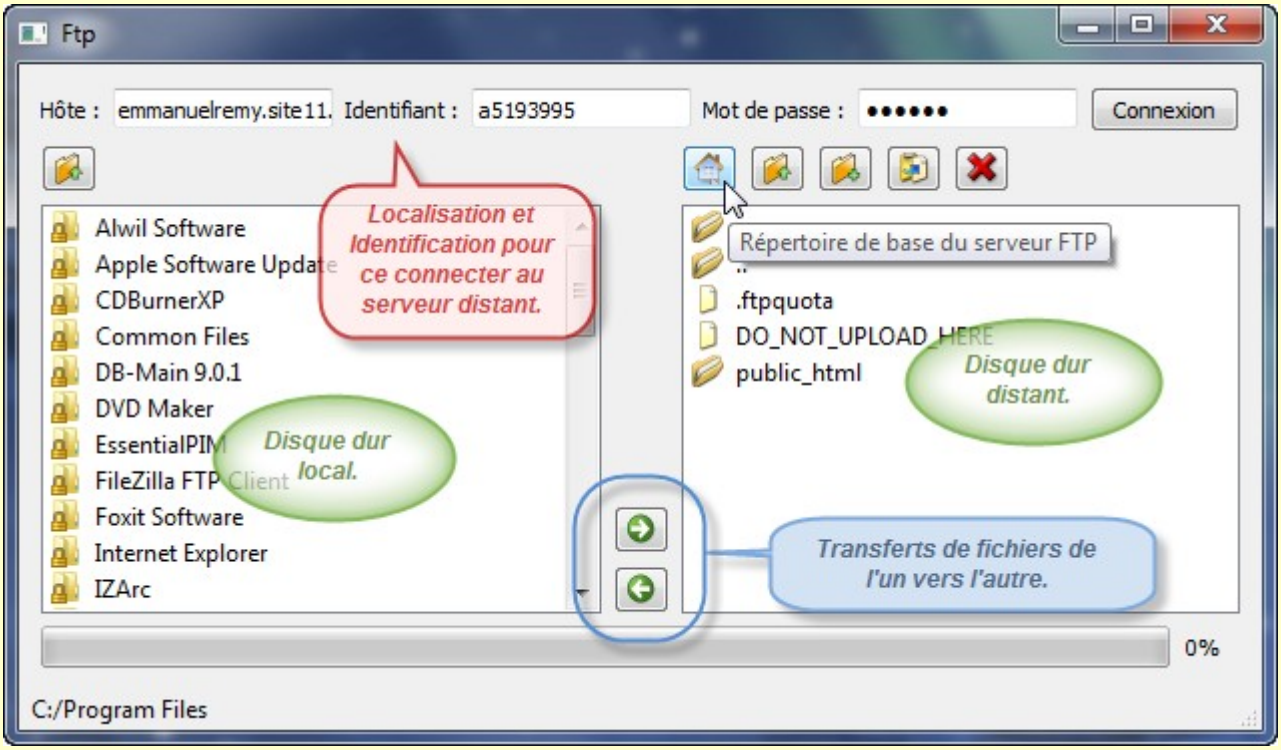
Lors de cette étude, nous allons nous intéresser plus particulièrement à la communication en réseau local. Nous pourront utiliser les notions de flux et de fichiers que nous avons abordés lors de l'étude précédente. Nous allons évaluer toutes ces techniques au travers de trois projets.

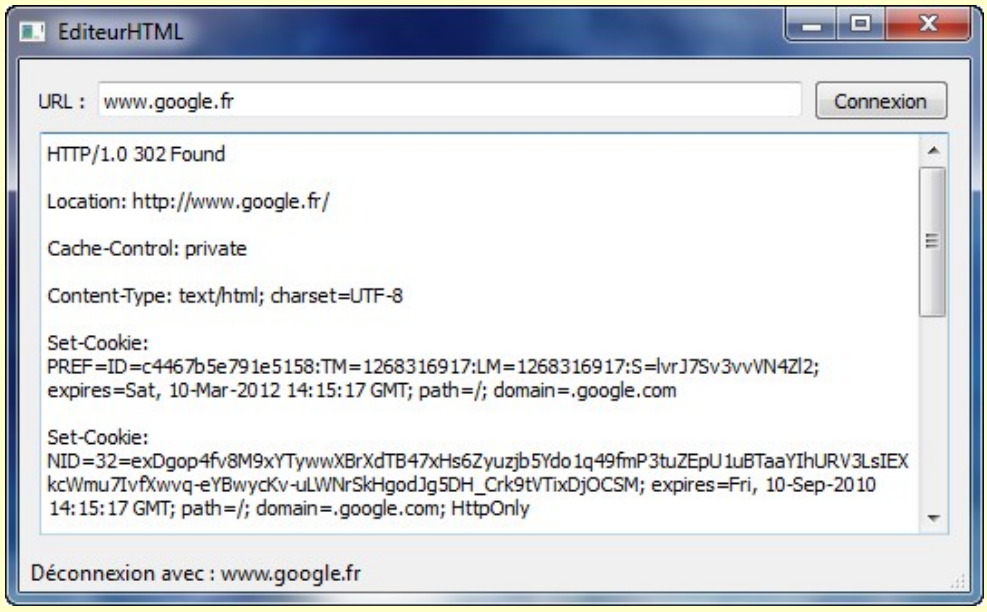
> *Pour mettre en œuvre ces différents applications, vous aurez besoin d'intégrer la bibliothèque **QtNetWork** dans chacun des projets respectifs. Cette étude nous permettra de connaître comment établir des connexions entre objets **par programme** au travers **des signaux et des slots.***

PROJETS

- ***Le premier projet consiste à réaliser un client FTP eu utilisant une classe prévue à cette effet : QFtp. L'objectif, comme son nom l'indique sera de pouvoir transférer, ou de récupérer, des fichiers sur un site distant, par exemple pour déployer notre site Web. Ce client est composé de deux parties. La première localise l'arborescence du disque dur local. La deuxième est une image de l'arborescence du disque dur distant.***



- ***Le deuxième projet nous permet de travailler avec la technique des sockets en ce connectant sur des serveurs Web et en demandant de récupérer le document HTML correspondant de la page d'accueil du site.***



GESTION DU RÉSEAU

Qt fournit les classes **QFtp** et **QHttp** pour la programmation de **FTP** et **HTTP**. Ces protocoles sont faciles à utiliser pour télécharger des fichiers et, dans le cas de HTTP, pour envoyer des requêtes aux serveurs Web et récupérer les résultats.

Qt fournit également les classes de bas niveau **QTcpSocket** et **QUdpSocket**, qui implémentent les protocoles de transport **TCP** et **UDP**.

- ***TCP est un protocole orienté connexion fiable qui agit en terme de flux de données transmis entre les nœuds du réseau.***
- ***UDP est un protocole fonctionnant en mode non connecté non fiable qui permet d'envoyer des paquets discrets entre les nœuds du réseau.***

*Tous deux peuvent être utilisés pour créer des applications réseau clientes et serveur. En ce qui concerne les serveurs, nous pouvons aussi avoir besoin de la classe **QTcpServer** pour gérer les connexions **TCP** entrantes.*

*Pour nos deux projets, nous utiliserons respectivement, la classe **QFtp** puisqu'elle est parfaitement adaptée à la mise en œuvre d'un client **FTP**, ainsi que la classe **QTcpSocket** qui nous permettra de comprendre comment communiquer avec un service distant quelle que soit le type de service demandé, Web ou autre.*

PROGRAMMER LES CLIENTS FTP

La classe **QFtp** implémente le côté client du protocole **FTP** dans Qt. Elle offre diverses méthodes destinées à réaliser les opérations **FTP** les plus courantes et nous permet ainsi d'exécuter des commandes **FTP** arbitraires.

Pour réaliser une communication entre poste client et un serveur **FTP**, vous devez suivre une certaine séquence avant que la communication soit opérationnelle :

- ***L'établissement d'une communication sur un serveur distant FTP se réalise au travers de la méthode connectToHost() d'un objet de type QFtp.***
- ***Une fois que la communication est établie, vous devez vous loguer à l'aide de la méthode login() en spécifiant éventuellement votre nom d'utilisateur suivi du mot de passe. Sinon, vous vous loguer en mode anonyme.***
- ***Dès lors il est possible de réaliser toutes les opérations utiles et nécessaires au commandes classiques du protocole FTP, comme : get(), put(), cd(), list(), mkdir(), etc.***
- ***Dès que vous avez terminé tout ce que vous deviez faire avec votre serveur FTP, vous pouvez clôturer la session au travers de la méthode close(). Ceci dit, cette méthode est automatiquement appelée dès que l'objet représente la classe QFtp est détruit.***

La classe QFtp fonctionne de façon asynchrone**. Lorsque nous appelons les méthodes telles que **get()** ou **put()**, qui permettent respectivement de récupérer ou d'envoyer des fichiers au travers du réseau, elles se terminent immédiatement et le transfert de données se produit quand le contrôle revient à la boucle d'événement de **Qt**. **Ainsi, l'interface utilisateur reste réactive pendant l'exécution des commandes FTP.

