INNOVATION MAKES SENSE

Acquisition de signaux physiologiques

Application en C++

BENSALAH YASMINE
DARDE LUCIE

MAI 2020
I2 **BIOMEDICAL**

TABLE DES MATIERES

I. Rappel du contexte et du cahier des charges.

a. Contexte.

Dans le cadre de notre projet d'acquisition de signaux physiologiques, nous avons créé un boîtier contenant un circuit électrique correspondant à un électrocardiogramme. Le boîtier contient la carte électronique et une batterie. Des capteurs relient le patient à la machine afin de récupérer des signaux physiologiques correspondant au rythme cardiaque du patient. Le boîtier traite le signal, le filtre grâce à plusieurs processus et en sortie nous avons le signal cardiaque du patient remodelé. La sortie de la carte est un signal cardiaque filtré.

La chaîne d'acquisition de notre projet est la suivante :

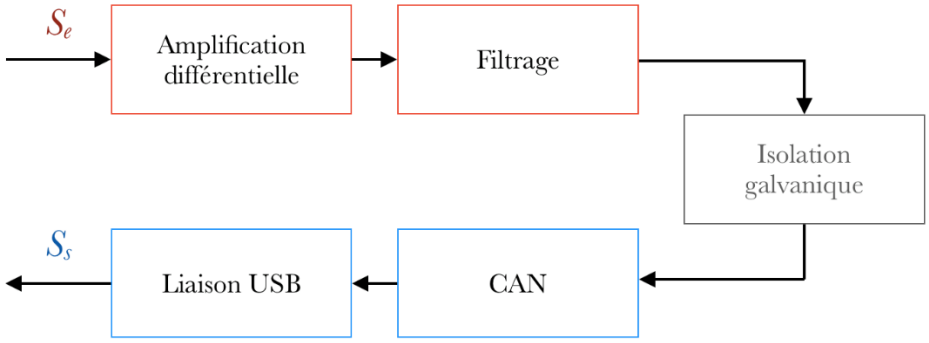
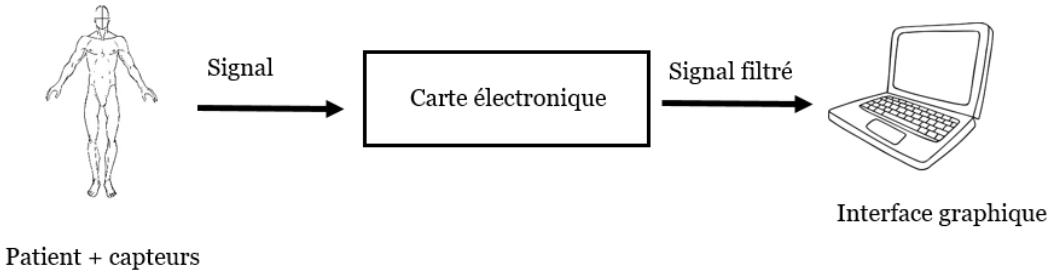


Figure 2 : Chaîne d'acquisition du projet.

En entrée de l'interface graphique, il y a donc les signaux ECG filtrés et convertis en signaux sonores par un convertisseur analogique numérique.

L'étape suivante est de créer une interface graphique en C++ permettant à l'utilisateur de visualiser les signaux ECG transmis sous forme numérique, par un câble USB, sur un ordinateur et donc de pouvoir interpréter le signal.



b. Cahier des charges.

Objectif : Créer une interface graphique en C++ sur Qt Creator pour permettre la visualisation sur ordinateur de signaux ECG.

Eléments constitutifs de l'interface graphique :

- Sélectionner la source du signal et ses caractéristiques techniques
- Démarrer, pauser et arrêter la lecture du signal
- Enregistrer sur le disque le résultat (format .wav)
- Afficher le signal en temps réel
- Calculer en temps réel la moyenne et les valeurs max puis les afficher
- Saisir et enregistrer un ensemble de données patient (nom, prénom, âge, sexe, taille, poids, zone de texte pour insérer des informations complémentaires).

Caractéristiques techniques :

- Taux d'échantillonnage : 44100 Hz
- Signal codé sur 16 bits
- Une seule voie utilisée
- Codec audio : audio/PCM

II. Environnement logiciel, librairies utilisées et contraintes.

a. Environnement logiciel.

Pour coder l'interface graphique, nous utilisons un environnement de développement intégré multiplateformes **Qt Creator** faisant partie de l'infrastructure logicielle de Qt. Qt est un ensemble de bibliothèques multiplateforme servant à créer des programmes utilisant des fenêtres, qui nous permettra de coder l'interface graphique attendue.

Qt Creator comprend :

- Un environnement de développement pour coder en C++
- Un éditeur de fenêtres
- Une documentation complète

Nous codons donc sur Qt Creator, en langage C++ (langage orienté objet).

b. Libraires utilisées.

Les librairies sont des collections de fragments de programmes préfabriqués, des classes, qui nous permettent de coder plus rapidement. Dans le cadre de ce projet, nous avons utilisé différentes librairies.

Pour ajouter une librairie on ajoute au code : #include<…>. Cela spécifie que notre application sera une instance de cette classe.

Nous avons également utilisé des modules Qt : **Audiodevices et Audiorecorder**.

Chaque classe de notre projet est constituée d'un fichier d'en-tête (.h) qui contiennent les prototypes de différentes fonctions structures et classes et d'un fichier source (.cpp) qui contiennent l'implémentation des différentes méthodes définies dans les fichiers d'en-tête. Les fichiers d'en-tête et les fichiers sources peuvent utiliser des librairies. Celles que nous avons utilisé sont les suivantes :

Partie audiodevices

Fichier.h

QaudioDeviceInfo	Cette classe fournit une interface pour questionner les appareils Audio et leurs fonctionnalités.
QMainWindow	Fournit une fenêtre d'application principale.
QObject	Classe de base de tous les objets Qt.

Partie audiolevel

Fichier.h

QWidget	Classe de base de tous les objets d'interface utilisateur.

Fichier.cpp

QPainter	Propose des dispositifs de peinture.

