# Qt et la communication réseau

par rocsan

Date de publication : 01/06/2010

Dernière mise à jour : 16/12/2010

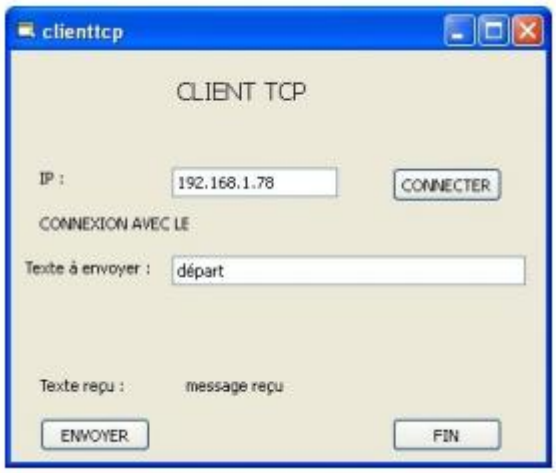
Cet article présente l'utilisation du réseau avec Qt, principalement à l'aide de TCP.
**Commentez !**

# I - Création d'un client TCP

## I-A - Présentation de l'application

On se propose de créer un client capable de se connecter à un serveur (étudié plus loin dans ce tutoriel). Le client enverra une chaîne de caractères au serveur et affichera la réponse du serveur comme texte dans un label. L'adresse IP du serveur est à saisir (on utilisera le port 4000). Un label permettra de savoir si la connexion avec le serveur est établie.

L'IHM (et les suivantes) sera créée avec QtDesigner (par exemple) mais ne sera pas étudiée ici.




La classe Client dérivera de QObject pour pouvoir être connectée à l'IHM. Elle aura comme attribut une socket (QTcpSocket). On utilise la méthode connectToHost() à qui on passe l'adresse du serveur et le port utilisé. L'objet QTcpSocket émet le signal connected() quand la connexion est établie. Le signal readyRead() est émis dès que l'objet QTcpSocket a reçu de nouvelles données en provenance du serveur. On utilise la méthode readLine() de la socket (qui renvoie un QString) pour lire les données en provenance du serveur. On pourra tester en permanence la possibilité de lecture avec la méthode canReadLine() de la socket qui renvoie un booléen.

## I-B - Code

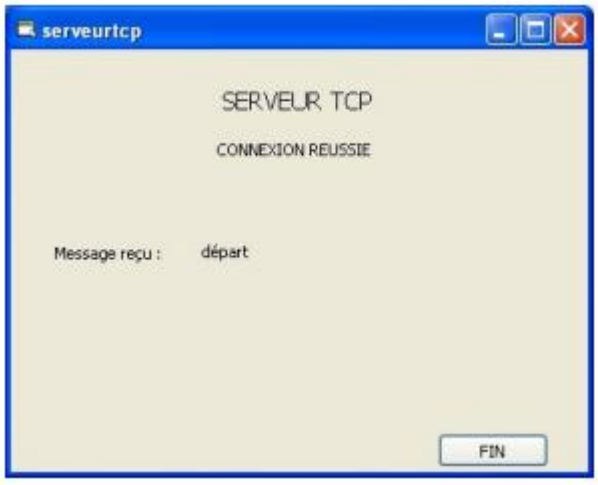
```
class ClientTcp : public QObject
{
Q_OBJECT
public :
 ClientTcp();
public slots :
 void recoit_IP(QString IP2);  // en provenance de l'IHM et se connecte au serveur
 void recoit_texte(QString t); // en provenance de l'IHM et écrit sur la socket
private slots :
 void connexion_OK();  // en provenance de la socket et émet un signal vers l'IHM
 void lecture();       // en provenance de la socket, lit la socket, émet un signal vers l'IHM
signals :
 void vers_IHM_connexion_OK();
 void vers_IHM_texte(QString);
private :
 QString IP;
 int port;
 QTcpSocket soc;
};

ClientTCP::ClientTcp()
{
 port=4000; // choix arbitraire (>1024)
 QObject::connect(&soc,SIGNAL(connected()),this,SLOT(connexion_OK()));
 // signal émis lors de la connexion au serveur
```

```
 QObject:: connect(&soc, SIGNAL(readyRead()), this, SLOT(lecture()));
 // signal émis lorsque des données sont prêtes à être lues
}
void ClientTCP::recoit_IP(QString IP2)
{
 IP=IP2;
 soc.connectToHost(IP,port); // pour se connecter au serveur
}
void ClientTCP::recoit_texte(QString t)
{
 QTextStream texte(&soc); // on associe un flux à la socket
 texte << t<<endl;        // on écrit dans le flux le texte saisi dans l'IHM
}
void ClientTCP::connexion_OK()
{
 emit vers_IHM_connexion_OK(); // on envoie un signal à l'IHM
}
void ClientTCP::lecture()
{
 QString ligne;
 while(soc.canReadLine()) // tant qu'il y a quelque chose à lire dans la socket
 {
  ligne = soc.readLine();     // on lit une ligne
  emit vers_IHM_texte(ligne); // on envoie à l'IHM
 }
}
```