- ***QFtp(parent) : construit un objet représentant le protocole FTP.***
- ***cd(répertoire par défaut) : permet de changer le répertoire courant dans le serveur FTP.***
- ***close() : clôture la connexion FTP.***
- ***connectToHost(localisation, port=21) : Établissement de la connexion avec le serveur FTP distant sur le port 21 par défaut et au moyen de l'adresse IP ou de son nom DNS.***
- ***get(nom du fichier distant, fichier en local) : Permet de récupérer le fichier désigné stocké dans le serveur distant sur le poste local en précisant en argument, à la fois le nom du fichier distant et en donnant également le QIODevice correspondant au fichier local.***
- ***list() : Demande de récupération de la liste des fichiers et des répertoires du répertoire courant dans le serveur distant. A l'issue de cette demande, le système contrôle effectivement la présence de fichiers (ou répertoires) dans ce répertoire. A chacun de ces fichiers (ou répertoires), un signal listInfo() est envoyé, qu'il suffit alors de capter pour connaître la nature de l'élément en question.***
- ***login(utilisateur, mot de passe) : Après l'établissement de la connexion, vous devez appeler cette méthode pour pouvoir utiliser ce serveur FTP distant. Cela permet de savoir si vous êtes un utilisateur anonyme (aucun argument est nécessaire dans ce cas là) ou si vous êtes un utilisateur autorisé (vous devez alors préciser le nom de l'utilisateur avec son mot de passe).***
- ***mkdir(nouveau répertoire) : Création d'un sous-répertoire dans le répertoire courant.***
- ***put(fichier en local, nom du fichier distant) : Permet d'envoyer le fichier local dans le répertoire courant du serveur distant avec, pour nom du fichier, celui proposé en argument.***
- ***remove(nom du fichier distant) : supprime définitivement le fichier distant dont le nom est proposé en argument.***

- **rename(ancien nom, nouveau nom) : permet de changer le nom du fichier sur le serveur distant.**
- **rmdir(nom du répertoire distant) : supprime le répertoire distant, si ce dernier est vide bien entendu.**

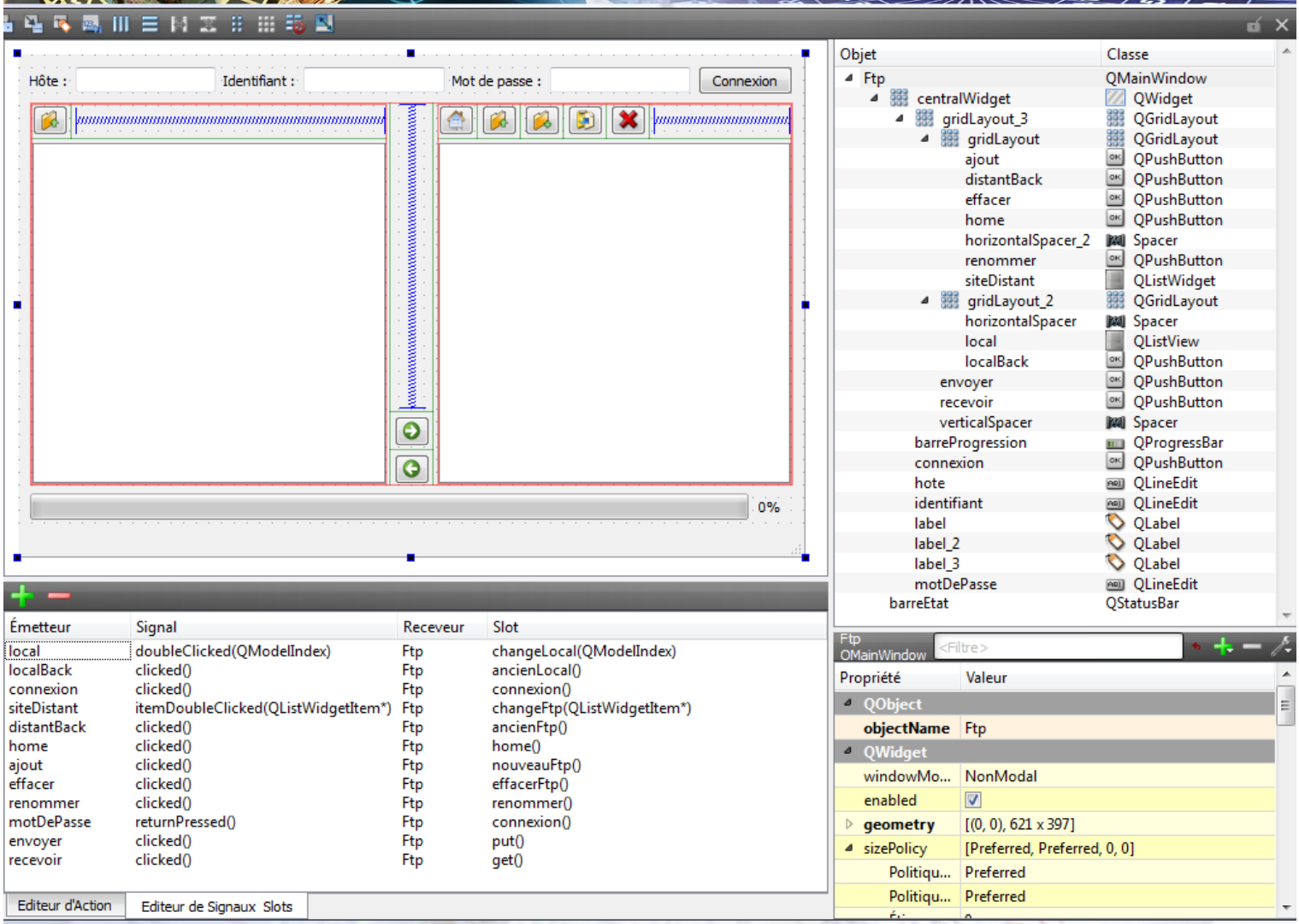
*Comme nous venons de l'évoquer, la classe **QFtp** fonctionne de façon asynchrone. Ainsi, lorsque nous proposons une commande particulière au travers de la méthode souhaitée, le système rend tout de suite la main au programme principal. Pour savoir si votre commande a abouti ou pour récupérer une valeur, vous devez alors repérer le bon signal qui vous alerte de la situation en cours. Voici ci-dessous l'ensemble des signaux que la classe Qftp est susceptible d'envoyer :*

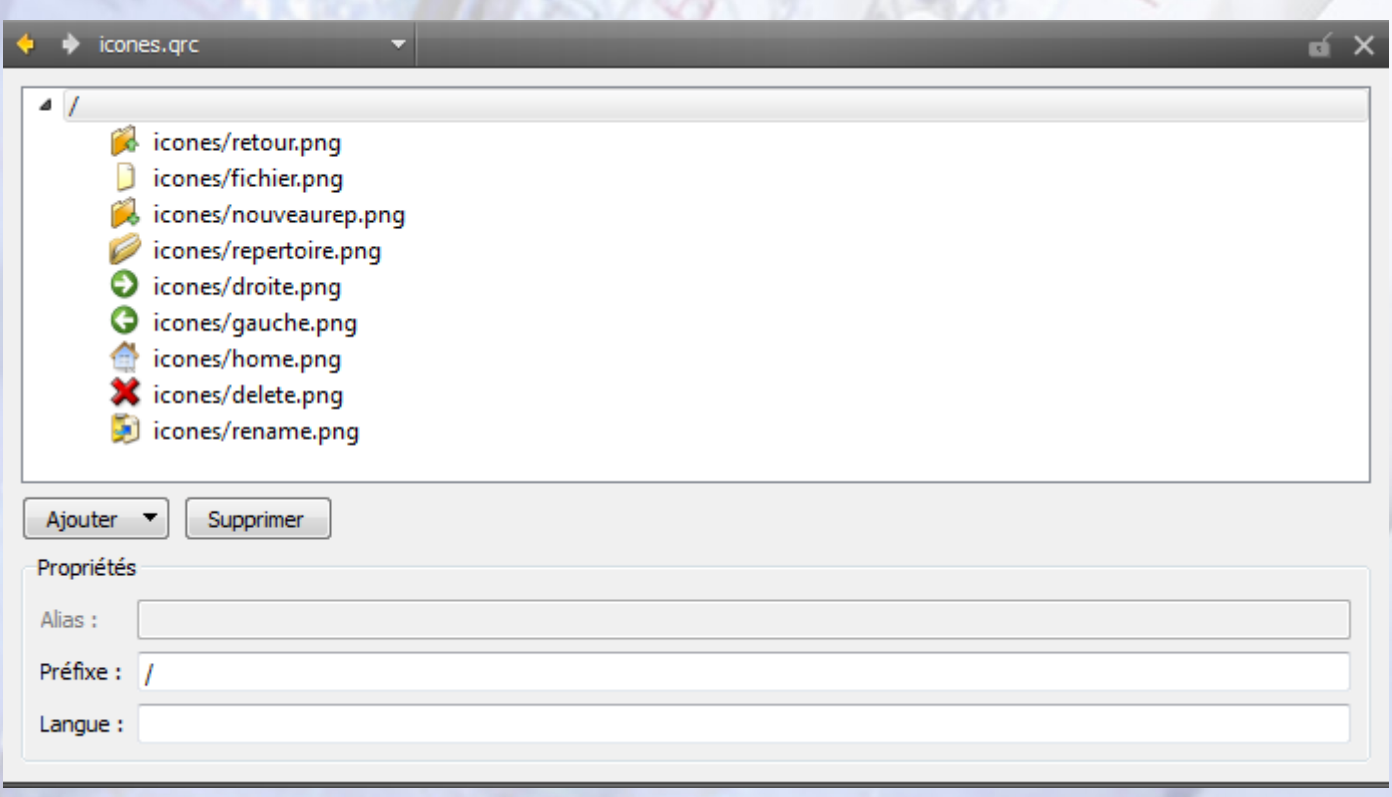
- **commandFinish(id, erreur) : Ce signal est émis une fois que la commande est terminée. La valeur de id correspond à l'identifiant de la méthode utilisée. En effet, toutes les méthodes comme connectToHost(), login(), close(), list(), cd(), get(), put(), remove(), mkdir(), rmdir() et rename() retournent un identifiant de type entier au moment de l'appel.**
- **commandStarted(id) : Ce signal est émis quand il commence à exécuter une commande.**
- **dataTransfertProgress(déjà transféré, total) : Signal très intéressant qui nous indique la progression du transfert des octets lors de l'appel des méthodes get() et put(). Les paramètres sont de type qint64 qui pourront devenir des int.**
- **done(erreur) : L'achèvement de toutes les commandes est indiqué par ce signal. Par exemple, il peut nous permettre de savoir si le téléchargement est terminé et que le fichier peut être fermé en toute sécurité.**
- **listInfo(info) : Ce signal est sollicité lorsque nous faisons appel à la commande list(). Ce signal est émis pour chaque fichier et répertoire contenu dans le répertoire à consulter.**
- **readyRead() : Ce signal est émis lorsque le système est près à faire une lecture.**

