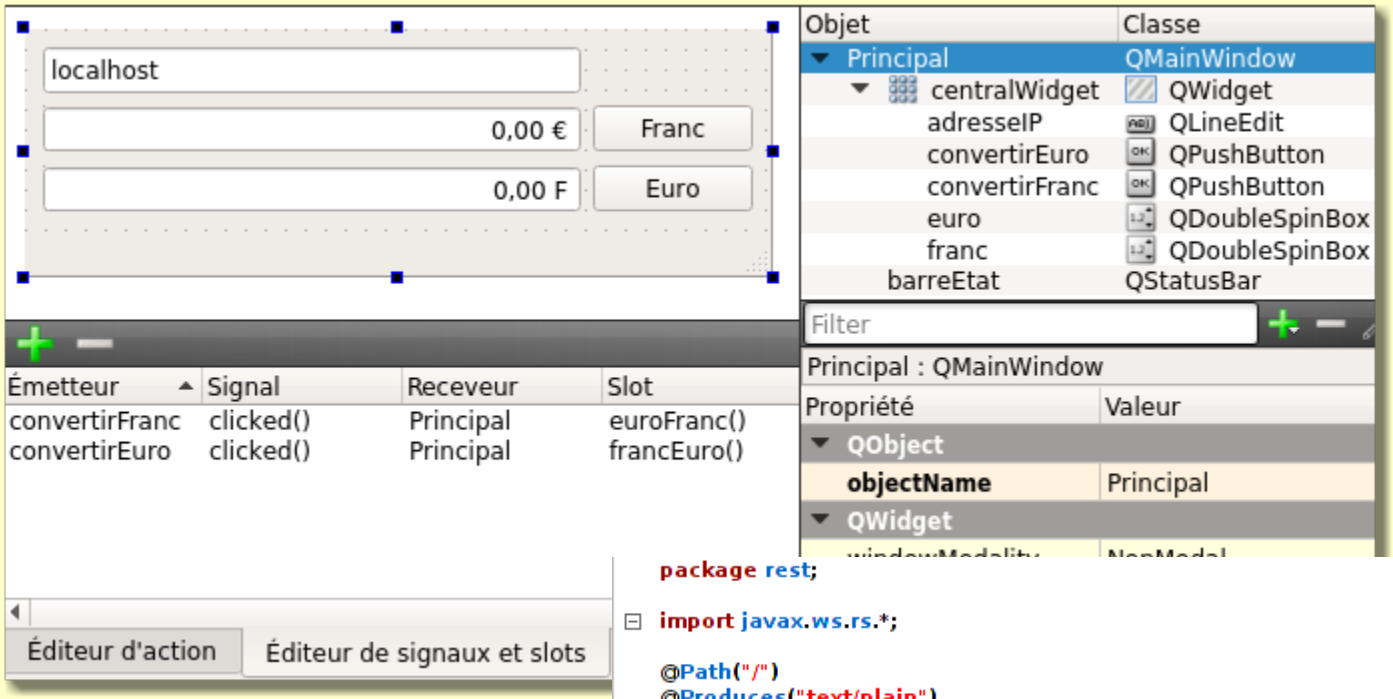
Lors de cette étude, nous allons voir comment communiquer avec des web services codés en Java. Il s 'agit ici des web service de type REST, c'est-à-dire des services qui permette de gérer des ressources à distance.

> *Je rappelle que les services Web sont conçus comme d'autres services, avec toutefois la particularité de transmettre les requêtes et les réponses tout simplement à l'aide du protocole **HTTP**. Du coup, notre pare-feu n'a pas besoin d'une configuration particulière et nous protège en nous laissant passer que le port **80**.*
>
> *Nous allons élaborer deux projets différents qui vont nous permettre de découvrir graduellement l'utilisation des méthodes offertes par le protocole **HTTP**, savoir les méthodes **GET**, **POST**, **PUT** et **DELETE**.*