Partie audiorecorder

Fichier.h

QMainWindow	Fournit une fenêtre d'application principale.
QMediaRecorder	Utilisée pour l'enregistrement de contenus medias.
QUrl	Fournit une interface adaptée pour travailler avec les URL.

Fichier.cpp

QtWidgets/QApplication	Gestion du contrôle des principaux paramètres de l'application.
QtWidgets/MessageBox	Fournit une boîte de dialogue pour informer ou questionner l'utilisateur et recevoir la réponse.
QAudioProbe	Permet à l'utilisateur de lancer l'enregistrement ou de l'écouter.
QAudioRecorder	Utilisée pour enregistrer un audio.
QDir	Fournit l'accès à des annuaires et leur contenu.
QFileDialog	Permet à l'utilisateur de sélectionner des fichiers et des répertoires.
QMediaRecorder	Utilisée pour enregistrer un média.
QtWidgets	Classe de base de tous les objets d'interface utilisateur.
QMultmedia/QAudioDeviceInfo	Fournit une interface pour lancer des audio et accéder leurs fonctionnalités.

Partie Form_patient

Fichier.h

QDialog	Classe de base des fenêtres de dialogues.

QtWidgets/QWidget	Classe de base de tous les objets d'interface utilisateur.
QtChart/QChartGlobal	Permet de créer des interfaces utilsateurs soignées, interactives et centrées sur les données.

Fichier.cpp

QFile	Fournit une interface pour lire et écrire sur des fichiers.
QTextStream	Fournit une interface pour lire et écrire du texte.
QtDebug	Permet de débuguer.

Partie Xyseriesiodevice

Fichier.h

QtCore/QIODevice	Interface de base de tous les appareils I/O.
QtCore/QPointF	Définit un point dans le plan en utilisant la précision de points flottants.
QtCore/QVector	Modèle fournissant un tableau dynamique.
QtCharts/QChartGlobal	Modèles fournissant un tableau dynamique.

Fichier.cpp

Qlabel	Fournit un texte ou une image.

Partie Widget

Fichier.cpp

QtMultmedia/QAudioDeviceInfo	Fournit une interface pour lancer des audio et accéder leurs fonctionnalités.
QtMultmedia/QAudioInput	Fournit une interface recevant des données audio à partir d'un dispositif d'entrées audio.
QtWidgets/QApplication	Gestion du contrôle des principaux paramètres de l'application.
QtWidgets/MessageBox	Fournit une boîte de dialogue pour informer ou questionner l'utilisateur et recevoir la réponse.
QtCharts/QLineSeries	
QtCharts/QChart	Gère les représentations des graphiques.

QtCharts/QValueAxis	Ajoute une valeur à un axe.

Main.cpp

QFile	Fournit une interface pour lire et écrire sur des fichiers.
QTextStream	Fournit une interface pour lire et écrire du texte.
QtDebug	Permet de débuguer.
QString	Fournit une chaîne de caractères Unicode.

c. Contraintes.

Pour ce projet nous avons listé les contraintes suivantes :

- **Délai** : Etant donné que l'apprentissage de ce langage a débuté il y a quatre mois, nous devons donc dû approfondir nos connaissances et en peu de temps et être efficaces afin de respecter le délai.

- **Qualité** : De par l'aspect médical du projet, le signal affiché doit-être propre et facile à manipuler pour pouvoir être interprétable à l'œil nu.

- **Manipulation des librairies** : Qt propose de nombreuses librairies, il faut donc choisir les plus pertinentes pour ce projet.

- **Confinement** : Dû au confinement, nous ne pouvons pas récupérer le signal sortant de l'ECG réalisé en première partie de ce projet, nous allons donc traiter un signal sonore tel qu'un claquement de main enregistré à partir de l'ordinateur.

- **Installation de Qt Creator** : L'installation de Qt Creator est assez lourde, et cela a pris un grand nombre d'essais.

III. Description fonctionnelle et schématique de l'architecture logicielle.

a. Diagramme de classe UML.

Voici le diagramme de classe UML représentant le projet.