CLIENT FTP

ftp.h | <Selectionner un symbole> | Ligne : 1, Col : 1

```
1   #ifndef FTP_H
2   #define FTP_H
3
4   #include <QMainWindow>
5   #include <QFileSystemModel>
6   #include <QFtp>
7   #include "ui_ftp.h"
8
9   class Ftp : public QMainWindow, Ui::Ftp {
10    Q_OBJECT
11  public:
12    Ftp(QWidget *parent = 0);
13    ~Ftp();
14  private slots:
15    void changeLocal(QModelIndex nouveau);
16    void changeFtp(QListWidgetItem *nouveau);
17    void ancienLocal();
18    void ancienFtp();
19    void home();
20    void nouveauFtp();
21    void renommer();
22    void effacerFtp();
23    void connexion();
24    void listeDistante(QUrlInfo);
25    void message(bool);
26    void effacer();
27    void get();
28    void put();
29    void progression(qint64 nombre, qint64 total);
30  private:
31    QFileSystemModel modele;
32    QModelIndex index;
33    QFtp distant;
34    QIcon iconeRep, iconeFichier;
35    QList<QUrlInfo> liste;
36    QFile fichier;
37  };
38
39  #endif // FTP_H
40
```


Hôte :
Identifiant :
Mot de passe :
Connexion
0%
Objet
Classe
Ftp
QMainWindow
centralWidget
QWidget
gridLayout_3
QGridLayout
gridLayout
QGridLayout
ajout
QPushButton
distantBack
QPushButton
effacer
QPushButton
home
QPushButton
horizontalSpacer_2
Spacer
renommer
QPushButton
siteDistant
QListWidget
gridLayout_2
QGridLayout
horizontalSpacer
Spacer
local
QListView
localBack
QPushButton
envoyer
QPushButton
recevoir
QPushButton
verticalSpacer
Spacer
barreProgression
QProgressBar
connexion
QPushButton
hote
QLineEdit
identifiant
QLineEdit
label
QLabel
label_2
QLabel
label_3
QLabel
motDePasse
QLineEdit
barreEtat
QStatusBar
Émetteur
Signal
Receveur
Slot
local
doubleClicked(QModelIndex)
Ftp
changeLocal(QModelIndex)
localBack
clicked()
Ftp
ancienLocal()
connexion
clicked()
Ftp
connexion()
siteDistant
itemDoubleClicked(QListWidgetItem*)
Ftp
changeFtp(QListWidgetItem*)
distantBack
clicked()
Ftp
ancienFtp()
home
clicked()
Ftp
home()
ajout
clicked()
Ftp
nouveauFtp()
effacer
clicked()
Ftp
effacerFtp()
renommer
clicked()
Ftp
renommer()
motDePasse
returnPressed()
Ftp
connexion()
envoyer
clicked()
Ftp
put()
recevoir
clicked()
Ftp
get()
Editeur d'Action
Editeur de Signaux Slots
Ftp
QMainWindow
<Filtre>
Propriété
Valeur
QObject
objectName
Ftp
QWidget
windowMo...
NonModal
enabled
geometry
[(0, 0), 621 x 397]
sizePolicy
[Preferred, Preferred, 0, 0]
Politiqu...
Preferred
Politiqu...
Preferred


icones.qrc
/
icones/retour.png
icones/fichier.png
icones/nouveaurep.png
icones/repertoire.png
icones/droite.png
icones/gauche.png
icones/home.png
icones/delete.png
icones/rename.png
Ajouter
Supprimer
Propriétés
Alias :
Préfixe :
/
Langue :

ftp.cpp | <Selectionner un symbole> | Ligne : 1, Col : 1

```
1   #include "ftp.h"
2   #include <QInputDialog>
3
4   Ftp::Ftp(QWidget *parent) : QMainWindow(parent), iconeRep(":/icones/repertoire.png"), iconeFichier(":/icones/fichier.png")
5   {
6       setupUi(this);
7       index = modele.setRootPath("C:/");
8       local->setModel(&modele);
9       connect(&distant, SIGNAL(listInfo(QUrlInfo)), this, SLOT(listeDistante(QUrlInfo)));
10      connect(&distant, SIGNAL(done(bool)), this, SLOT(message(bool)));
11      connect(&distant, SIGNAL(dataTransferProgress(qint64,qint64)), this, SLOT(progression(qint64,qint64)));
12  }
13
14  Ftp::~Ftp() { distant.close(); }
15
16  void Ftp::changeLocal(QModelIndex nouveau)
17  {
18    local->setRootIndex(index = nouveau);
19    barreEtat->showMessage(modele.filePath(index));
20  }
21
22  void Ftp::changeFtp(QListWidgetItem *nouveau)
23  {
24    distant.cd(nouveau->text());
25    effacer();
26  }
27
28  void Ftp::ancienLocal()
29  {
30    local->setRootIndex(index = index.parent());
31    barreEtat->showMessage(modele.filePath(index));
32  }
33
34  void Ftp::ancienFtp()
35  {
36    distant.cd("..");
37    effacer();
38  }
39
40  void Ftp::nouveauFtp()
41  {
42    QString dir = QInputDialog::getText(this, "Création", "Nouveau répertoire :");
43    distant.mkdir(dir);
44    effacer();
45  }
46
47  void Ftp::renommer()
48  {
49    QString nom = QInputDialog::getText(this, "Renommer", "Nouveau nom :");
50    int index = siteDistant->currentIndex().row();
51    distant.rename(liste[index].name(), nom);
52    effacer();
53  }
54
55  void Ftp::effacerFtp()
56  {
57    int index = siteDistant->currentIndex().row();
58    if (liste[index].isDir()) distant.rmdir(liste[index].name());
59    else distant.remove(liste[index].name());
60    effacer();
61  }
```

QFtp fonctionne de façon asynchrone. Il est donc nécessaire de travailler avec les événements et ainsi de lancer les méthodes adéquates suivant les signaux proposés par QFtp. De plus, comme QFtp n'est pas un composant graphique, vous devez établir vos connexions événementielles par programme.

Tous les composants possède la méthode connect() qui permet de relier un signal à un slot. Nous devons préciser quatre arguments, les mêmes qu'en mode « design » : le premier est le pointeur de l'objet qui propose le signal, le deuxième est le signal lui-même, le troisième, le pointeur de l'objet qui doit recevoir le signal et le quatrième le slot à exécuter.

ftp.cpp | <Selectionner un symbole> | Ligne : 1, Col : 1

```
63  void Ftp::home()
64  {
65    distant.close();
66    connexion();
67  }
68
69  void Ftp::connexion()
70  {
71    distant.connectToHost(hote->text());
72    distant.login(identifiant->text(), motDePasse->text());
73    effacer();
74  }
75
76  void Ftp::listeDistante(QUrlInfo info)
77  {
78    liste.append(info);
79    if (info.isDir()) new QListWidgetItem(iconeRep, info.name(), siteDistant);
80    else new QListWidgetItem(iconeFichier, info.name(), siteDistant);
81  }
82
83  void Ftp::message(bool erreur)
84  {
85    if (erreur) barreEtat->showMessage("ATTENTION : l'opération n'a pas eu être exécutée correctement");
86    else barreEtat->showMessage("Opération exécutée avec succès");
87    fichier.close();
88  }
89
90  void Ftp::effacer()
91  {
92    siteDistant->clear();
93    liste.clear();
94    distant.list();
95  }
96
97  void Ftp::get()
98  {
99    barreEtat->showMessage("Récupération...");
100   int indexDistant = siteDistant->currentIndex().row();
101   if (liste[indexDistant].isFile()) {
102     QString nomFichier = modele.filePath(index) + "/" + liste[indexDistant].name();
103     fichier.setFileName(nomFichier);
104     if (fichier.open(QIODevice::WriteOnly)) distant.get(liste[indexDistant].name(), &fichier);
105   }
106 }
107
108 void Ftp::put()
109 {
110   barreEtat->showMessage("Envoi...");
111   QModelIndex index = local->currentIndex();
112   if (modele.fileInfo(index).isFile()) {
113     fichier.setFileName(modele.filePath(index));
114     if (fichier.open(QIODevice::ReadOnly)) {
115       distant.put(&fichier, modele.fileName(index));
116       effacer();
117     }
118   }
119 }
120
121 void Ftp::progression(qint64 nombre, qint64 total)
122 {
123   barreProgression->setMaximum(total);
124   barreProgression->setValue(nombre);
125 }
```

PROGRAMMER DES APPLICATIONS CLIENTES TCP

Les classes **QTcpServer** et **QTcpSocket** peuvent être utilisées pour implémenter des serveurs et des clients **TCP**. **TCP** est un protocole de transport sur lequel sont basés la plupart des protocoles Internet de niveau application, y compris **FTP** et **HTTP**. En outre, il est susceptible d'être utilisé pour des protocoles personnalisés.