- ***GET :** cette méthode permet de récupérer une ressource depuis le serveur pour l'avoir sur le poste local.*
- ***POST :** cette méthode est l'inverse de la précédente, cette fois-ci nous envoyons une ressource depuis le poste local. Elle permet ainsi de sauvegarder à distance une ressource que nous possédons sur notre ordinateur.*
- ***PUT :** cette méthode permet de modifier une ressource distante.*
- ***DELETE** : cette méthode permet, comme sont non l'indique de supprimer une ressource du serveur.*

PREMIER PROJET : CONVERSION ENTRE LES €UROS ET LES FRANCS

Le premier projet consiste à communiquer avec un Web Service qui permet de faire la conversion entre les €uros et les francs. Dans cet exemple, la communication est très simple puisque nous utilisons uniquement la méthode HTTP GET, qui renvoie les valeurs sous forme de chaînes de caractères.




```
package rest;

import javax.ws.rs.*;

@Path("/")
@Produces("text/plain")
public class Conversion {
  private final double TAUX =6.55957;

  @Path("franc")
  @GET
  public String euroFranc(@QueryParam("euro") double euro) {
    return (euro * TAUX) + "";
  }

  @Path("euro")
  @GET
  public String francEuro(@QueryParam("franc") double franc) {
    return (franc / TAUX) + "";
  }
}
```

CREATION DU WEB SERVICE REST

Le web service est réalisé en Java. Je vous donne juste le code nécessaire pour comprendre comment communiquer avec lui, sans explication particulière, parce que ce n'est pas le sujet ici. Toutefois, il est intéressant de connaître exactement quelles sont les formes des deux requêtes que nous pouvons proposer afin d'obtenir les résultats requis. Voici les deux types d'exemple ci-dessous :

- ***http://localhost:8080/franc?euro=15.24 : retourne une valeur en Franc à partir d'une valeur donnée en €uro.***
- ***http://localhost:8080/euro?franc=100 : retourne une valeur en €uro à partir d'une valeur donnée en Franc.***

CLASSES UTILISÉES POUR COMMUNIQUER AVEC UN WEB SERVICE REST

La communication avec un Web service REST est extrêmement facile à réaliser avec une application cliente développée avec la librairie QT. En effet, QT propose uniquement trois classes pour résoudre toutes les situations possibles :

- **QNetworkAccessManager :** C'est au travers de cette classe que nous établissons la communication avec le service Web distant. Je rappelle que dans la philosophie de QT, la communication réseau se fait au travers d'une gestion événementielle. Ainsi, à chaque fois qu'une requête sera proposée, un événement sera automatiquement lancé dès que la réponse sera prête. Cette classe est également spécialisée dans la communication au travers du protocole HTTP. Ainsi, elle possède des méthodes toutes prêtes pour traduire tous les souhaits de l'utilisateur, grâce notamment aux méthodes : **get(), post(), put()** et **deleteResource()** qui font appel aux méthodes respectives du **protocole HTTP : GET, POST, PUT** et **DELETE**.
- **QNetworkRequest :** Comme sont nom l'indique, cette classe nous permet d'élaborer les différentes requêtes requises en proposant à chaque fois la bonne **URL** en adéquation avec ce que souhaite le **web service REST**. L'objet ainsi créé servira d'argument à l'une des méthodes précédentes - **get(), post(), put()** et **deleteResource()** - de la classe **QNetworkAccessManager**.
- **QNetworkReply :** Cette classe représente la réponse à la requête sollicité par **QNetworkRequest**. En réalité, l'objet de cette classe est un paramètre d'une méthode (**SLOT**) qui sera automatiquement appelée lorsque effectivement une réponse sera reçue du **service Web REST** (gestion événementielle). Bien entendu, cette classe dispose de méthodes adaptées à la gestion du résultat, avec notamment : la méthode **error()** qui nous prévient si la réponse a été correctement envoyée, la méthode **readAll()** qui nous retourne la totalité du contenu espéré, la méthode **readLine()** qui récupère le texte reçu ligne par ligne, **canReadLine()** qui teste si il existe encore une ligne de texte à lire, etc.