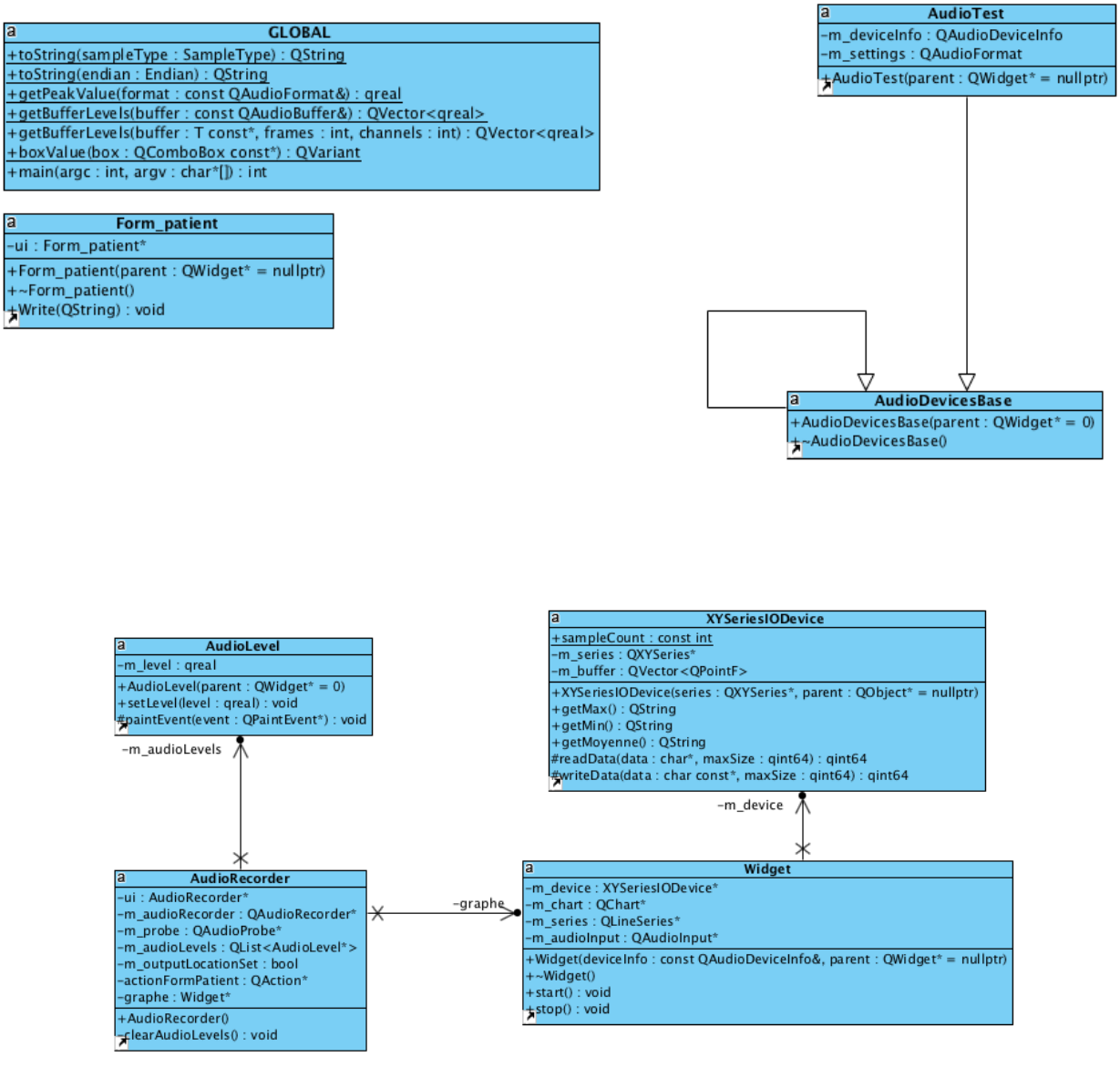


Figure 3 : Diagramme de classes.

b. Diagramme d'activités de l'interface homme/machine.

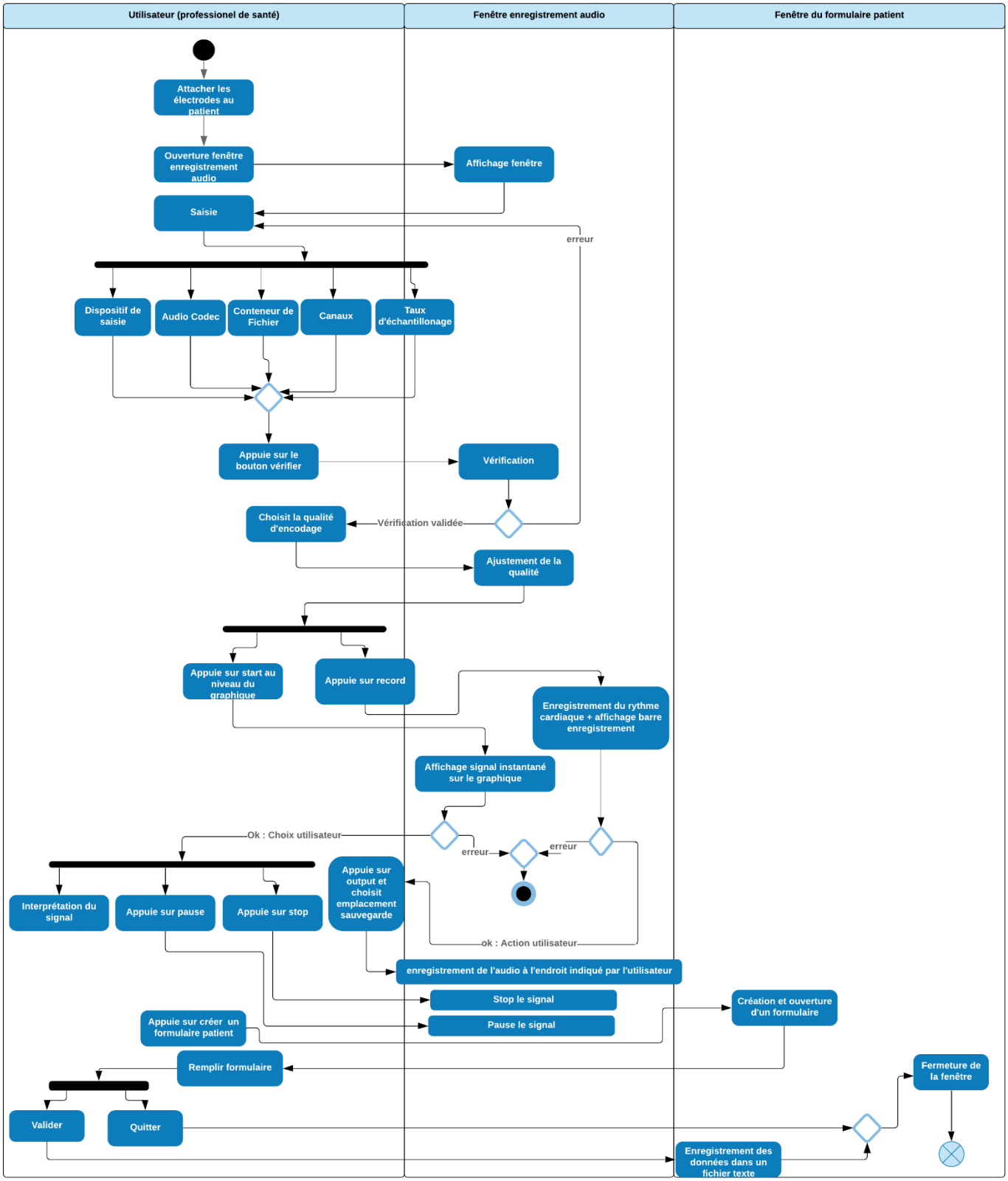


Figure 4 : Diagramme d'activités du système.

IV. Description détaillée de l'implémentation.

a. Audio Devices

Afin de réaliser ce projet, nous nous sommes basées sur un exemple fournit par Qt qui permet de tester les périphériques audio disponibles et leur configuration.