TCP est un protocole orienté flux. Pour les applications, les données apparaissent sous la forme d'un long flux, de la même manière que la lecture ou l'écriture dans un fichier, sujet que nous avons abordé dans l'étude précédente. Par ailleurs, les protocoles de haut niveau basés sur **TCP** sont généralement orientés ligne ou bloc.

- ***Les protocoles orientés ligne transfèrent les données sous la forme de lignes de texte, chacune étant terminée par un retour à la ligne. Les flux sous forme de lignes de texte peuvent être maîtrisés au moyen de la classe QTextStream que nous connaissons bien.***
- ***Les protocoles orientés bloc transfèrent les données sous la forme de blocs de données binaires. Chaque bloc comprend un champ de taille suivi de la quantité de données spécifiées. Les flux binaires peuvent être envoyés ou reçus au moyen de la classe QDataStream.***

***QTcpSocket** hérite de **QIODevice** par le biais de la classe abstraite **QAbstractSocket**. Il peut donc être lu et écrit au moyen d'un **QDataStream** ou d'un **QTextStream**. Ainsi, envoyer ou recevoir des informations sur le réseau, procède de la même façon que pour enregistrer ou lire des informations dans un fichier.*

*Toutefois, la différence notable entre la lecture de données à partir d'un réseau et celle effectuée depuis un fichier est que nous devons veiller à avoir reçu suffisamment de données avant d'utiliser l'opérateur **>>**.*

***ATTENTION**, cette remarque est très importante. Si vous désirez transférer les données sous forme de bloc, vous ne pouvez pas écrire les données directement dans le **QTcpSocket** car nous ne connaissons pas à priori la taille du bloc avant d'y avoir placé toutes les données. Il est donc judicieux de passer par un **QByteArray** intermédiaire.*

LA CLASSE QTCPSOCKET QUI HÉRITE DE QABSTRACTSOCKET

- ***QTcpSocket(parent) : construit un objet représentant la communication réseau à un serveur distant.***
- ***bytesAvailable() : nombre d'octets disponibles sur le réseau pour être récupérés et donc pour la lecture.***
- ***bytesToWrite() : nombre d'octets en attente pour être envoyéer.***
- ***canReadLine() : renvoie true si une ligne complète est en attente dans le buffer de réception.***
- ***connectToHost(localisation, port) : Établissement de la connexion avec le serveur distant sur le port correspondant au service demandé en localisant le serveur au moyen de l'adresse IP ou de son nom DNS.***
- ***close() : clôture la connexion réseau.***
- ***disconnectFromHost() : clôture la connexion réseau en attendant toutefois que les données aient bien été transférées.***
- ***flush() : vide le tampon et envoi les données stockées dans ce tampon.***
- ***localAdress() : donne l'adresse IP du poste local.***
- ***localPort() : donne le numéro de service de la socket.***
- ***peerAdress() : donne l'adresse IP du poste distant.***
- ***peerName() : donne le nom DNS du site distant.***
- ***readLine() : lit la ligne complète de la chaîne de caractères.***

*Comme pour la classe **QFtp**, la classe **QTcpSocket** dispose d'un certain nombre de signaux à prendre en compte pour savoir quand récupérer les informations envoyées par le serveur. A ce sujet **QTcpSocket** fonctionne également en mode asynchrone. Voici justement ci-dessous la liste des signaux utiles :*

- ***connected() : ce signal est émis à l'issue de la demande de connexion établie par connectToHost(), si cette dernière s'est bien déroulée correctement.***
- ***deconnected() : signal émis pour la prise en compte de la déconnexion.***
- ***readyRead() : ce signal est émis dès que des informations sont récupérées et disponibles sur le réseau.***

ÉDITEUR HTML

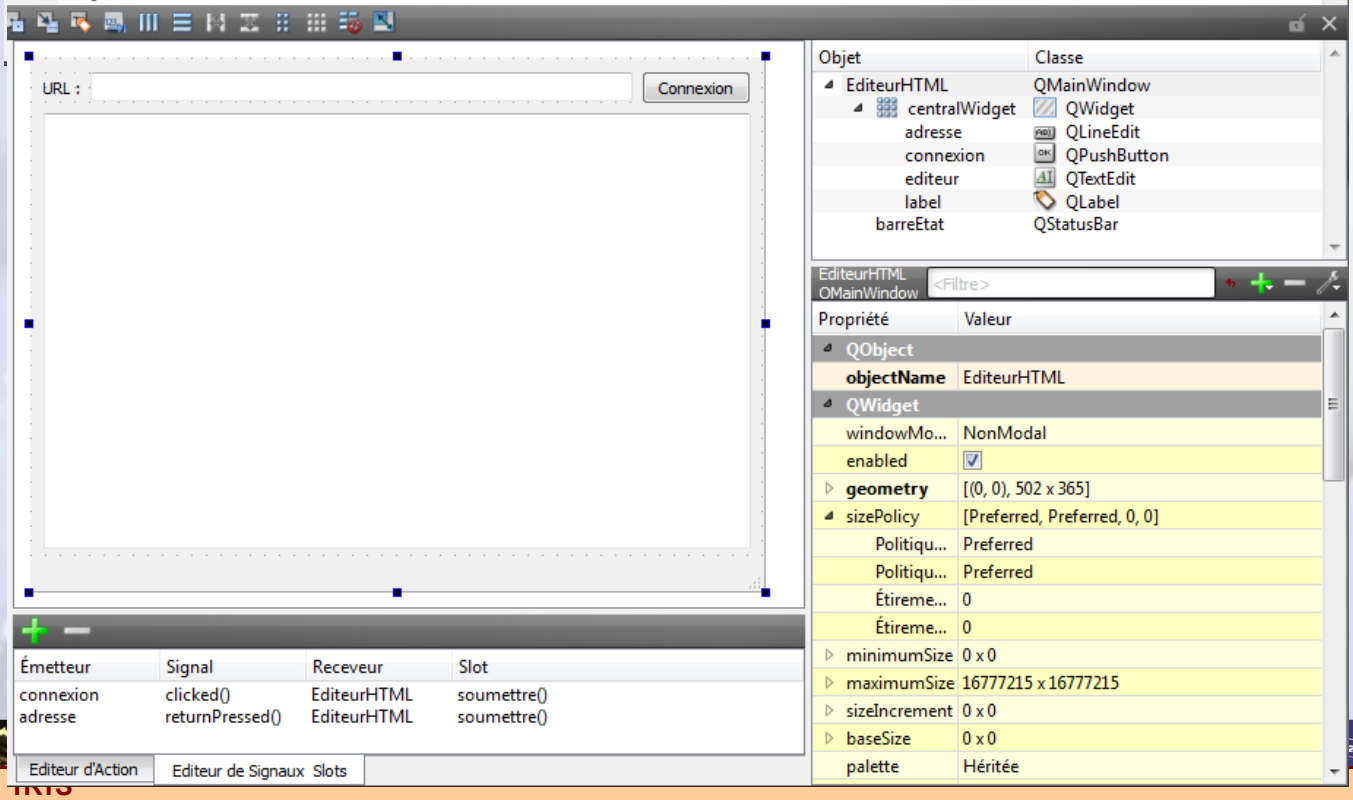
Nous disposons maintenant de suffisamment de compétence pour élaborer notre deuxième projet qui travaille avec la technique des sockets en ce connectant sur des serveurs Web et en demandant de récupérer le document HTML correspondant à la page d'accueil du site.

editeurhtml.h

```
#ifndef EDITEURHTML_H
#define EDITEURHTML_H

#include <QMainWindow>
#include <QtNetwork>
#include "ui_editeurhtml.h"

class EditeurHTML : public QMainWindow, public Ui::EditeurHTML {
    Q_OBJECT
public:
    EditeurHTML(QWidget *parent = 0);
private slots:
    void soumettre();
    void lecture();
    void deconnecter()  { barreEtat->showMessage("Déconnexion avec : "+ adresse->text()); }
private:
    QTcpSocket service;
};

#endif // EDITEURHTML_H
```

editeurhtml.cpp

```
#include "editeurhtml.h"

EditeurHTML::EditeurHTML(QWidget *parent) : QMainWindow(parent)
{
    setupUi(this);
    connect(&service, SIGNAL(readyRead()), this, SLOT(lecture()));
    connect(&service, SIGNAL(disconnected()), this, SLOT(deconnecter()));
}

void EditeurHTML::soumettre()
{
   service.connectToHost(adresse->text(), 80);
   QTextStream requete(&service);
   requete << "GET / HTTP/1.0\n" << endl;
}

void EditeurHTML::lecture()
{
   editeur->clear();
   while (service.canReadLine()) editeur->append(service.readLine());
   service.close();
}
```



PROGRAMMER LES APPLICATIONS CLIENT/SERVEUR TCP

Il est tout-à-fait possible de créer notre propre service, par exemple un service qui calcule pour nous la conversion entre les €uros et les Francs. Je rappelle qu'un service est un simple programme qui fonctionne constamment et qui « écoute » si un éventuel client n'a pas besoin de ses compétences. Pour que le client accède à ce service particulier, il doit solliciter, comme nous l'avons vu précédemment, un numéro de port (de service) correspondant à la requête souhaitée.

