



Lecture Qt011

Networking

Instructor: David J. Coe

CPE 353 – Software Design and Engineering

Department of Electrical and Computer Engineering

Outline

- Overview
- UDP Communication
 - Hands-On Example: UDP and **QDataStream**
 - Hands-On Example: UDP and **QTextStream**
- TCP Client-Server Applications
 - Hands-On Example: TCP and **QDataStream**
- Key Points

QtNetwork Module

- **QtNetwork** module provides support for
 - UDP sockets
 - TCP servers/sockets
 - SSL sockets (inherits from TCP)
 - FTP protocol
 - HTTP protocol
- See module description in Qt Assistant for additional details

QtNetwork Module

- Check your project file (**.pro**) before you attempt to compile
- You will need to add the following line if it is missing

QT += network

UDP User Datagram Protocol

- Low-overhead protocol
- No “connection”
- Not for applications where reliability is required since packet delivery is not guaranteed
 - Packets may arrive out-of-order
 - Duplicate packets may be received
 - Packets may not arrive at all
- Typical uses of UDP
 - Streaming media
 - VoIP
 - Real-time multiplayer gaming

Example: UDP/QDataStream

- **Sender** Application
 - **QLineEdit** widget allows user to input a text message
 - Send button forwards message to Receiver application via UDP and clears sent message from **QLineEdit** widget
 - Quit button terminates Sender application
 - Uses **QDataStream** abstraction object
- **Receiver** Application
 - **QLabel** widget initially displays “***Ready to Receive***”
 - Each incoming UDP message replaces previously displayed text in the **QLabel** widget

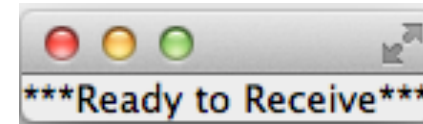
Example: UDP/QDataStream

```
// main.cpp -- Receiver
#include <QApplication>
#include "udplabel.h"

int main(int argc, char* argv[])
{
    QApplication myapp(argc, argv);

    UDPLabel myUDPLabel("***Ready to Receive***");
    myUDPLabel.show();

    return myapp.exec();
}
```



Example: UDP/QDataStream

```
// udplabel.h - Receiver

#ifndef UDPLABEL_H
#define UDPLABEL_H

#include <QLabel>
#include <QUdpSocket>

class UDPLabel : public QLabel
{
    Q_OBJECT;

public:
    UDPLabel(QString msg) ;

private:
    QUdpSocket *myUDPSocket;

private slots:
    void processPendingDatagrams() ;
};

#endif
```


Example: UDP/QDataStream

```
// udplabel.cpp -- Receiver

#include <QtGui>
#include "udplabel.h"
#include <QString>
#include <QByteArray>
#include <QDataStream>

UDPLabel::UDPLabel(QString msg) : QLabel(msg)
{
    myUDPSocket = new QUdpSocket(this);

    myUDPSocket->bind(5678);

    connect( myUDPSocket, SIGNAL(readyRead()),
             this, SLOT(processPendingDatagrams()) );
}
```

Example: UDP/QDataStream

```
// udplabel.cpp -- Receiver continued

void UDPLabel::processPendingDatagrams()
{
    QByteArray mydatagram;

    while ( myUDPSocket->hasPendingDatagrams() )
    {
        mydatagram.resize(myUDPSocket->pendingDatagramSize());
        myUDPSocket->readDatagram(mydatagram.data(), mydatagram.size());
    }

    QString msg;
    QDataStream in(&mydatagram, QIODevice::ReadOnly);

    in.setVersion(QDataStream::Qt_4_4);

    in >> msg;
    this->setText(msg);
}
```

Example: UDP/QDataStream

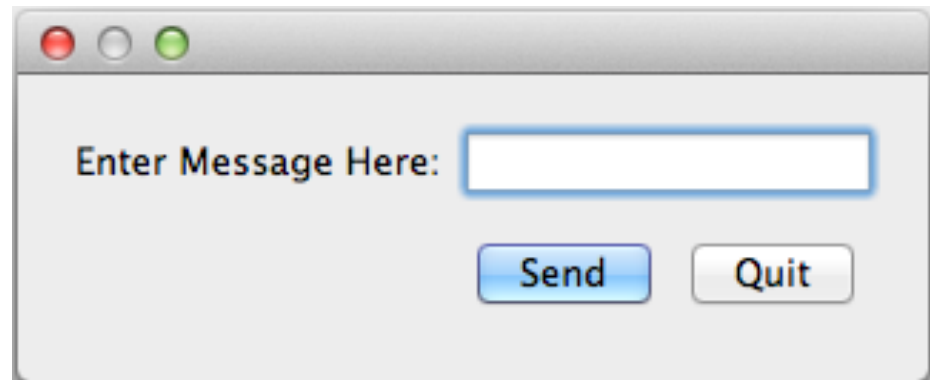
```
// main.cpp -- Sender

#include <QApplication>
#include "senderdialog.h"

int main(int argc, char* argv[])
{
    QApplication myapp(argc, argv);

    SenderDialog mysender;
    mysender.show();

    return myapp.exec();
}
```