Le code des fichiers fournis créé une interface simple pour répertorier et tester la configuration (Codec, Fréquence, Canaux, Type d'échantillons, Bits) des différents périphériques audio disponibles sur le périphérique cible. Les fichiers sont les suivants :

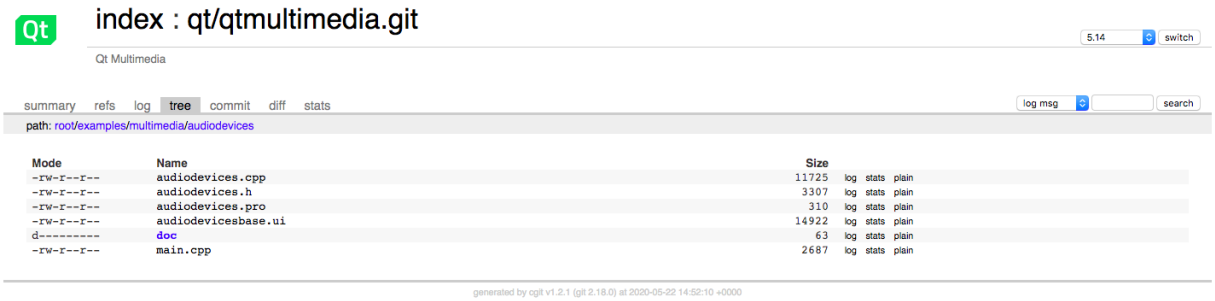


Figure 5 : Fichiers exemple Qt.

Le résultat de audiodevices dans notre projet (audiodevicesbase.ui) est le suivant :

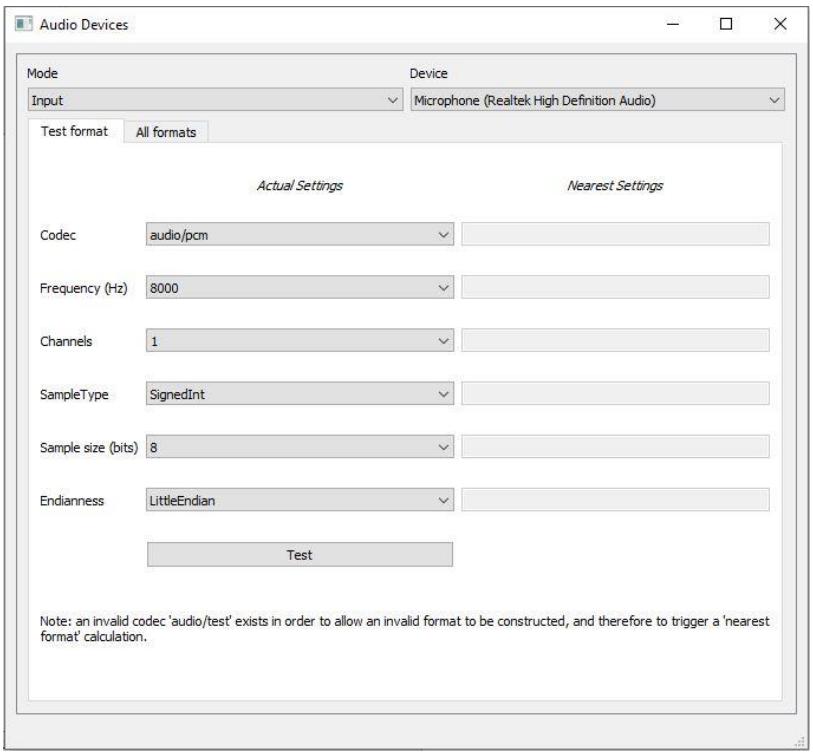


Figure 6 : Interface audiodevices.

✓ Cahier des charges : Sélectionner la source du signal et ses caractéristiques.

b. Audiorecorder

Pour assurer l'identification des périphériques audio disponibles, les codecs compatibles (dispositif matériel ou logiciel permettant l'encodage ou le décodage d'un flux de données numériques, en vue d'une transmission ou d'un stockage) et l'utilisation de la classe AudioRecorder (venant de la librairie Qt).

Ce code affiche une fenêtre permettant à l'utilisateur de sélectionner l'entrée audio, le codec, le conteneur et la fréquence d'échantillonnage appropriés (voir annexes page 23).

Afin de faire évoluer le code pour que la fenêtre corresponde au cahier des charges nous avons ajouté les fonctionnalités à travers de nouveaux fichiers d'en tête et sources ; **widget, xyseriesiodevice et audiolevel**.

1. Widget.

Il permet d'afficher le graphique représentant le signal sonore (l'ECG).

Dans le fichier.h, nous avons d'abord inclus les classes Qt dont nous avons besoin, dont la classe **Qwidget**. Ce module fournit un ensemble d'éléments d'interfaces utilisateurs pour créer des interfaces utilisateurs classiques de bureau.

On a défini les classes Qt QLineSeries, QChart, QAudioInput et QAudioDeviceInfo dans des name space pour réduire la lourdeur du code.

On a créé une nouvelle classe Widget de type public, pour qu'elle soit accessible par les autres fichiers et déclaré les fonctions souhaitées :

- Fonctions publiques :
  - Widget(const QAudioDeviceInfo &deviceInfo, QWidget *parent = nullptr) ;
    - ➔ Fonction qui permet de générer les principaux éléments de création de l'interface utilisateur (données, informations d'état, recevoir des entrées utilisateurs et fournir un conteneur).
  - start() : void
    - ➔ Fonction qui permet de lancer l'enregistrement du signal.
  - stop() : void
    - ➔ Fonction qui permet d'arrêter l'enregistrement du signal.

- Instanciations privées (création d'objets dynamiques) :
  - XYSeriesIODevice *m_device = nullptr ;
    - ➔ ne pointe pas vers un objet.
  - QChart *mchart ;
    - ➔ Pointeur objet de type QChart.
  - QLineSeries *m_series ;
    - ➔ Pointeur objet de type QLineSeries.
  - QAudioInput *m_audioInput = nullptr ;

➔ Pointeur de la classe QAudioInput ne pointe pas vers un objet.

- Slots publics (communication entre objets)
  - UpdateTitle() : void
  - SetPause() : void
  - SetStop() : void
  - SetStart() : void

Toutes les classes contenant des slots doivent être précédées de la mention Q_OBJECT
Dans le fichier widget.cpp, on a implémenté les fonctions présentées dans la partie widget.h. Concernant, ce fichier, on a utilisé plusieurs classes Qt après les avoir importées (voir annexes page 23).