# II - Création d'un serveur TCP

## II-A - Présentation de l'application

On souhaite maintenant créer le serveur TCP auquel se connectera le client du paragraphe précédent. La classe Serveur sera dérivée de la classe QTcpServer. La méthode listen() met le serveur à l'écoute des demandes de connexions. Le signal newConnection() est émis à chaque fois qu'un nouveau client se connecte. Il faut alors appeler la méthode nextPendingConnection() qui créera une nouvelle socket connectée au client.




Dans la classe serveur, la méthode demande_connexion() est un slot appelé à chaque connexion d'un client. Elle va permettre d'affecter une socket à ce client. La méthode lecture() permettra de lire le texte en provenance du client et de l'envoyer à l'IHM par émission du signal vers_IHM_texte(QString). Dès que le texte est lu, on écrit sur la socket le texte « message reçu ». La méthode est la même que pour le client.

## II-B - Code

```
class ServeurTcp : public QTcpServer
{
```

```
Q_OBJECT
public :
 ServeurTcp(QObject *parent=0)
private slots :
 void demande_connexion() ;
 void lecture();
signals :
 void vers_IHM_connexion();
 void vers_IHM_texte(QString);
private :
 QTcpSocket *clientConnection;
};

ServeurTcp :: ServeurTcp (QObject *parent)
{
 listen(QHostAddress::Any,4000);
 QObject:: connect(this, SIGNAL(newConnection()),
 this, SLOT(demande_connexion()));
}

// si un client demande une connexion
void ServeurTcp :: demande_connexion()
 {
 emit vers_IHM_connexion(); // on envoie un signal à l'IHM
 // on crée une nouvelle socket pour ce client
 clientConnection = nextPendingConnection();
 // si on reçoit des données, le slot lecture() est appelé
 QObject:: connect(clientConnection, SIGNAL(readyRead()),
 this, SLOT(lecture()));
}
void ServeurTcp ::lecture()
{
 QString ligne;
 while(clientConnection->canReadLine())    // tant qu'on peut lire sur la socket
 {
  ligne = clientConnection->readLine(); // on lit une ligne
  emit vers_IHM_texte(ligne);           // on l'envoie à l'IHM
 }
 QTextStream texte(clientConnection);      // création d'un flux pour écrire dans la socket
 texte << "message reçu" << endl;          // message à envoyer au client
}
```

# III - Création d'un client FTP

## III-A - Présentation de l'application

On souhaite réaliser l'application suivante : on indique l'IP du serveur FTP, son login et son mot de passe pour se connecter (le texte du bouton change et indique qu'on est effectivement connecté). On saisit ensuite le nom du fichier à transférer. Une barre indique l'état de la progression.