*Lorsque nous montons un service, nous devons créer une classe qui hérite de la classe **QTcpServer**, spécialement prévue à cette effet. Ensuite, nous devons proposer l'écoute sur un port qui n'est pas utilisé par un autre service au moyen de la méthode **listen()**. Enfin, nous devons redéfinir la méthode **incomingConnection()** qui est appelée automatiquement dès qu'un client tente d'établir une connexion au port écouté par le serveur.*

LA CLASSE QTCPSERVER

- **QTcpServer(parent) : construit un objet représentant le service souhaité.**
- **close() : clôture définitivement le service. Plus d'écoute n'est alors prise en compte.**
- **errorString() : renvoie la dernière occurrence d'une erreur survenue, sous forme de chaîne de caractères.**
- **incomingConnection(descripteur de socket) : Cette méthode est automatiquement appelée par le serveur lorsque une nouvelle connexion est demandée par un client. Le paramètre de la méthode est une valeur entière correspondant au descripteur de socket qui va être utile au serveur puisque ce dernier possède en interne la liste de toutes les connexions actuelles. Vous vous servez ensuite de cette méthode pour créer une socket, comme nous l'avons vu précédemment, au moyen de la classe QTcpSocket qui va être le représentant du point de communication avec ce client qui propose la requête. Servez-vous de la méthode setSocketDescriptor() de la socket pour prendre en compte le descripteur donné en paramètre. A l'issu de cet appel, le signal newConnection() est alors émis.**
- **listen(adresses autorisées, port) : appelle le service à écouter toutes communications issues des différents clients sur le numéro de port spécifié. C'est à ce moment là que le service est réellement activé. Il est possible de filtrer et de n'autoriser que certains clients. Si vous proposer la constante QHostAdress::Any, alors toutes les machines du réseau peuvent accéder à ce service.La méthode renvoie true si le service peut se lancer (par exemple si le numéro de port est bien vacant).**
- **serverAdress() : donne l'adresse IP du serveur.**
- **serverError() : retourne le code d'erreur survenu.**
- **serverPort() : retourne le numéro de service utilisé par le serveur.**

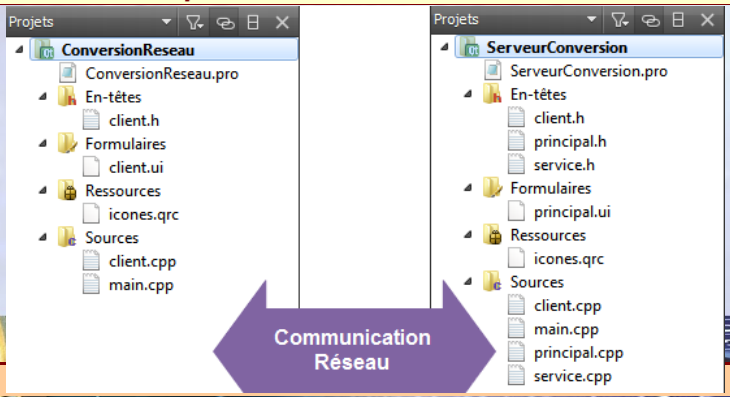
*Comme pour la classe **QFtp**, la classe **QTcpServer** dispose du signal ci-dessous à prendre en compte pour savoir quand un client tente de se connecter au serveur. A ce sujet **QTcpServer** fonctionne également en mode asynchrone.*

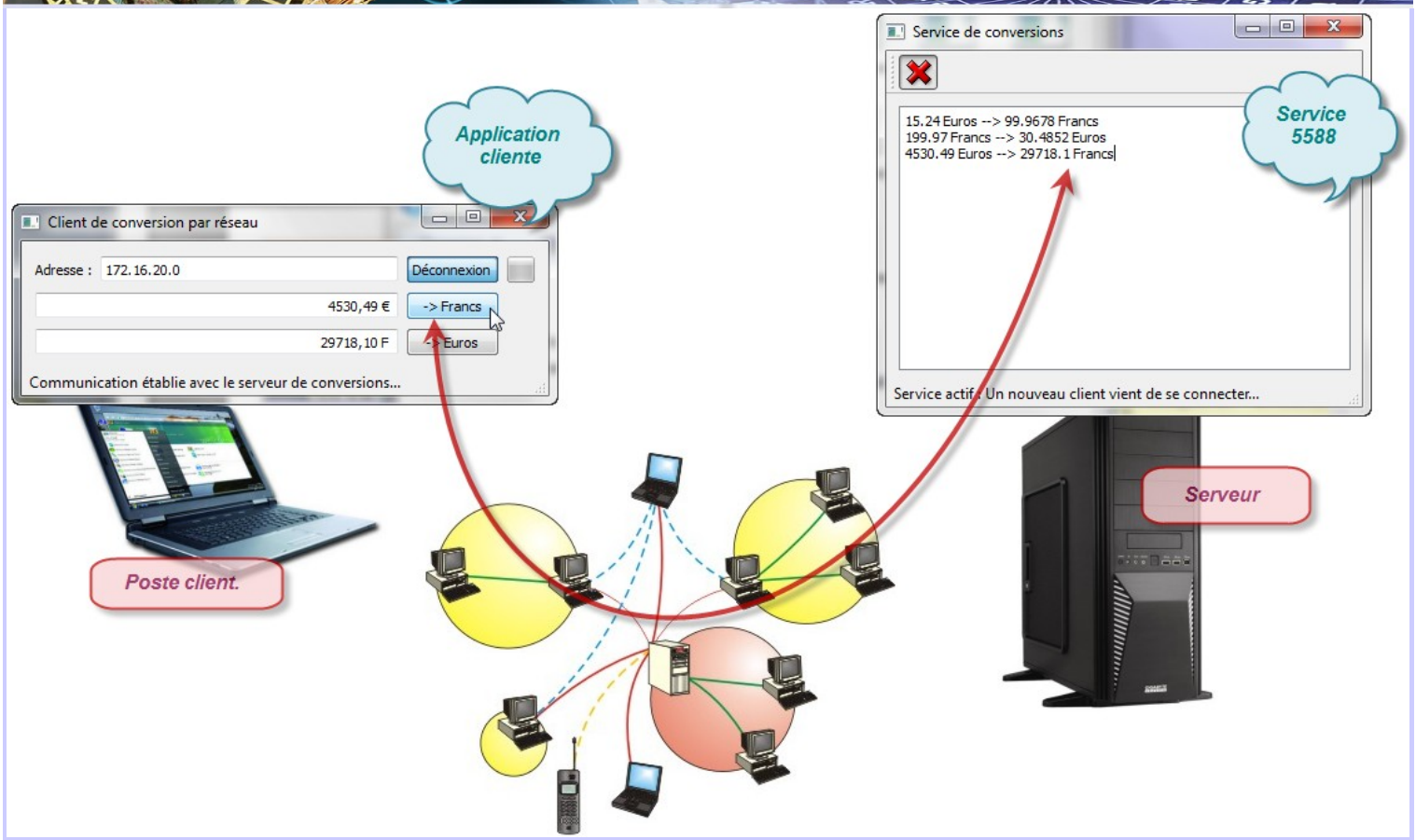
- **newConnection() : ce signal est émis à chaque fois qu'un client tente de se connecter à ce numéro de service spécifique.**

CLIENT / SERVEUR CONVERSIONS

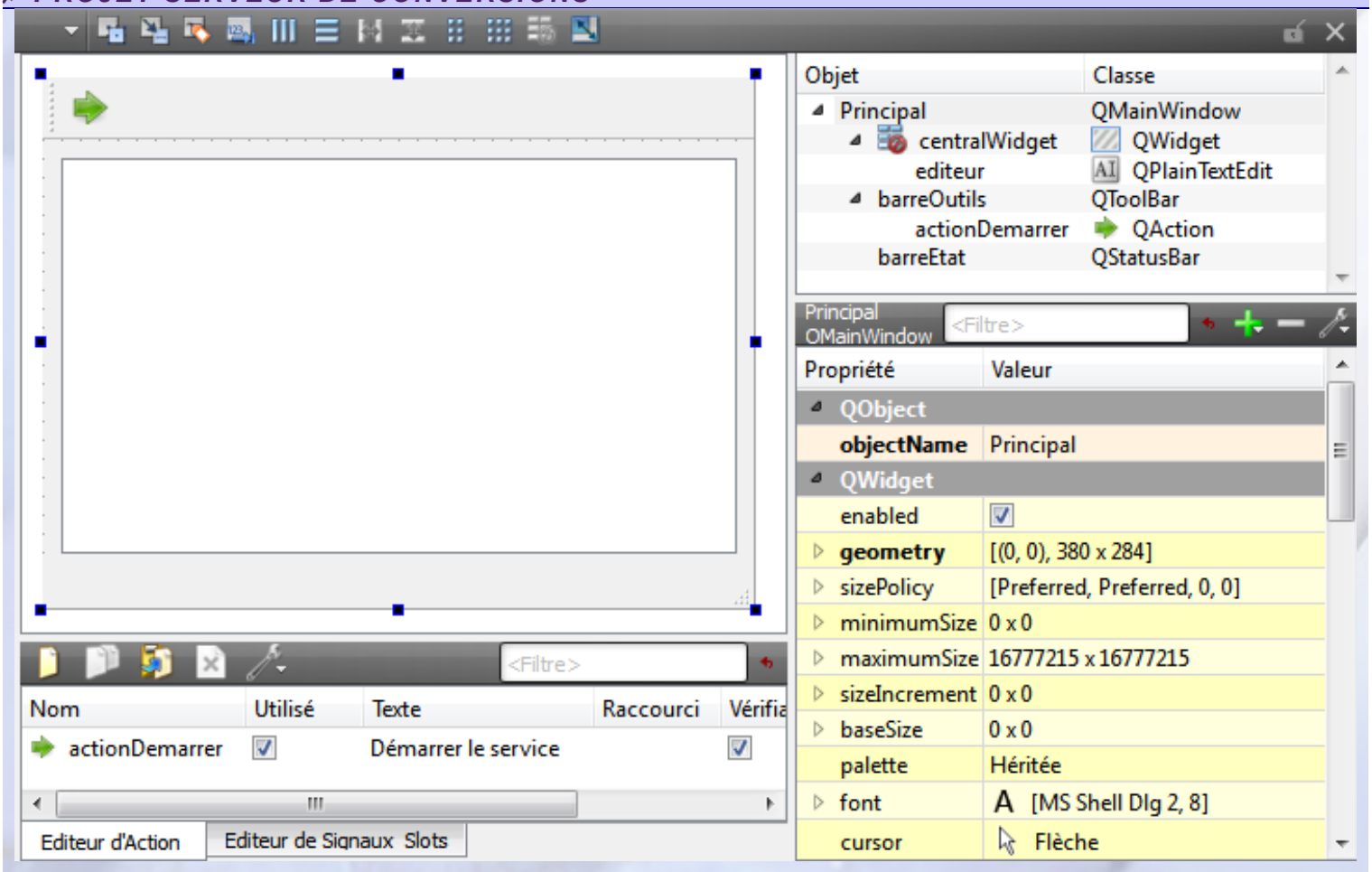
Pour comprendre cette notion de service, je vous propose de mettre en œuvre deux projets qui à eux deux vont permettre de réaliser la conversion entre les €uros et les Francs.