QtMultimedia -> Ce module contient un ensemble complet de fonctionnalités permettant de profiter facilement des capacités multimédias d'une plateforme (pour ce projet on utilise la lecture multimédia).

QtCharts -> Ce module fournit un ensemble de composants graphiques faciles à utiliser.

QtWidgets -> Ce module fournit un ensemble d'éléments d'interface utilisateur pour créé des interfaces utilisateur classiques de bureau.

```
QT_CHARTS_USE_NAMESPACE

Widget::Widget(const QAudioDeviceInfo &deviceInfo, QWidget *parent) :
    QWidget(parent),
    m_chart(new QChart),
    m_series(new QLineSeries)
{
```

La fonction Widget regroupe les éléments permettant de créer l'interface graphique pour voir le signal et choisir ses paramètres. Elle prend en argument l'adresse d'un objet deviceInfo de la classe QAudioDeviceInfo (qui permet de créer une interface pour interroger les périphériques audio et leurs fonctionnalités et qui fait partie du module QtMultimedia). Comme on a vu dans le fichier .h, la classe widget hérite de la classe QWidget, le deuxième argument est donc son widget parent (fenêtre de second plan car le pointeur est non nul).

m_chart est un objet de la classe Qchart et m_series de la classe QLineSeries que nous utilisons plus bas.

```
    QChartView *chartView = new QChartView(m_chart);
```

Permet de créer une fenêtre graphique pour afficher le graphe du signal sonore entrant.

```
chartView->setMinimumSize(600, 300); // changer la taille de l'audio record
```

Permet de changer la taille du graphique affichant le signal ECG.

```
m_chart->addSeries(m_series);
```

Permet de créer des données (pour les axes du graphique affichant le signal ECG mais à ce stade, elles ne sont pas encore définies).

```
QValueAxis *axisX = new QValueAxis;
axisX->setRange(0, XYSeriesIODevice::sampleCount);
axisX->setLabelFormat("%g");
axisX->setTitleText("Samples");
```

Cette partie permet de configurer et de créer l'axe X du graphique qui se nommera « Samples » et qui représente le nombre d'échantillons.

```
QValueAxis *axisY = new QValueAxis;
axisY->setRange(-1, 1);
axisY->setTitleText("Audio level");
```

Cette partie permet de configurer et de créer l'axe Y du graphique qui se nommera « Audio level », allant de 0 à 1 et représentant l'amplitude.

```
m_chart->addAxis(axisX, Qt::AlignBottom);
m_series->attachAxis(axisX);
m_chart->addAxis(axisY, Qt::AlignLeft);
m_series->attachAxis(axisY);
m_chart->legend()->hide();
```

Permet d'affecter les axes sur le graphique et de ne pas mettre de légende.

```
QVBoxLayout *mainLayout = new QVBoxLayout(this);
mainLayout->addWidget(chartView);
```

La classe QVBoxLayout permet de construire des objets ayant la forme de cases disposées verticalement. Ici, on l'utilise pour le graphique.

```
QAudioFormat formatAudio;
formatAudio.setSampleRate(8000);
formatAudio.setChannelCount(1);
formatAudio.setSampleSize(8);
formatAudio.setCodec("audio/pcm");
formatAudio.setByteOrder(QAudioFormat::LittleEndian);
formatAudio.setSampleType(QAudioFormat::UnSignedInt);
```

On créé un objet de type QAudioFormat auquel on paramètre le taux d'échantillonnage, le canal, la taille de l'échantillon, le codec, l'ordre des bits et le type d'échantillonnage.

```
m_audioInput = new QAudioInput(deviceInfo, formatAudio, this);
m_device = new XYSeriesIODevice(m_series, this);
m_device->open(QIODevice::WriteOnly); //écrire sur l'affichage

m_audioInput->start(m_device); // demarre en utilisant les données donné par la sortie de l'input
//TIMER
QTimer *t = new QTimer(this);
connect(t,SIGNAL(timeout()),this,SLOT(UpdateTitle()));
t->start(200);
```

Cette dernière partie permet de prendre en entrée un format audio puis d'écrire sur la partie titre les valeurs de max,min et moyenne. Ensuite, on configure l'entrée du système pour lire le fichier audio en input et on configure l'horloge, pour permettre de mettre à jour l'affichage du titre toutes les 200ms avec la fonction UpdateTitle() décrite ce-dessous.

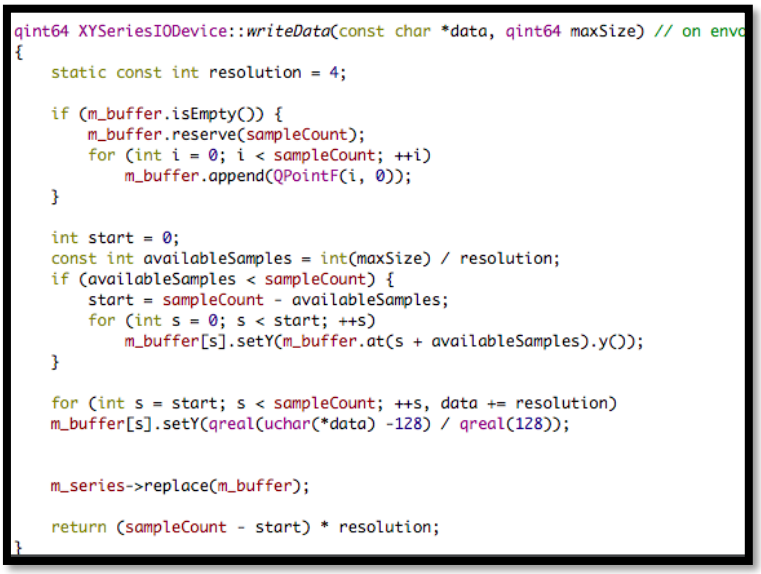
```
void Widget::setStart(){
    this->m_audioInput->start();
}

void Widget::setPause(){
    this->m_audioInput->suspend();
}

void Widget::setStop(){
    this->m_audioInput->stop();
}

void Widget::UpdateTitle(){
    QString Max=m_device->getMax();
    QString Min=m_device->getMin();
    QString Moyenne=m_device->getMoyenne();
    m_chart->setTitle("Le maximum : "+Max+", le minimum : "+Min+", la moyenne : "+ Moyenne);
}
```

Enfin, la dernière partie du code contient les fonctions qui permettent de démarrer le signal, le mettre sur pause et enfin l'arrêter grâce à l'affectation du signal m_audioInput aux fonctions start(), suspend() et stop(). UpdateTitle() permet de récupérer les valeurs du maximum, du minimum et la moyenne de l'amplitude du signal et de les afficher en temps réel (ces fonctions ont été implementées dans les fichiers xyseriesiodevice). Cela permet également d'afficher au-dessus du graphique, les valeurs.

