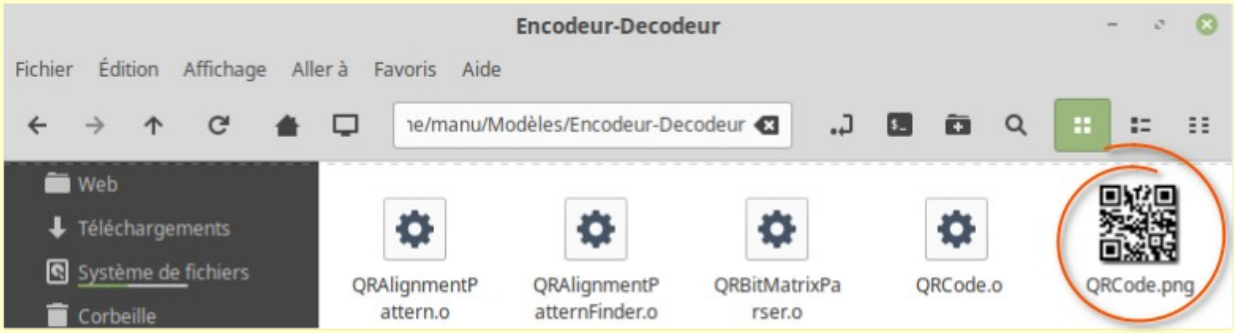
```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle qt
include(QZXing/QZXing.pri)
SOURCES += main.cpp
```



main.cpp

```
#include <QZXing.h>

int main()
{
  QString data = "BTS-SNIR\nSalle-214";
  QImage barcode = QZXing::encodeData(data);
  barcode.save("QRCode.png");
```

```
}
```



Très très peu de lignes de code pour réaliser ce fichier image. C'est impressionnant de simplicité. Je vous propose maintenant de voir la partie décodage sur le même projet, avec ce même fichier. Il suffit juste de modifier l'intérieur de la fonction main().

main.cpp

```
Fichier Édition Affichage Rechercher Terminal Aide
BTS-SNIR
Salle-214
Appuyez sur <ENTRÉE> pour fermer cette fenêtre...
```

```
#include <QZXing.h>
#include <iostream>
using namespace std;

int main()
{
  QImage imageToDecode("QRCode.png");
  QZXing decoder;
  decoder.setDecoder(QZXing::DecoderFormat_QR_CODE | QZXing::DecoderFormat_CODABAR | QZXing::DecoderFormat_DATA_MATRIX);
  QString result = decoder.decodeImage(imageToDecode);
  cout << result.toStdString() << endl;
}
```

Comme vous le remarquez, vous pouvez choisir tous les types de décodage pour le même fichier Image. Bien entendu, vous pouvez être beaucoup plus restrictif si vous connaissez déjà la nature du codage.

Génération d'un QR Code en QML

Toujours à l'aide de cette librairie **QZXing**, je vous propose maintenant de réaliser un générateur de **QR Code** à partir d'un texte saisie dans une zone prévue à cet effet. Le **QR Code** se génère à la volée sur la **vue** principale, après chaque caractère introduit. Le fichier image correspondant se génère également à chaque modification du texte saisi.

Pour implémenter ce générateur de code, vous devez préciser dans le fichier de projet que vous prenez en compte le mode QML.

Encodeur-QML.pro

```
QT += quick
CONFIG += c++11 qzxing_qml
SOURCES += main.cpp
HEADERS += capturer.h
RESOURCES += qml.qrc

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

include(QZXing/QZXing.pri)
```

Pour la génération du fichier image, nous devons passer par un contrôleur avec juste un « slot » qui réalise le traitement souhaité, le même que nous avons déjà réalisé dans le chapitre précédent.

Capturer.h

```
#ifndef CAPTURER_H
#define CAPTURER_H

#include <QObject>
#include <QZXing.h>

class Capturer : public QObject
{
  Q_OBJECT
public:
  explicit Capturer(QObject *parent = nullptr) : QObject(parent) {}
public slots:
  void enregistrer(QString message) {
    QImage barcode = QZXing::encodeData(message);
    barcode.save("QRCode.png");
  }
};

#endif // CAPTURER_H
```

Ensuite, dans le fichier principal « main.cpp », vous devez bien sûr prendre en compte ce contrôleur, mais aussi permettre l'utilisation de cette librairie QZXing dans les fichiers QML. Pour cela, vous devez enregistrer tous les composants (les classes) nécessaires au codage et décodages suivant les situations requises. Introduisez la commande QZXing::registerQMLTypes();

Enfin, pour la génération automatique de l'image représentant le QR Code dans la vue, cette librairie propose un fournisseur d'image (provider) que vous devez intégrer dans votre projet à l'aide de la méthode QZXing::registerQMLImageProvider(engine);

Enfin, côté vue, il suffit ensuite de fixer le source du composant Image à l'aide de l'url qui fait appel au fournisseur que nous venons d'évoquer :

```
Image { source: "image://QZXing/encode/"+message.text }
```

main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QZXing.h>
#include <QtQml>
#include <capturer.h>

int main(int argc, char *argv[])
{
  QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
  QGuiApplication app(argc, argv);
  QQmlApplicationEngine engine;

  QZXing::registerQMLTypes();
  QZXing::registerQMLImageProvider(engine);