- **Le premier projet consiste à créer le service lui-même avec une IHM qui sera bien utile pour nous permettre à tout moment de lancer ou d'arrêter le service sur le port 5588. Par ailleurs, sur cette IHM, nous verrons apparaître les différentes requêtes proposées par les clients. C'est uniquement le service qui réalise la conversion entre les deux monnaies.**
- **Le deuxième projet consiste à réaliser l'IHM du système de saisie et de réponse en relation avec le serveur. Nous retrouvons alors pratiquement la même fenêtre que nous avons déjà implémentée plusieurs fois sur ce système de conversion. Cette fenêtre possède un champ supplémentaire qui permet de localiser le serveur au travers de son adresse IP ou au travers de son nom DNS. Un bouton permet alors d'établir la connexion avec le serveur distant. A partir de cet état, il est ensuite possible de solliciter les différentes conversions. Je rappelle que c'est uniquement le serveur qui réalise l'opération de conversion. Le client ne s'occupe uniquement que de la saisie des valeurs à calculer et que de l'affichage du résultat issu de l'opération de conversion.**





x PROJET SERVEUR DE CONVERSIONS



principal.h | <Selectionner un symbole> | Ligne : 1, Col : 1

```
1  #ifndef PRINCIPAL_H
2  #define PRINCIPAL_H
3
4  #include <QMainWindow>
5  #include "ui_principal.h"
6  #include "service.h"
7
8  class Principal : public QMainWindow, public Ui::Principal {
9      Q_OBJECT
10 public:
11     Principal(QWidget *parent = 0);
12 private:
13     Service service;
14 private slots:
15     void activerService(bool activation);
16 };
17
18 #endif // PRINCIPAL_H
19
```

principal.cpp* | Principal::activerService(bool) | Ligne : 25, Col : 2

```
1  #include "principal.h"
2  #include "ui_principal.h"
3
4  Principal::Principal(QWidget *parent) : QMainWindow(parent)
5  {
6    setupUi(this);
7    connect(actionDemarrer, SIGNAL(toggled(bool)), this, SLOT(activerService(bool)));
8    connect(&service, SIGNAL(message(QString)), barreEtat, SLOT(showMessage(QString)));
9    connect(&service, SIGNAL(enregistrer(QString)), editeur, SLOT(appendPlainText(QString)));
10   barreEtat->showMessage("ATTENTION : Le service est actuellement arrêté...");
11 }
12
13 void Principal::activerService(bool activation)
14 {
15   if (activation) {
16     service.demarrerService();
17     actionDemarrer->setIcon(QIcon(":/icones/stop.png"));
18     actionDemarrer->setToolTip("Arrêter le service");
19   }
20   else {
21     service.arreterService();
22     actionDemarrer->setIcon(QIcon(":/icones/run.png"));
23     actionDemarrer->setToolTip("Démarrer le service");
24   }
25 }
```

service.h* | Service | Ligne : 21, Col : 20

```
1   #ifndef SERVICE_H
2   #define SERVICE_H
3
4   #include <QTcpServer>
5
6   class Service : public QTcpServer
7   {
8     Q_OBJECT
9   signals:
10    void message(QString);
11    void enregistrer(QString);
12  public:
13    void demarrerService();
14    void arreterService();
15  private:
16     void incomingConnection(int idSocket);
17   private slots:
18     void cloture() { message("Service toujours actif : Le client s'est déconnecté..."); }
19  };
20
21  #endif // SERVICE_H
```

service.cpp* | Service::incomingConnection(int) | Ligne : 25, Col : 2

```
1   #include "service.h"
2   #include "client.h"
3
4   void Service::demarrerService()
5   {
6     if (listen(QHostAddress::Any, 5588)) {
7       message("Service 5588 démarré...");
8     }
9     else message("ATTENTION : Ce numéro de service est déjà utilisé...");
10  }
11
12  void Service::arreterService()
13  {
14    close();
15    message("Service 5588 interrompu...");
16  }
17
18  void Service::incomingConnection(int idSocket)
19  {
20    Client *connexion = new Client(this);
21    connexion->setSocketDescriptor(idSocket);
22    connect(connexion, SIGNAL(message(QString)), this, SIGNAL(enregistrer(QString)));
23    connect(connexion, SIGNAL(disconnected()), this, SLOT(cloture()));
24    message("Service actif : Un nouveau client vient de se connecter...");
25  }
```

client.h* | Client | Ligne : 17, Col : 19

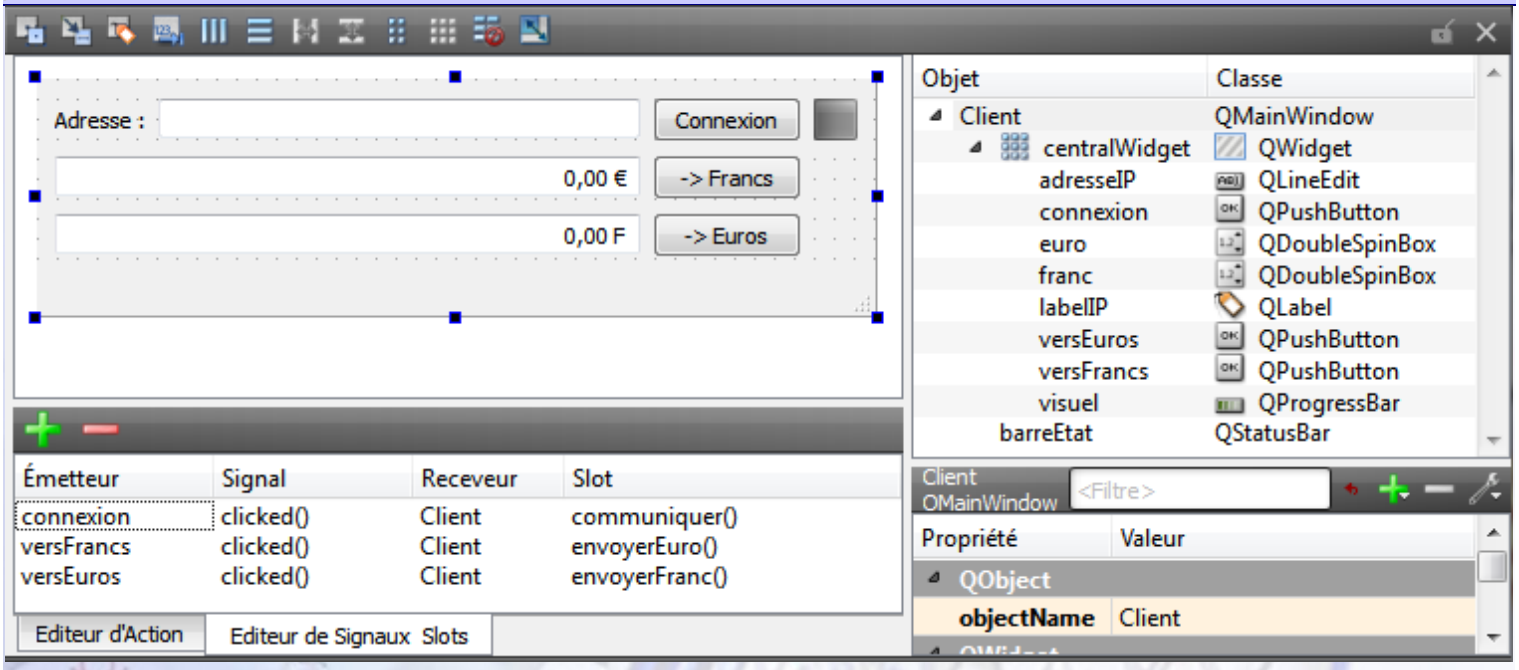
```
1   #ifndef CLIENT_H
2   #define CLIENT_H
3
4   #include <QTcpSocket>
5
6   class Client : public QTcpSocket
7   {
8     Q_OBJECT
9   public:
10    Client(QObject *parent = 0);
11  signals:
12    void message(QString);
13  private slots:
14    void communiquerAvecClient();
15  };
16
17  #endif // CLIENT_H
```

client.cpp* | Client::communiquerAvecClient() | Ligne : 27, Col : 2

```
1   #include "client.h"
2   #include <QHostAddress>
3   #include <QStringList>
4
5   Client::Client(QObject *parent) : QTcpSocket(parent)
6   {
7     connect(this, SIGNAL(readyRead()), this, SLOT(communiquerAvecClient()));
8   }
9
10  void Client::communiquerAvecClient()
11  {
12    static const double TAUX = 6.55957;
13    QString texte;
14    QString resultat;
15    if (canReadLine()) texte = readLine();
16    double monnaie = texte.split(' ')[0].toDouble();
17    if (texte.contains('F')) {
18      resultat = QString::number(monnaie/TAUX);
19      message(QString("%1 Francs --> %2 Euros").arg(monnaie).arg(resultat));
20    }
21    else {
22      resultat = QString::number(monnaie*TAUX);
23      message(QString("%1 Euros --> %2 Francs").arg(monnaie).arg(resultat));
24    }
25    QTextStream reponse(this);
26    reponse << resultat << endl;
27  }
```