2. Xyseriesiodevice

Le fichier .h définit le prototype des fonctions. Le fichier .cpp contient l'implémentation des fonctions pour calculer le maximum, le minimum et la moyenne du signal (getMax(), getMin(), getMoyenne()). Il permet également d'envoyer le signal en input sur le graphique et de l'afficher (readData() ; writeData()).

Nous allons nous concentrer sur les parties les plus importantes du code :



```
qint64 XYSeriesIODevice::writeData(const char *data, qint64 maxSize) // on envo
{
    static const int resolution = 4;

    if (m_buffer.isEmpty()) {
        m_buffer.reserve(sampleCount);
        for (int i = 0; i < sampleCount; ++i)
            m_buffer.append(QPointF(i, 0));
    }

    int start = 0;
    const int availableSamples = int(maxSize) / resolution;
    if (availableSamples < sampleCount) {
        start = sampleCount - availableSamples;
        for (int s = 0; s < start; ++s)
            m_buffer[s].setY(m_buffer.at(s + availableSamples).y());
    }

    for (int s = start; s < sampleCount; ++s, data += resolution)
    m_buffer[s].setY(qreal(uchar(*data) -128) / qreal(128));


    m_series->replace(m_buffer);

    return (sampleCount - start) * resolution;
}
```

La méthode **write data** prend en entrée les données (le signal audio) et retourne en sortie les points sur le graphique. SampleCount correspond au nombre d'élément qu'on lit à chaque fois (2000).
S'il n'y a pas de données entrantes, sur le graph on affiche un point d'abscisse i (le numéro du point manquant) et d'ordonnée 0. Sinon, on place les points sur l'axe des ordonnées en prenant compte de la résolution du graphique (en remettant à la bonne échelle).

```
double moyenne = 0.0;
double total = 0.0;
```

```
QString XYSeriesIODevice::getMoyenne()
{
    for (auto elem : m_buffer)
    {
        total+=qAbs(elem.y());
        moyenne=total/(m_buffer.length());
    }

   QString moyenneString = QString::number(moyenne,'f',2);
    return moyenneString;
}
```

Principe de la fonction : On collecte chaque valeur de m_buffer (les données représentant le signal) qu'on additionne dans la variable totale. On divise la valeur totale de tous les éléments, divisé par le nombre d'éléments. On retourne la moyenne sous forme de QString.

```
QString XYSeriesIODevice::getMin()
{ double min = 0.0;
    for (auto elem : m_buffer)
    {
        if (elem.y() > min) min=elem.y();
    }
   QString minString = QString::number(min,'f',2);
    return minString;
}
```

Principe de la fonction : On compare une à une les valeurs et on retourne la plus petite sous forme de QString.

```
QString XYSeriesIODevice::getMax()
{ double max = 0.0;
    for (auto elem : m_buffer)
    {
        if (elem.y() > max) max=elem.y();
      //  qDebug()<<max;

    }
   QString maxString = QString::number(max,'f',2);
    //qDebug()<<maxString;
    return maxString;
}
```

Principe de la fonction : On compare une à une les valeurs et on retourne la plus grande sous forme de QString.

3. Audiolevel.

Permet d'afficher des données audio en temps réel. Ces fichiers font partie du module Audiorecorder de Qt.

Audiorecorder.ui

Les fichiers audiolevel, widget, audiorecorder, et xyseriesodevices permettent donc de parvenir au résultat suivant :