FICHIER DE PROJET – CONVERSIONREST.PRO

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-01-12T10:30:24
#
#-------------------------------------------------
QT       += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
CONFIG   += c++11
TARGET    = ConversionREST
TEMPLATE  = app
SOURCES  += main.cpp principal.cpp  rest.cpp
HEADERS  += principal.h   rest.h
FORMS    += principal.ui
```

Dans ce fichier de projet, n'oubliez pas d'intégrer la librairie propre à la programmation réseau. Par ailleurs, si vous devez écrire une syntaxe récente, pensez également à configurer QT pour prendre en compte la dernière version du langage, savoir **c++11**.

REST.H

```
#ifndef REST_H
#define REST_H

#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QnetworkReply>

class Rest : public QObject
{
    Q_OBJECT
    QNetworkAccessManager reseau;
public:
    Rest();
    void requete(const QString &adresse, const QString &demande);
signals:
    void alerte(const QString&);
    void reception(const QString&);
private slots:
    void reponse(QNetworkReply *resultat);
};

#endif // REST_H
```

Nous avons mis en œuvre une classe spécialisée pour toute la gestion du réseau et de la communication avec le **service Web REST**. Nous retrouvons :

- **reseau :** un objet de la classe **QNetworkAccessManager** qui permet d'établir la communication avec le service Web.
- **requete()** : une méthode qui permet de formaliser la requête souhaitée.
- **reponse()** : une méthode qui récupère la réponse à la requête.
- **reception()** : signal envoyé avec le contenu de la réponse.
- **alerte()** : signal envoyé au cas où un problème de communication apparaît.

REST .CPP

```
#include "rest.h"

Rest::Rest()  { connect(&reseau, SIGNAL(finished(QNetworkReply*)), this, SLOT(reponse(QNetworkReply *))); }

void Rest::requete(const QString &adresse, const QString &demande)
{
    QString url = "HTTP://";
    url+=adresse;
    url+=":8080/Conversion/";
    url+=demande;
    reseau.get(QNetworkRequest(QUrl(url)));
}

void Rest::reponse(QNetworkReply *resultat)
{
    if (resultat->error() == QNetworkReply::NoError)
    {
        QByteArray donnees = resultat->readAll();
        reception(donnees.data());
    }
    else alerte("Problème de communication !");
    delete resultat;
}
```

- **Le constructeur** met en œuvre la gestion événementielle qui permet de lancer automatiquement la méthode **reponse()** dès que le résultat a été obtenu à l'issue de chaque requête.
- **requete()** : la méthode formalise correctement la bonne **URL** afin de se connecter au bon service Web d'une part et de faire la demande souhaitée par l'application cliente. Nous passons par la classe **QUrl** pour que la chaîne de caractère soit bien formater. Nous passons ensuite par la classe **QNetworkRequest** pour envoyer notre requête avec l'URL formatée et nous utilisons l'objet **reseau** de type **QNetworkAccessManager** pour finaliser le type de requête **HTTP** souhaitée, ici la méthode **GET**.
- **reponse()** : la méthode est appelée automatiquement après chaque envoi d'une requête. Elle prend en paramètre un objet de type **QNetworkReply** qui représente le résultat reçu. Il convient de vérifier systématiquement si l'opération s'est bien déroulée en appelant la méthode **error()**. Si effectivement la communication a pue être établie, nous pouvons récupérer la totalité de la donnée envoyée par le web service à l'aide de la méthode **readAll()**. **ATTENTION !** Une fois que vous avez bien récupéré votre valeur, vous devez également systématiquement la supprimer du buffer de réception à l'aide de l'opération **delete**.

PRINCIPAL.H

```
#ifndef PRINCIPAL_H
#define PRINCIPAL_H

#include <QMainWindow>
#include "ui_principal.h"
#include "rest.h"

class Principal : public QMainWindow, public Ui::Principal
{
    Q_OBJECT
Public:
    explicit Principal(QWidget *parent = 0);
private:
    Rest rest;
    enum {AUCUN, EURO, FRANC} commande = AUCUN;
private slots:
    void euroFranc();
    void francEuro();
    void resultat(QString monnaie);
};