- *La communication avec le client ne se fait qu'à ce niveau là. Il est en effet nécessaire d'avoir une socket de chaque côté (côté client et côté serveur) pour que le tube de communication se mette en place au travers du réseau.*
- *Vous remarquez que le protocole de communication est sous forme de chaîne de caractères. Dans un des Tps, je vous demanderez de changer ce protocole au profit d'une communication par bloc de données.*

x PROJET CLIENT CONVERSION



client.h* — Client — Ligne : 25, Col : 19

```
1   #ifndef CLIENT_H
2   #define CLIENT_H
3
4   #include <QMainWindow>
5   #include <QtNetwork>
6   #include "ui_client.h"
7
8   class Client : public QMainWindow, public Ui::Client
9   {
10    Q_OBJECT
11  public:
12    Client(QWidget *parent = 0);
13  private slots:
14    void communiquer();
15    void envoyerEuro();
16    void envoyerFranc();
17    void reponse();
18    void connexionEtablie();
19    void problemeConnexion() { barreEtat->showMessage("ATTENTION : Impossible de communiquer avec le serveur..."); }
20  private:
21    QTcpSocket service;
22    char demande;
23  };
24
25  #endif // CLIENT_H
```

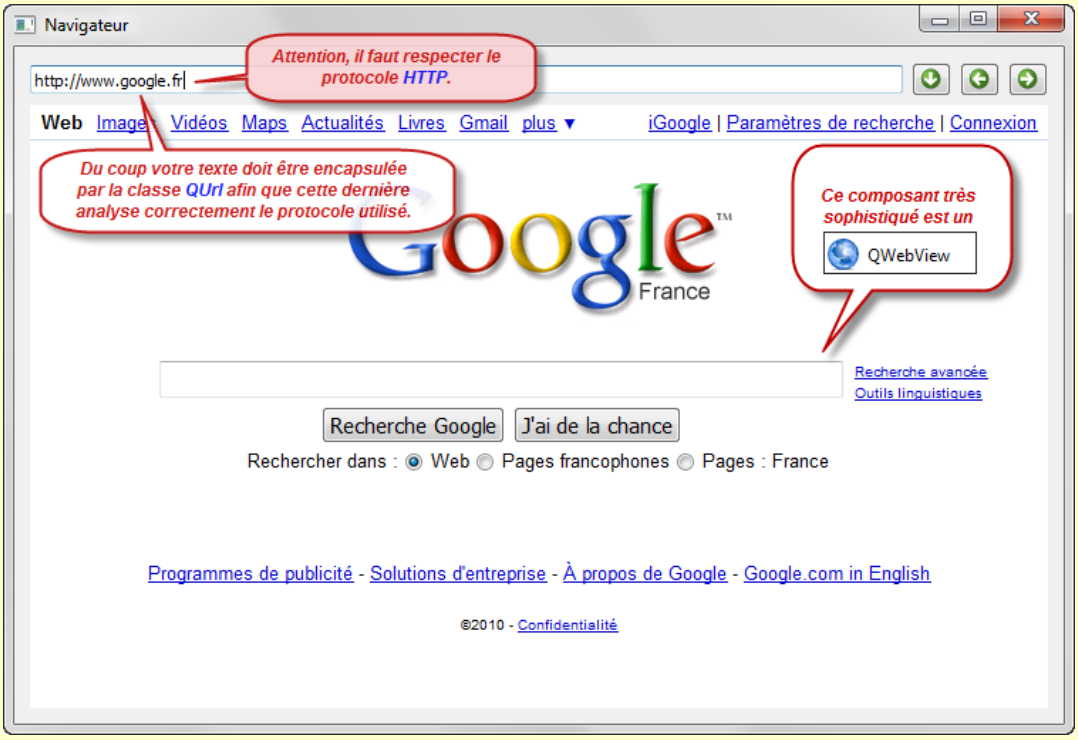
client.cpp* | Client::reponse() | Ligne : 56, Col : 2

```
#include "client.h"

Client::Client(QWidget *parent) : QMainWindow(parent)
{
  setupUi(this);
  visuel->hide();
  connect(&service, SIGNAL(readyRead()), this, SLOT(reponse()));
  connect(&service, SIGNAL(connected()), this, SLOT(connexionEtablie()));
  connect(&service, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(problemeConnexion()));
  barreEtat->showMessage("ATTENTION : La communication avec le serveur n'est pas encore établie...");
}

void Client::communiquer()
{
  connexion->setText(connexion->isChecked() ? "Déconnexion" : "Connexion");
  if (connexion->isChecked()) service.connectToHost(adresseIP->text(), 5588);
  else {
    service.close();
    visuel->setVisible(false);
    barreEtat->showMessage("ATTENTION : La communication avec le serveur est interrompue...");
  }
}

void Client::connexionEtablie()
{
  visuel->setVisible(true);
  barreEtat->showMessage("Communication établie avec le serveur de conversions...");
}

void Client::envoyerEuro()
{
  if (connexion->isChecked()) {
    demande = 'F';
    QTextStream donnees(&service);
    donnees << euro->text() << endl;
  }
}

void Client::envoyerFranc()
{
  if (connexion->isChecked()) {
    demande = 'E';
    QTextStream donnees(&service);
    donnees << franc->text() << endl;
  }
}

void Client::reponse()
{
  if (service.canReadLine()) {
    QString ligne;
    ligne = service.readLine();
    if (demande=='F') franc->setValue(ligne.toDouble());
    else euro->setValue(ligne.toDouble());
  }
}
```

TRAVAUX PRATIQUES

Comme d'habitude, nous allons maintenant élaborer un certain nombre de projets pour exploiter au mieux les connaissances acquises durant cette étude.

- ***Le premier projet consiste à réaliser un petit navigateur Web à l'aide du composant QWebView qui est capable d'interpréter les documents HTMLs et de prendre en compte automatiquement la navigation au travers des liens hypertextes. Pour ce projet, vous n'avez pas besoin d'intégrer le module QtNetwork. En contre partie, vous devez en prendre un autre qui se nomme QtWebkit.***
- ***La zone d'adresse est représentée par un simple QEditLine dont le texte saisie correspond à une URL. Pour que le composant QWebView affiche la bonne page Web et fonctionne correctement, il est nécessaire de transformer cette ligne de texte en un QUrl correspondant.***
- ***QWebView est en effet capable d'interpréter le protocole HTTP pour localiser l'adresse du site afin de récupérer ensuite automatiquement sa page d'accueil.***



- ***Le deuxième projet permet de piloter à distance le système de barrière NetPark en utilisant le protocole adéquat. Il faut se référer aux différents TPs prévus à ce sujet.***

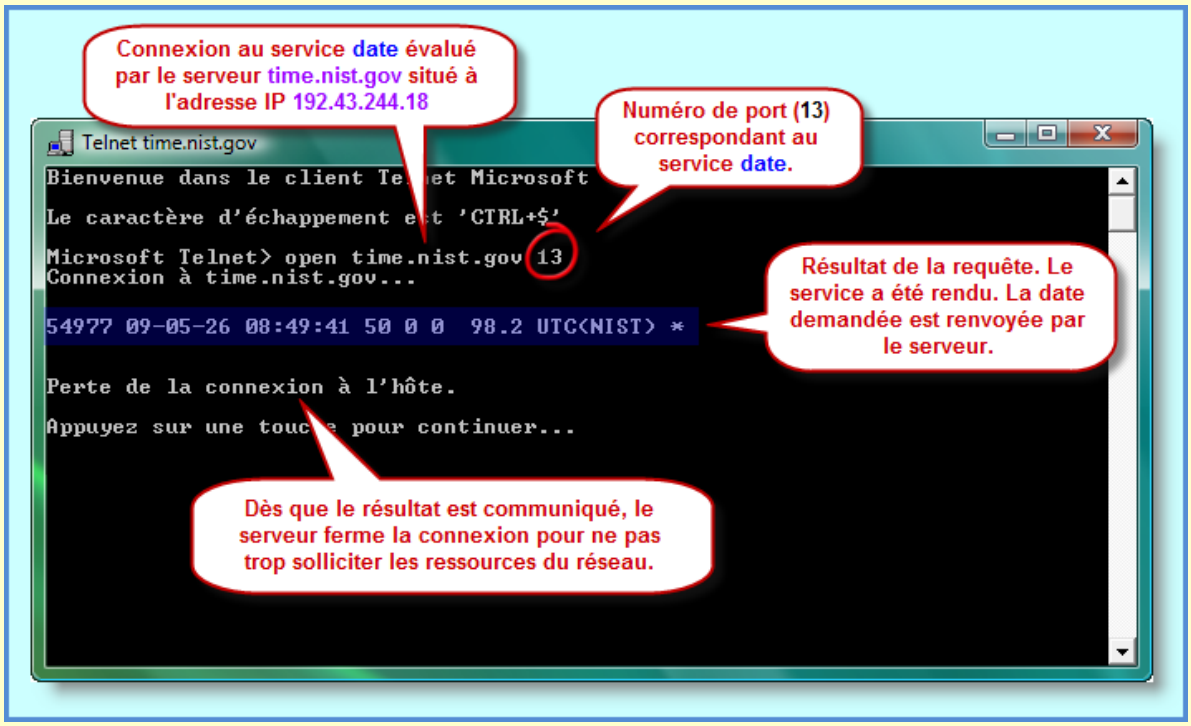
- ***Le troisième projet consiste à interroger toutes les secondes le service de date standard ITS (Internet Time Service) qui utilise le port 13.***