On peut écrire des applications FTP (File Transfert Protocol) avec un objet QFtp. Quelques méthodes de la classe QFtp :

- connectToHost(ftp_adress) ;
- login(login, passwd) ;
- cd(directory) ;
- get(FileName,&qfile) ;
- close().

Dès qu'une commande FTP est transmise, le signal commandStarted(int) est émis. Quand elle est terminée, c'est commandFinished(int, bool) qui est émis. Pendant la transmission, le signal dataTransferProgress(qint64 done, qint64 total) est émis. total représente le nombre total d'octets à transmettre et donne le nombre d'octets transmis (0 si le total ne peut pas être connu). Lors de la connexion, des signaux stateChanged(int) sont émis à différents moments. La valeur de l'entier transmis est décrite ci-dessous :

| Constant | Value | Description |
|---|---|---|
| QFtp::Unconnected | 0 | There is no connection to the host. |
| QFtp::HostLookup | 1 | A host name lookup is in progress. |
| QFtp::Connecting | 2 | An attempt to connect to the host is in progress. |
| QFtp::Connected | 3 | Connection to the host has been achieved. |
| QFtp::LoggedIn | 4 | Connection and user login have been achieved. |
| QFtp::Closing | 5 | The connection is closing down, but it is not yet closed. (The state will be Unconnected when the connection is closed.) |

Pour se connecter, on attendra d'être à l'état QFtp::LoggedIn. Pour transférer un fichier (get), on testera le signal commandFinished() émis après la commande get (utilisation d'un drapeau).

## III-B - Code

```
class ClientFtp : public QObject
{
Q_OBJECT
public :
 ClientFtp() ;
public slots :
 void se_connecter(QString IP,QString log,QString pass);
 void get_fichier(QString phrase);
private slots :
 void transfertOK(int,bool);
 void fait (int v) ;
signals :
 void connexion_ok();
 void progress_vers_IHM( qint64,qint64 );
private :
 QFtp f;
 QFile fichier;
 int drapeau;
};

ClientFtp ::ClientFtp()
{
 drapeau=0; // initialisation du drapeau
}
void ClientFtp ::se_connecter(QString IP,QString log,QString pass)
```

```
{
 // pour détecter la connexion au serveur
 QObject::connect(&f,SIGNAL(stateChanged(int)),this,SLOT(fait(int)));
 // pour détecter la fin de transfert
 Qobject::connect(&f,SIGNAL(commandFinished(int,bool)),
 this,SLOT(transfertOK(int,bool)));
 // pour connaître la progression du transfert (vers une QProgressBar)
 Qobject::connect(&f,SIGNAL(dataTransferProgress(qint64,qint64)),
 this,SIGNAL(progress_vers_IHM( qint64,qint64)));
 f.connectToHost(IP);
 f.login(log,pass);
}
void ClientFtp ::get_fichier(QString phrase)
{
 fichier.setFileName (phrase);       // on garde le même nom de fichier sur le disque dur
 fichier.open(QIODevice::WriteOnly); // ouverture en écriture
 f.get(phrase,&fichier);             // on commence le transfert FTP
 drapeau=1;                          // mémorisation de début de transfert
}
void ClientFtp ::transfertOK()
{
 if (drapeau==1) // si on a commencé le transfert
 {
  fichier.close();   // on ferme le fichier
  f.close();         // on ferme la connexion FTP
  drapeau=0;         // on réinitialise le drapeau à 0
 }
}
// à chaque changement d'état de la connexion
void ClientFtp ::fait (int v)
{
 if (v==4) // si on a réussi à se logguer
  emit connexion_ok(); // envoi d'un signal vers l'IHM
}
```

# IV - Création d'un client HTTP

## IV-A - Présentation de l'application

On souhaite réaliser l'application suivante : on indique un numéro de mesure à un serveur Web (IP entrée en « dur » et page demandée : suite.php). On affiche alors le type et la valeur de la mesure correspondante. La méthode utilisée est GET et le nom de la variable est numero.