#endif // PRINCIPAL_H
```

PRINCIPAL.CPP

```
#include "principal.h"
#include <QmessageBox>

Principal::Principal(QWidget *parent) : QMainWindow(parent)
{
    setupUi(this);
    connect(&rest, SIGNAL(alerte(QString)), barreEtat, SLOT(showMessage(QString)));
    connect(&rest, SIGNAL(reception(QString)), this, SLOT(resultat(QString)));
}
```

```
void Principal::euroFranc()
{
    commande = FRANC;
    rest.requete(adresseIP->text(), QString("franc?euro=%1").arg(euro->value()));
}

void Principal::francEuro()
{
    commande = EURO;
    rest.requete(adresseIP->text(), QString("euro?franc=%1").arg(franc->value()));
}

void Principal::resultat(QString monnaie)
{
    switch (commande) {
    case FRANC : franc->setValue(monnaie.toDouble()); break;
    case EURO  : euro->setValue(monnaie.toDouble()); break;
    case AUCUN : barreEtat->showMessage("Choisissez votre monnaie"); break;
    }
    commande = AUCUN;
}
```

✗ DEUXIÈME PROJET : ARCHIVAGE DE PHOTO À DISTANCE

Dans ce deuxième projet, nous allons mettre en œuvre un système qui permet d'archiver un ensemble de photos à distance. À tout moment, il doit être possible de stocker une photo depuis le disque du poste local vers le serveur, ensuite à l'inverse de la récupérer, de modifier à distance le nom de la photo et pour finir de pouvoir la supprimer définitivement du serveur.

> *Grâce à ce projet, nous voyons bien que nous allons utiliser l'ensemble des méthodes usuelles du **protocole HTTP**, je le rappelle, les méthodes **GET, PUT, PUT** et **DELETE**.*

✗ REQUÊTES POSSIBLES POUR LE SERVICE WEB D'ARCHIVAGE DE PHOTOS

Nous allons le découvrir bientôt, le **service web REST** d'archivage de photos propose cinq fonctionnalités associées, bien entendu, à cinq méthodes de la classe **Archivage** qui représente ce service Web. Vous avez ci-dessous les **URLs** à proposer afin d'obtenir les requêtes souhaitées :

- ***http://localhost:8080/Archivage/ : GET : donne la liste des noms des photos stockées dans le serveur, sous forme de chaîne de caractères.***
- ***http://localhost:8080/Archivage/nom-photo : GET : renvoie l'image sous forme de flux d'octets, avec le type MIME « image/jpeg » depuis le serveur, correspondant au nom proposée par l'URL.***
- ***http://localhost:8080/Archivage/nom-photo : POST : envoie l'image depuis le poste local vers le serveur, le nom du fichier image correspond au nom proposée par l'URL. Le contenu à envoyer est l'ensemble des octets constituant la photo. Nous devons également préciser le type MIME, ici également « image/jpeg ».***
- ***http://localhost:8080/Archivage/change?ancien=ancien-nom&nouveau=nouveau-nom : PUT : permet de changer le nom du fichier image du serveur distant.***
- ***http://localhost:8080/Archivage/nom-photo : DELETE : supprime définitivement la photo stockée sur le serveur distant, dont le nom est proposée à la fin de l'URL.***

✗ MISE EN ŒUVRE DU SERVICE WEB REST D'ARCHIVAGE DE PHOTOS

Vous avez ci-dessous le code Java correspondant au service même d'archivage dans la technologie REST. Malgré le fait que nous manipulions des données relativement conséquentes, le code lui-même demeure relativement simple au vue des fonctionnalités proposées.