  Capturer capture;
  engine.rootContext()->setContextProperty("capture", &capture);

  engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
  if (engine.rootObjects().isEmpty()) return -1;
  return app.exec();
}
```

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.4

Window {
  visible: true
  width: 320
  height: 480
  color: "bisque"
  title: qsTr("QR Code")

  TextField {
    id: message
    font.pixelSize: 14
    anchors {
      top: parent.top
      topMargin: 10
      left: parent.left
      leftMargin: 10
      right: parent.right
      rightMargin: 10
    }
    placeholderText: "Votre message à coder"
    background: Rectangle { color: "ivory"; radius: 7 }
  }

  Image {
    id: image
    anchors {
      top: message.bottom
      topMargin: 50
      horizontalCenter: parent.horizontalCenter
    }
    source: "image://QZXing/encode/"+message.text
    sourceSize.width: 240
    sourceSize.height: 240
    onSourceChanged: capture.enregistrer(message.text)
  }
}
```

Décoder un QR Code ou un DataMatrix à l'aide d'une application QML

Nous connaissons tous des applications Android qui permettent de décoder un **QR Code.** C'est l'application que je vous propose de réaliser maintenant avec en plus la possibilité de décoder aussi un **Datamatrix**, toujours avec la même librairie **QZXing**.

Pour cela, nous avons besoin de l'objectif du smartphone qui a pour conséquence d'utiliser les classes Camera et VideoOutput, comme nous l'avons déjà réalisé lors d'une prise de vue. À ceci se rajoute le composant QZXingFilter qui va réaliser l'analyse vidéo nécessaire pour interpréter l'image avec le QR Code ou le Datamatrix. C'est à ce niveau là que nous réalisons les réglages et la prise en compte du décodage (la prise de vue peut être de travers, cela ne pose aucun problème).

Pour que cette classe QZXingFilter puisse être prise en compte, vous devez alors remplacer le module « qzxing_qml » par le module « qzxing_multimedia », dans le fichier de projet.

Decodeur-QML.pro

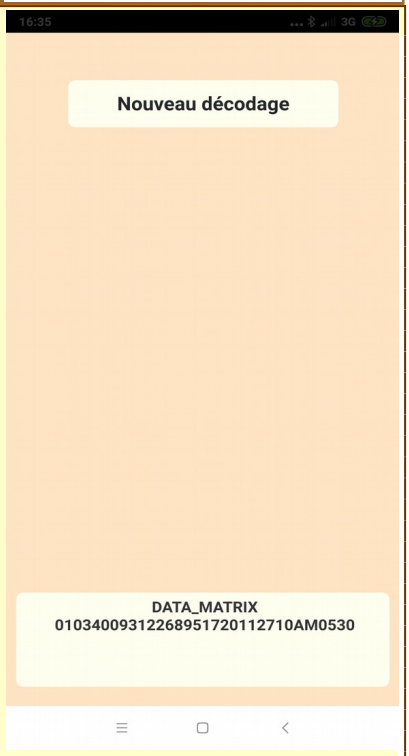
```
QT += quick
CONFIG += c++11 qzxing_multimedia

DEFINES += QT_DEPRECATED_WARNINGS
SOURCES += main.cpp
RESOURCES += qml.qrc

# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

include(QZXing/QZXing.pri)
```

main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QZXing.h>

int main(int argc, char *argv[])
{
  QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
  QGuiApplication app(argc, argv);

  QZXing::registerQMLTypes();

  QQmlApplicationEngine engine;
  engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
  if (engine.rootObjects().isEmpty()) return -1;

  return app.exec();
}
```

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2
import QtMultimedia 5.9
import QZXing 2.3
import QtQuick.Controls 2.4

Window {
  visible: true
  width: 320
  height: 480
  color: "bisque"
  title: qsTr("Scanneur de codes")

  Camera {
    id: objectif
    focus {
      focusMode: CameraFocus.FocusContinuous
      focusPointMode: CameraFocus.FocusPointAuto
    }
  }
  VideoOutput {
    id: visu
    source: objectif
    filters: [filtres]
    autoOrientation: true
    anchors.fill: parent
  }
  QZXingFilter {
    id: filtres
    decoder {
      enabledDecoders: QZXing.DecoderFormat_DATA_MATRIX | QZXing.DecoderFormat_QR_CODE
      onTagFound: {
        réponse.text = decoder.foundedFormat()
        réponse.append(tag)
        objectif.stop()
      }
    }
    onDecodingStarted: réponse.text = "Scan..."
  }

  Button {
    anchors {
      top: parent.top
      topMargin: 50
      horizontalCenter: parent.horizontalCenter
    }
    width: 270
    height: 50
    font.pixelSize: 20
    font.bold: true
    text: "Nouveau décodage"
    onClicked: objectif.start()
    background: Rectangle { color: "ivory"; radius: 7 }
  }
```



```
  TextArea {
    id: réponse
    background: Rectangle { color: "ivory"; radius: 7 }
    readOnly: true
    font.pixelSize: 16
    font.bold: true
    height: 100
    horizontalAlignment: TextEdit.AlignHCenter
    wrapMode: Text.Wrap
    anchors {
      bottom: parent.bottom
      bottomMargin: 20
      left: parent.left
      leftMargin: 10
      right: parent.right
      rightMargin: 10
    }
  }
}
```