Le serveur Web renvoie une donnée de la forme « courant=4,5 ». On peut imaginer que le serveur appelle un CGI capable d'effectuer une mesure réelle ou qu'il interroge une base de données. On peut écrire des applications HTTP (Hyper Text Transfert Protocol) avec un objet QHttp. Comme pour QFtp, il est possible de transférer un fichier (méthode get()), mais ceci ne sera pas étudié ici. Quelques méthodes de la classe QHttp :

- setHost(http_adress) ;
- request(http_header) : envoie une requête HTTP ;
- close() : doit être fait lorsque le fichier est totalement transmis.

Les signaux de la classe QHttp ressemblent à ceux de la classe QFtp. Pour l'application étudiée, le signal utilisé sera readyRead( const QHttpResponseHeader &) émis dès que des données sont présentes (après l'appel à la méthode GET). Pour lire les données, on utilise la méthode read(char *donnee, int longueur_max). Pour plus de commodité, il est intéressant de copier le contenu de la chaîne dans une variable de type QString.

La requête à envoyer est de type QHttpRequestHeader dont le constructeur demande deux paramètres (QString) : la méthode utilisée et l'URL. Cette URL contiendra la page Web et les éventuels paramètres de la forme « nom_variable1=valeur1&nom_variable2=valeur2 ». Il faut appeler la méthode setValue du header, à qui on passe deux QString : la clé (ici « Host ») et sa valeur (l'adresse IP du serveur Web).

## IV-B - Code

```
class HttpClient: public QObject
{
Q_OBJECT
public:
 HttpClient();
private slots :
 void fin( const QHttpResponseHeader &); // pour lire des données du serveur
public slots :
 void question(QString); // connecté à un signal de l'IHM après la saisie
signals :
 void vers_ihm_get1(QString);  // envoi à l'IHM de la première donnée
 void vers_ihm_get2(QString);  // envoi à l'IHM de la deuxième donnée
private :
 QHttp http;
};

HttpClient::HttpClient()
{
 http.setHost("127.0.0.1"); // à modifier selon l'IP
 // slot traitant la réponse du serveur
 QObject::connect(&http,SIGNAL(readyRead( const QhttpResponseHeader&)),
 this,SLOT(fin( const QHttpResponseHeader &)));
}
void HttpClient::question(QString mot)
{
 QString ques="/suite.php?numero="+mot;  // paramètres de la méthode GET
 QHttpRequestHeader header("GET", ques); // exemple : /suite.php?numero=11
 header.setValue("Host", "127.0.0.1");
 http.request(header); //envoi de la requête
}
// lecture de la réponse de la forme : "frequence=1500"
void HttpClient::fin( const QHttpResponseHeader &e)
{
 char *str=new char [5000];
 http.read(str,5000); // on lit tout
 QString toto=str;
 QString sig;
 QString sig2;

 // on lit le contenu de la première variable
 if( toto.contains("courant", Qt::CaseInsensitive))
  sig="courant";
 if( toto.contains("tension", Qt::CaseInsensitive))
  sig="tension";
 if( toto.contains("frequence", Qt::CaseInsensitive))
  sig="frequence";
 emit vers_ihm_get1(sig); // envoi de la première variable à l'IHM

 int pos=toto.indexOf(sig,Qt::CaseInsensitive);
 int pos2=toto.indexOf("=",pos, Qt::CaseInsensitive)+1;   // on se place après le signe =
 int i=pos2,j=0;
```

```
 while(toto.at(i)!='<')
 {
  sig2[j]=toto.at(i); // on lit ce qu'il y a après le signe =
  i++;j++;
 }
 emit vers_ihm_get2(sig2); //envoi de la deuxième variable à l'IHM
 http.close(); //fermeture de la connexion http
}
```

## V - Divers

L'utilisation des classes QFtp et QHttp est désormais dépréciée au profit de QNetworkAccessManager. Voir à ce sujet le tutoriel **Un updater avec Qt : le téléchargement de fichiers** ainsi que **les entrées de la FAQ**.

Merci à **dourouc05** et **jacques_jean** pour leur relecture orthographique !