```
package service;

import java.io.*;
import java.util.*;
import javax.ws.rs.*;

@Path("/")
public class Archivage {
   private final String répertoire = "/home/manu/Applications/Archivage/";
```

```
    @GET
    @Produces("text/plain")
    public String listePhotos() {
        String[] liste = new File(répertoire).list();
        StringBuilder noms = new StringBuilder();
        for (String nom : liste) noms.append(nom.split(".jpg")[0]+'\n');
        return noms.toString();
    }

    @GET
    @Path("{nomFichier}")
    @Produces("image/jpeg")
    public InputStream restituer(@PathParam("nomFichier") String nom) throws FileNotFoundException {
        return new FileInputStream(répertoire+nom+".jpg");
    }

    @POST
    @Path("{nomFichier}")
    @Consumes("image/jpeg")
    public void stocker(@PathParam("nomFichier") String nom, InputStream flux) throws IOException {
        byte[] octets = lireOctets(flux);
        FileOutputStream fichier = new FileOutputStream(répertoire+nom+".jpg");
        fichier.write(octets);
        fichier.close();
    }

    @PUT
    @Path("change")
    public void  changerNom(@QueryParam("ancien") String ancien, @QueryParam("nouveau") String nouveau) {
        new File(répertoire+ancien+".jpg").renameTo(new File(répertoire+nouveau+".jpg"));
    }

    @DELETE
    @Path("{nomFichier}")
    public void supprimer(@PathParam("nomFichier") String nom) {
        new File(répertoire+nom+".jpg").delete();
    }

    private byte[] lireOctets(InputStream stream) throws IOException {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024]; int octetsLus = 0;
        do {
            octetsLus = stream.read(buffer);
            if (octetsLus > 0) { baos.write(buffer, 0, octetsLus); }
        }
        while (octetsLus > -1);
        return baos.toByteArray();
    }
}
```

MISE EN ŒUVRE DE LA PARTIE CLIENTE AU WEB SERVICE REST D'ARCHIVAGE

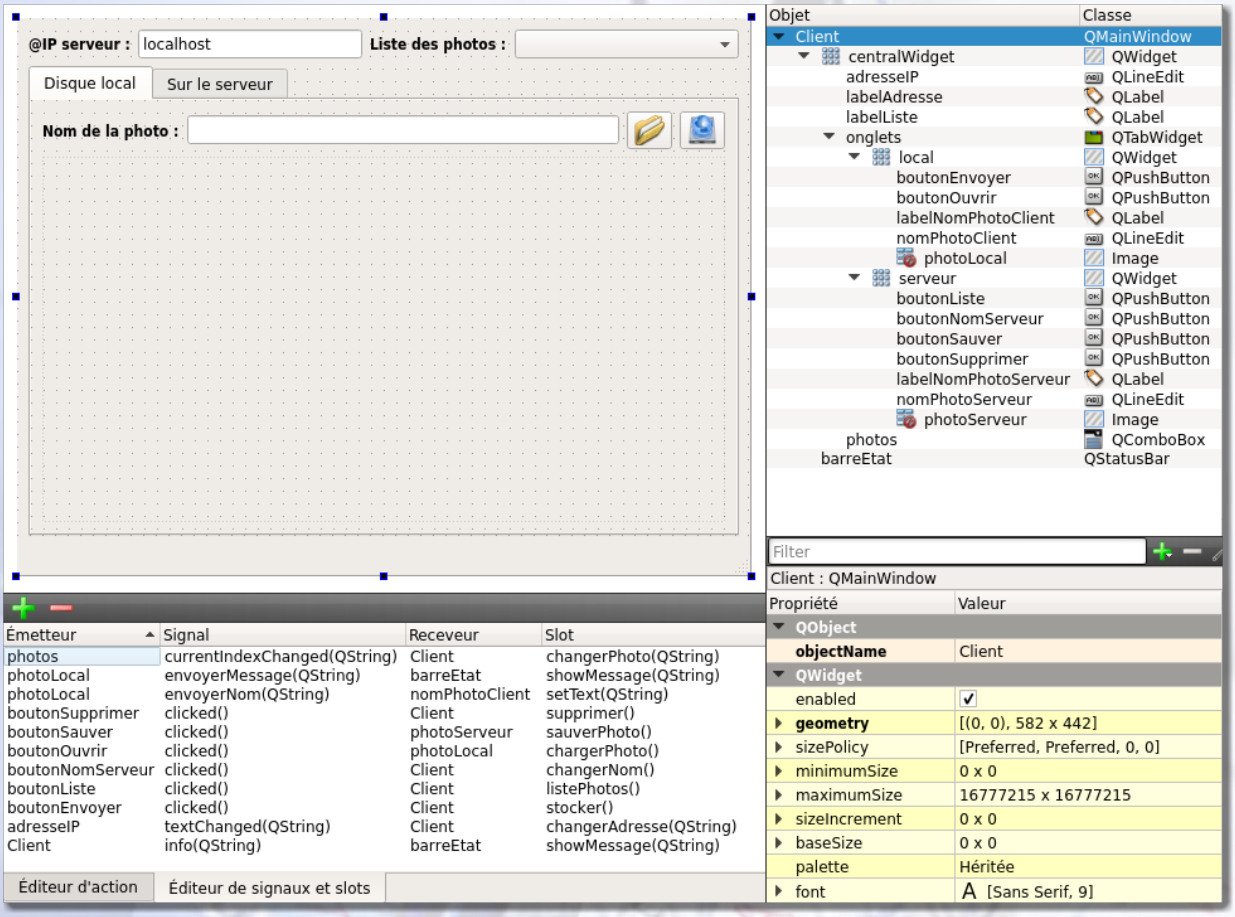
Par rapport au premier projet, ce fichier de configuration de projet est relativement similaire, si ce n'est les noms des fichiers sources et des fichiers en-têtes utilisés.

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-01-15T21:34:58
#
#-------------------------------------------------
QT       += core gui network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11
TARGET = ClientPhotoREST
TEMPLATE = app
SOURCES += main.cpp\
           client.cpp \
           image.cpp
HEADERS  += client.h \
            image.h
FORMS    += client.ui
```