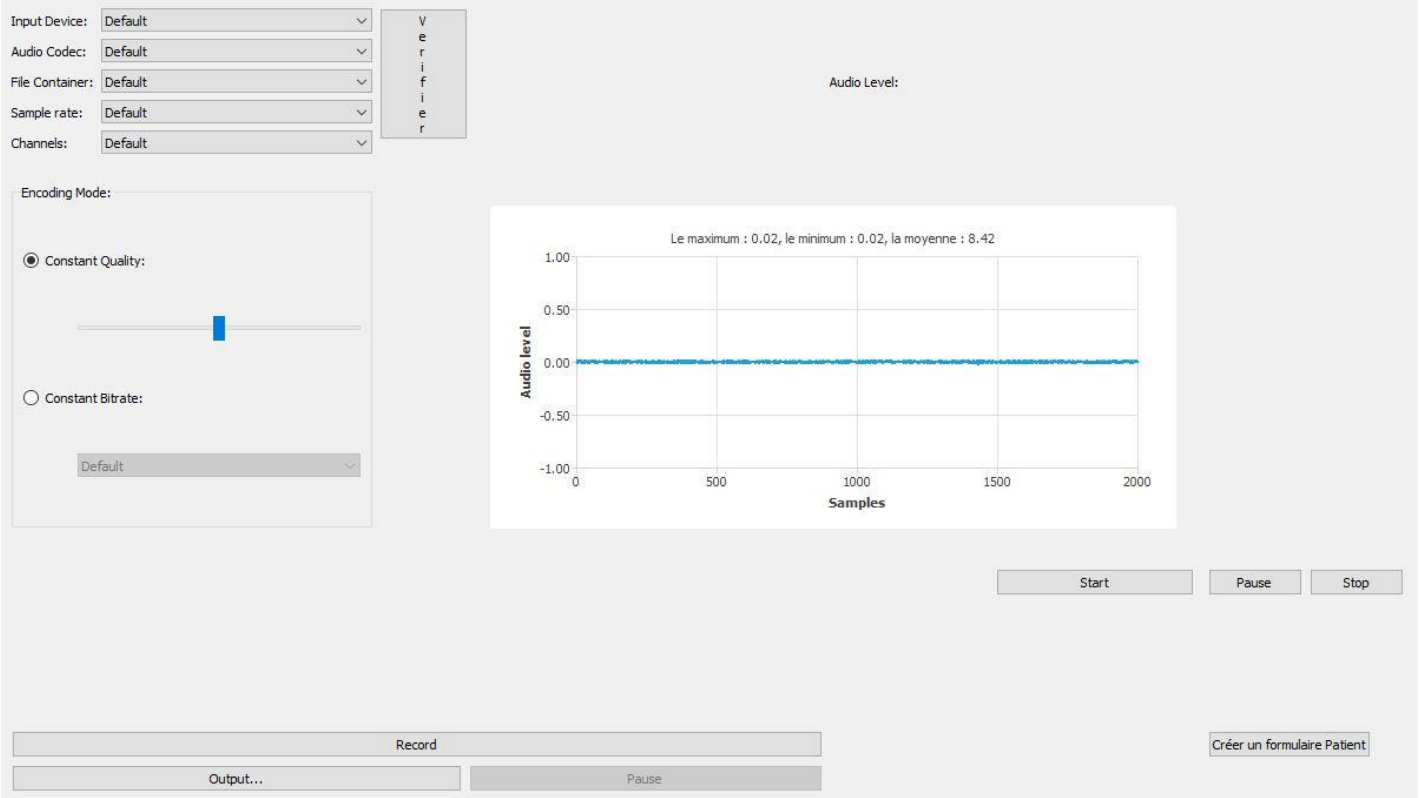


Figure 7 : Interface affichant les signaux récupérés.

Cahier des charges :

- ✓ Démarrer, pauser et arrêter la lecture du signal
- ✓ Afficher le signal en temps réel
- ✓ Calculer en temps réel la moyenne et les valeurs max puis les afficher

c. *Form_patient.*

Form_patient contient les fonctions nécessaires à la création d'un formulaire.

Dans le fichier recorder.cpp, on a déjà implémenté les deux fonctions suivantes :

```
//fonction qui permet de créer une fenetre de formulaire patient
void AudioRecorder::creerFenetreFormPatient(){
    Form_patient *formulaire = new Form_patient();
    formulaire->show();
}
```

```
  //Creation du bouton pour créer la fenetre le formulaire du patient
  QPushButton * creationFormPatient = new QPushButton("Créer un formulaire Patient",ui->creerFormulairePatient);
  connect(creationFormPatient, SIGNAL(clicked()), this, SLOT(creerFenetreFormPatient()));
  // quand on clique la fonction creerFenetreFormPatient() est lancée
```

La première fonction implémentée dans le fichier Form_patient.cpp est la suivante :

```
Form_patient::Form_patient(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Form_patient)
{
    ui->setupUi(this);
    QPushButton * validerFormulaire = new QPushButton("Valider formulaire",ui->ValideFormulaire);
    connect(validerFormulaire, SIGNAL(clicked()), this, SLOT(CreationFormPatient()));

}
```

La fonction Form_patient permet de créer un bouton poussoir nommé « Valider formulaire ». Lorsque l'utilisateur appuie sur valider le formulaire, il y a la validation du formulaire grâce à la fonction connect qui va faire le lien entre l'action de cliquer sur le bouton « Valider Formulaire » et créer le formulaire patient (sauvegarder les informations en fichier texte). Une fois le formulaire validé, la fenêtre se ferme grâce à la fonction close().

On a ensuite implémenté une fonction Write qui permet de générer le fichier texte avec les informations entrées par l'utilisateur, valeurs de out dans le code. Cette fonction permet de générer un fichier .txt à partir des données que l'utilisateur va entrer dans le formulaire.

```
void Form_patient::Write(QString Filename) // Write() permet de générer un fichier .txt,
                                           //avec comme contenu la valeur de out
{
    QFile mFile(Filename);

    if(!mFile.open(QFile::WriteOnly | QFile::Text))
    {
        qDebug() << "erreur";
        return;
    }

    QTextStream out(&mFile);

    out <<" Nom : "<< ui->Nom->text() <<"\r\n Prénom : "<< ui->Prenom->text()
       << "\r\n Date de naissance : " << ui->Jour->currentText()
       <<"/"<<ui->Mois->currentText()<<"/"<<ui->Annee->currentText()
      <<"\r\n Taille : "<<ui->Taille->text()
      <<" cm \r\n Poids : "<<ui->Poids->text()<<" Kg";
    // sur le fichier .txt sera écrit toutes les données du out

    mFile.flush();
    mFile.close();
}
```

La dernière étape a été de créé le fichier texte en utilisant la fonction Write dans la fonction CreationFormPatient() puis ferme la fenêtre Fiche Patient. Ici le fichier .txt sera enregistré à l'adresse : "C:/Users/Lucie Darde/Desktop/myfile.txt"

```
void Form_patient::CreationFormPatient(){ // on creer le fichier.txt
    QString mFilename="C:/Users/Lucie Darde/Desktop/myfile.txt";
     Write(mFilename);
     this->close(); //on ferme la fenetre une fois validé
```

Lorsqu'on appuie sur créer un formulaire patient, nous avons :

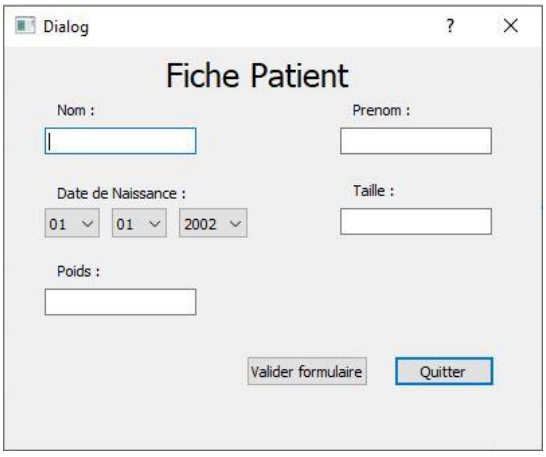


Figure 8 : Formulaire patient obtenu à l'issu du programme.