*La plupart des serveurs UNIX implémentent en permanence ce système de date. Le serveur auquel nous allons nous connecter se trouve au NIST à Boulder dans le Colorado, et fournit l'heure d'une horloge atomique au césium. Naturellement, le temps affiché n'est pas parfaitement précis à cause des délais de propagation des informations sur le réseau. Par convention, le service date est toujours rattaché au port **13**.*

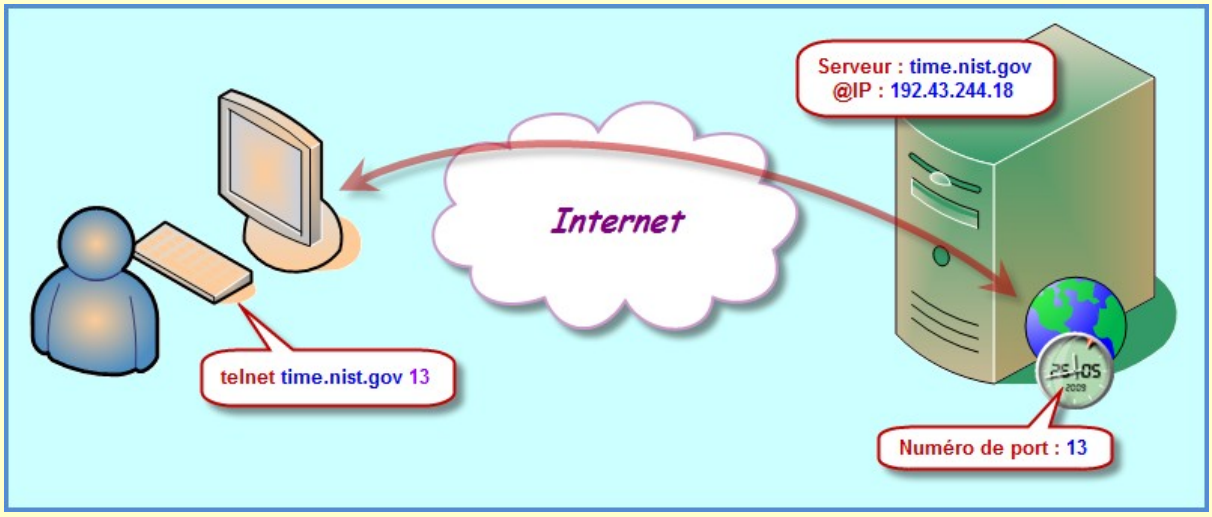
Je vous propose de lancer le logiciel client Telnet et d'établir ensuite la connexion au serveur time.nist.org (Situé aux Etats-Unis à

@IP 192.43.244.18). Pour cela nous utilisons la commande open. Rappelez-vous que par défaut ce service est réglé sur le port 13.

open time.nist.gov 13 ou open 192.43.244.18 13.

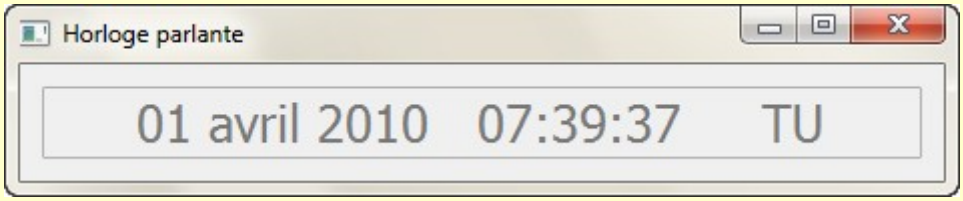


✗ Le programme du serveur fonctionne en permanence sur la machine distante, attendant un paquet du réseau qui essaierait de communiquer avec le port 13. Lorsque le système d'exploitation de l'ordinateur distant reçoit le paquet contenant une requête de connexion sur le port 13, le processus d'écoute du serveur est activé et la connexion est établie. Cette connexion demeure jusqu'à ce qu'elle soit arrêtée par l'une des deux parties.



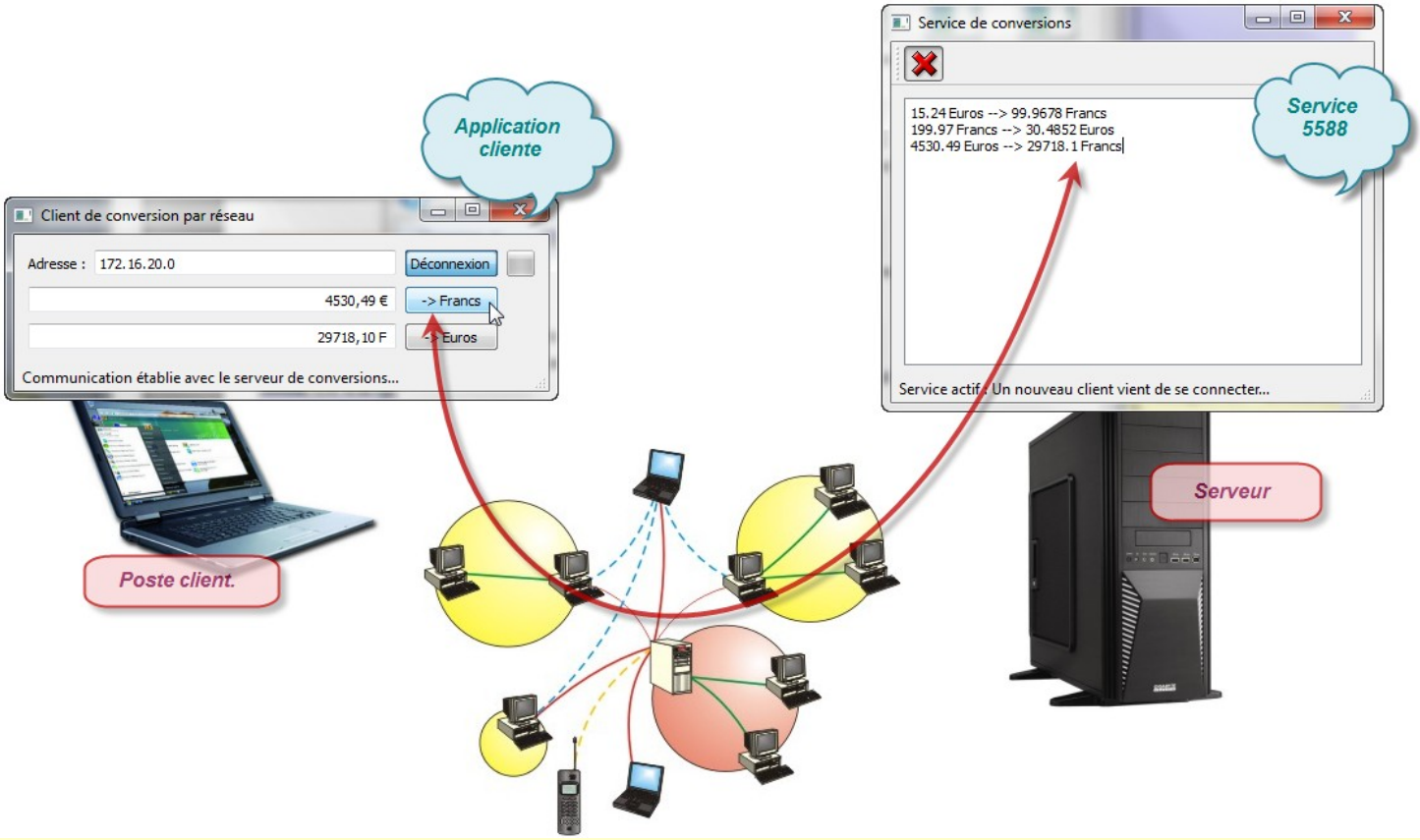
✗ Une fois que cette connexion a été établie, le programme de l'ordinateur distant a renvoyé un ensemble de données, puis il a terminé la connexion.

✗ Pour revenir à notre projet, je vous invite à créer une application qui interroge toutes les secondes ce service afin d'avoir une horloge avec le temps exacte. Attention, il s'agit alors du temps universel sans le décalage horaire proposé par la plupart des pays.



✗

✗ ***Dans le dernier projet, vous allez reprendre le client/serveur de conversions monétaires, mais cette fois-ci, vous allez une communication entre les deux applications par bloc de données.***



✗