x CONSTITUTION DE L'IHM ET DE LA GESTION ÉVÉNEMENTIELLE



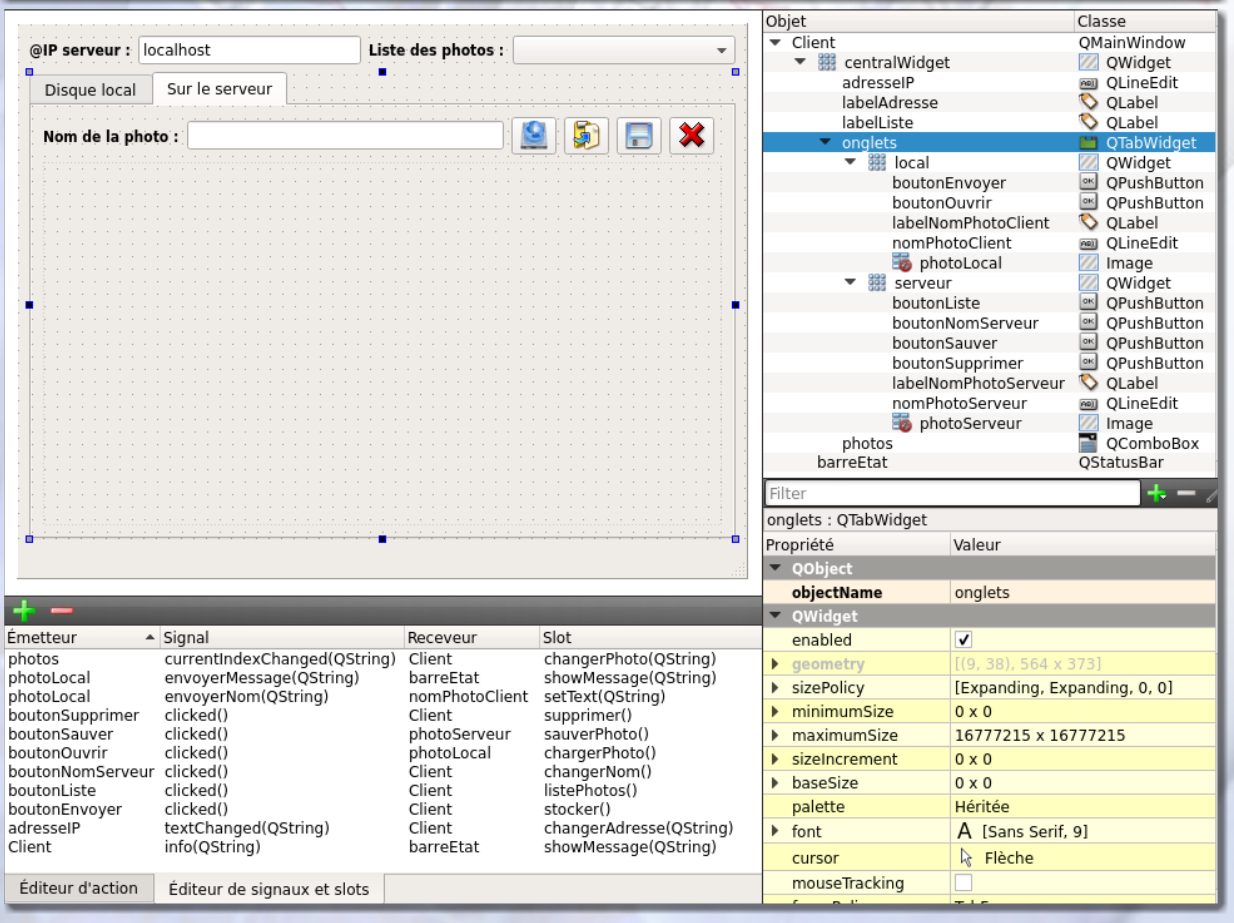


IMAGE.H

```
#ifndef IMAGE_H
#define IMAGE_H

#include <QWidget>

class Image : public QWidget
{
    Q_OBJECT
public:
    explicit Image(QWidget *parent = 0) : QWidget(parent) {}
    void chargerPhoto(const QByteArray &octets);
    QByteArray getOctets() { return octets; }
signals:
    void envoyerMessage(const QString &message);
    void envoyerNom(const QString &nom);
public slots:
    void chargerPhoto();
    void sauverPhoto();
protected:
    void paintEvent(QPaintEvent *) override;
private:
    QImage photo;
    QByteArray octets;
};

#endif // IMAGE_H
```

IMAGE.CPP

```
#include "image.h"

#include <QFileDialog>
#include <QPainter>
#include <QRect>
#include <Qfile>

void Image::chargerPhoto()
{
    QString nom = QFileDialog::getOpenFileName(this, "Choisissez votre photo", "", "Images (*.jpeg *.jpg)");
    if (!nom.isEmpty())
    {
        envoyerMessage(QString("Local : %1").arg(nom));
        QFile fichier(nom);
        fichier.open(QIODevice::ReadOnly);
        octets = fichier.readAll();
        photo.loadFromData(octets);
        update();
        QFileInfo infoFichier(nom);
        envoyerNom(infoFichier.baseName());
    }
}

void Image::chargerPhoto(const QByteArray &octets)
{
    photo.loadFromData(this->octets = octets);
    update();
}

void Image::sauverPhoto()
{
    if (!photo.isNull())
    {
        QString nom = QFileDialog::getSaveFileName(this, "Sauvegardez la photo");