Cahier des charges :

- ✓ Saisir et enregistrer un ensemble de données patient (nom, prénom, âge, sexe, taille, poids, zone de texte pour insérer des informations complémentaires).

V. Difficultés rencontrées, ce qu'il manque, pistes à explorer, conclusion

a. Difficultés rencontrées

Nous avons eu du mal à comprendre comment on pouvait relier Form_patient et audiorecorder. Ensuite nous avons rencontrer des difficultés sur la partie de création du fichier .txt à partir de données collectées dans des widgets.
Enfin nous avons eu une dernière difficulté à la quelle nous n'avons pas trouvé de solution aboutie ce sont de relier les slots setStart(), setPause() et setStop() au boutons.

```
// Connection des slots aux BP
connect(ui->Pause, SIGNAL(clicked()), graphe, SLOT(Widget::setPause()));
connect(ui->Stop, SIGNAL(clicked()), this, SLOT(Widget::setStop()));
connect(ui->Start, SIGNAL(clicked()), this, SLOT(Widget::setStart()));
```

b. A faire

- Faire fonctionner le setStart(), setPause() et setStop()
- Une fois que le bouton Pause fonctionne, appuyer à nouveau dessus pour redémarrer la lecture du signal.
- Essayer l'application avec un signal physiologique.

c. Pistes à explorer

- Le secteur médical est exposé à la protection des données, il faudrait donc trouver une façon de protéger voire crypter les données récupérées par le formulaire.
- Lié automatiquement le formulaire patient créer avec l'ECG enregistré

VI. Bibliographie

Sites internet :
BRAUN, N. (2016). Utiliser QT Designer : Les formulaires. Consulté le 24 avril 2020, à l'adresse http://www.siloged.fr/cours/QTCreator/QTDesignerLesformulaires.html

5 - Données attributaires - Création de formulaire avec QtCreator. (s. d.-a). Consulté le 12 mai 2020, à l'adresse https://data.sigea.educagri.fr/download/sigea/supports/QGIS/distance/perfectionnement/M06_Donnees_attributaires_gen_web/co/10_N2_Edition_Interface.html

5 - Données attributaires - Création de formulaire avec QtCreator. (s. d.-b). Consulté le 5 mai 2020, à l'adresse https://data.sigea.educagri.fr/download/sigea/supports/QGIS/distance/perfectionnement/M06_Donnees_attributaires_gen_web/co/10_N2_Edition_Interface.html?fbclid=IwAR0sZBsT_zK6rzIlmBDeV05COfTJvwA-fn9NPyqut7WRdSoPWra614dA5JQ

Company, T. Q. (s. d.). Qt | Cross-platform software development for embedded & desktop. Consulté à l'adresse https://www.qt.io/

Nebra, M. (s. d.). Programmez avec le langage C++. Consulté à l'adresse https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c
QGis-QTCreator : Créer son formulaire dans QGis. (2014, mai 3). Consulté le 28 avril 2020, à l'adresse https://archeomatic.wordpress.com/2012/03/06/qgis-qtcreator-creer-son-formulaire-dans-qgis/

Wikipedia contributors. (2019, mai 26). Qt Creator — Wikipédia. Consulté à l'adresse https://fr.wikipedia.org/wiki/Qt_Creator

Wikipedia contributors. (2020, mai 16). Qt — Wikipédia. Consulté à l'adresse https://fr.wikipedia.org/wiki/Qt

Vidéos :
sociamix. (2018, avril 8). Créer des applications graphiques avec Qt (C++) [Fichier vidéo]. YouTube. Consulté à l'adresse https://www.youtube.com/watch?v=050zzD4c-5c&t=1098s

VII. Annexes

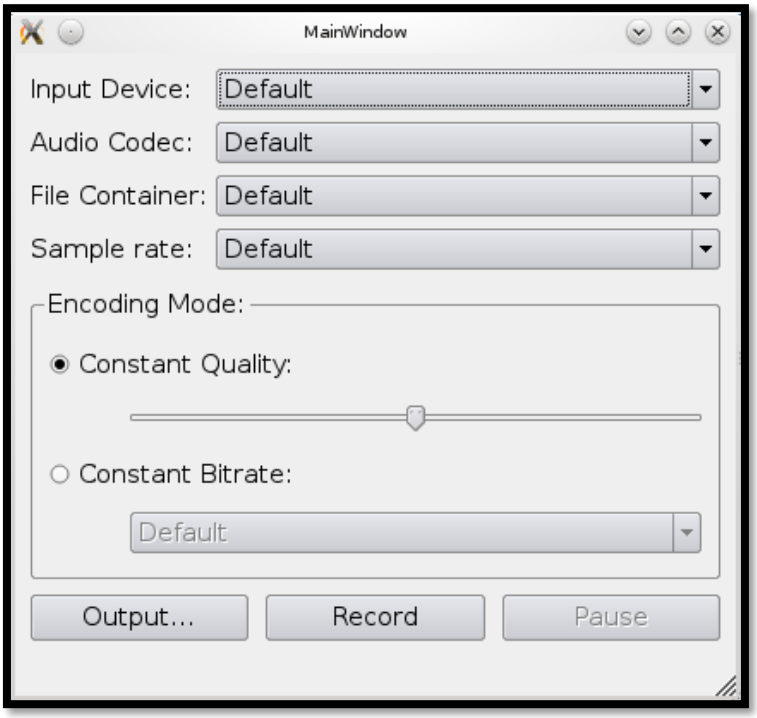


Figure 9 : Résultat de l'exemple Qt audiodevices.

Bibliothèques utilisée pour la classe Widget.

```
#include <QtCharts/QChartView>
#include <QtCharts/QLineSeries>
#include <QtCharts/QChart>
#include <QtCharts/QValueAxis>
#include <QTimer>

#include <QtWidgets/QVBoxLayout>

#include "widget.h"
#include "xyseriesiodevice.h"
#include "ui_audiorecorder.h"

#include <QtMultimedia/QAudioDeviceInfo>
#include <QtMultimedia/QAudioInput>
#include <QtMultimedia/QAudioDeviceInfo>
#include <QtWidgets/QApplication>
#include <QtWidgets/QMessageBox>
```

XIII. Table des figures.