        if (!nom.isEmpty())
        {
            QFile fichier(nom);
            fichier.open(QIODevice::WriteOnly);
            fichier.write(octets);
            envoyerMessage("La photo est sauvegardée sur le disque local");
        }
    }
}
```

```
void Image::paintEvent(QPaintEvent *)
{
    if (!photo.isNull())
    {
        QPainter dessin(this);
        double ratio = (double) photo.width() / photo.height();
        int largeur = width();
        int hauteur = width() / ratio;
        QRect cadrage(0, 0, largeur, hauteur);
//        QImage image = photo.scaledToWidth(width());
        dessin.drawImage(cadrage, photo, photo.rect());
    }
    else envoyerMessage("Choisissez votre photo");
}
```

CLIENT.H

```
#ifndef CLIENT_H
#define CLIENT_H

#include <QMainWindow>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QUrlQuery>
#include "ui_client.h"

class Client : public QMainWindow, public Ui::Client
{
    Q_OBJECT
public:
    explicit Client(QWidget *parent = 0);
private:
    QNetworkAccessManager reseau;
    QString adresse;
    enum {Aucune, ListePhotos, Restituer, Stocker, ChangerNom, Supprimer} requete = Aucune;
private slots:
    void reponse(QNetworkReply *resultat);
    void changerAdresse(const QString &adresse);
    void listePhotos();
    void changerPhoto(const QString &nom);
    void stocker();
    void changerNom();
    void supprimer();
signals:
    void info(const QString &info);
private:
    void restituer();
};

#endif // CLIENT_H
```

CLIENT.CPP

```
#include "client.h"

Client::Client(QWidget *parent) : QMainWindow(parent)
{
    setupUi(this);
    connect(&reseau, SIGNAL(finished(QNetworkReply*)), this, SLOT(reponse(QNetworkReply*)));
    adresse = "http://";
    adresse += adresseIP->text();
    adresse += ":8080/Archivage/";
}

void Client::reponse(QNetworkReply *resultat)
{
    if (resultat->error() == QNetworkReply::NoError)
    {
        switch (requete) {
        case ListePhotos :
            photos->clear();
            while(resultat->canReadLine()) {
                QString photo = resultat->readLine();
                photo.remove("\n");
                photos->addItem(photo);
            }
            nomPhotoServeur->setText(photos->currentText());
            break;
```

```
        case Restituer :
            photoServeur->chargerPhoto(resultat->readAll());
            break;
        case Stocker :
            info("Photo envoyée sur le serveur");
            listePhotos();
            break;
        case ChangerNom :
            info("Le nom de la photo est changée sur le serveur");
            listePhotos();
            break;
        case Supprimer :
            info("La photo a été supprimée sur le serveur");
            listePhotos();
            break;
        case Aucune :
            info("Aucune requête demandée");
            break;
        }
    }
    else info("Problème avec le service d'archivage !");
    delete resultat;
}


void Client::listePhotos()
{
    requete = ListePhotos;
    reseau.get(QNetworkRequest(QUrl(adresse)));
}


void Client::changerPhoto(const QString &nom)
{
    nomPhotoServeur->setText(nom);
    onglets->setCurrentIndex(1);
    restituer();
}


void Client::restituer()
{
    requete = Restituer;
    QString nomImage = QString("%1%2").arg(adresse).arg(nomPhotoServeur->text());
    info(nomImage);
    reseau.get(QNetworkRequest(QUrl(nomImage)));
}


void Client::stocker()
{
    QString nom = nomPhotoClient->text();
    if (!nom.isEmpty())
    {
        requete = Stocker;
        QString url = adresse;
        url+=nom;
        QNetworkRequest envoi;
        envoi.setUrl(QUrl(url));
        envoi.setRawHeader("Content-Type", "image/jpeg");
        reseau.post(envoi, photoLocal->getOctets());
    }
}


void Client::changerNom()
{
    QString nom = nomPhotoServeur->text();
    if (!nom.isEmpty())
    {
        requete = ChangerNom;
        QString url = QString("%1change?ancien=%2&nouveau=%3").arg(adresse).arg(photos->currentText()).arg(nom);
        QByteArray vide;
        reseau.put(QNetworkRequest(QUrl(url)), vide);
    }
}
```

```
void Client::supprimer()
{
    QString nom = nomPhotoServeur->text();
    if (!nom.isEmpty())
    {
        requete = Supprimer;
        QString url = adresse;
        url+=nom;
        reseau.deleteResource(QNetworkRequest(QUrl(url)));
    }
}

void Client::changerAdresse(const QString &adresse)
{
    this->adresse = "http://";
    this->adresse += adresse;
    this->adresse += ":8080/Archivage/";
    info(this->adresse);
}
```

- Vous remarquez que cette fois-ci nous avons proposé l'ensemble des requête HTTP en utilisant les méthodes **get()**, **post()**, **put()** et **deleteResource()** de la classe **QNetworkAccessManager**.
- Par ailleurs, lorsque nous devons spécifier le type mime lors de l'envoi d'un contenu supplémentaire, comme c'est le cas avec les méthodes **post()** et **put()**, nous devons donc au préalable utiliser la méthode **setRawHeader()** de la classe **QNetworkRequest**.