Example: UDP/QDataStream

```
// senderdialog.h - Sender
```

```
#ifndef SENDERDIALOG_H  
#define SENDERDIALOG_H
```

```
#include <QDialog>  
#include <QLineEdit>  
#include <QPushButton>  
#include <QUdpSocket>  
#include <QLabel>  
#include <QHBoxLayout>  
#include <QVBoxLayout>
```

```
class SenderDialog : public QDialog  
{  
    Q_OBJECT  
  
public:  
    SenderDialog(QWidget *parent = 0);  
  
private:  
    QLabel* label;  
    QLineEdit* lineEdit;  
    QPushButton* sendButton;  
    QPushButton* quitButton;  
  
    QVBoxLayout* mainLayout;  
    QHBoxLayout* editLayout;  
    QHBoxLayout* buttonLayout;  
  
    QUdpSocket* sendSocket;  
  
private slots:  
    void writeMessage();  
};  
#endif
```

Example: UDP/QDataStream

```
// senderdialog.cpp

#include "senderdialog.h"
#include <QUdpSocket>
#include "senderdialog.h"

SenderDialog::SenderDialog(QWidget* parent) : QDialog(parent)
{
    label = new QLabel("Enter Message Here:");
    lineEdit = new QLineEdit;
    sendButton = new QPushButton("&Send");
    quitButton = new QPushButton("&Quit");
    mainLayout = new QVBoxLayout(this);
    editLayout = new QHBoxLayout;
    buttonLayout = new QHBoxLayout;

    mainLayout->addLayout(editLayout);
    mainLayout->addStretch();
    mainLayout->addLayout(buttonLayout);
    editLayout->addWidget(label);
    editLayout->addWidget(lineEdit);
    buttonLayout->addStretch();
    buttonLayout->addWidget(sendButton);
    buttonLayout->addWidget(quitButton);

    connect(sendButton, SIGNAL(clicked()), this, SLOT(writeMessage()));
    connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));

    sendSocket = new QUdpSocket;
}
```

Example: UDP/QDataStream

```
// senderdialog.cpp -- continued

void SenderDialog::writeMessage()
{
    QByteArray datagram;

    QDataStream out(&datagram, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_4);

    QString msg = lineEdit->text();
    lineEdit->setText("");

    out << msg;

    sendSocket->writeDatagram(datagram, QHostAddress::LocalHost, 5678);
}
```

Example: UDP/QTextStream

```
// senderdialog.cpp

...

void SenderDialog::writeMessage()
{
    QByteArray datagram;
    QTextStream out(&datagram, QIODevice::WriteOnly);

    QString msg = lineEdit->text();
    lineEdit->setText("");

    out << msg << endl;

    sendSocket->writeDatagram(datagram, QHostAddress::LocalHost, 5678);
}
```

Minor changes to one method

Example: UDP/QTextStream

```
// udplabel.cpp -- Receiver
...

void UDPLabel::processPendingDatagrams()
{
    QByteArray mydatagram;

    while (myUDPSocket->hasPendingDatagrams())
    {
        mydatagram.resize(myUDPSocket->pendingDatagramSize());
        myUDPSocket->readDatagram(mydatagram.data(), mydatagram.size());
    }

    QString msg;
    QTextStream in(&mydatagram, QIODevice::ReadOnly);

    in >> msg;

    this->setText(msg);
}
```

Minor changes to one method

Lessons Learned

- **QTextStream** interface objects may be used for either **UDP** or **TCP** communications with only minor changes to the send/receive code that utilizes **QDataStream** objects

TCP Transmission Control Protocol

- Stream-oriented protocol
- One of the core internet protocols
- Unlike UDP
 - TCP is connection oriented
 - Connection must be established
 - Data transferred
 - Connection terminated
 - TCP provides reliable delivery
 - Handles duplicate, lost, and out-of-order packets
- Typical uses of TCP
 - File transfer, email

Server Applications

- Server applications typically run invisibly in the background
 - Don't require a GUI
- Save memory and disk space by avoiding linking with GUI classes
 - Use **QCoreApplication** object in **main()** instead of **QApplication** object
 - Add **QT -= gui** to project file

QTcpServer

- Class allows your application to listen for an incoming TCP connection
 - The server can listen to a particular port or any port using **listen()**
 - It can also listen for a specific machine address or any address
 - Methods **serverAddress()** and **serverPort()** identify particular address and port
 - **close()** will terminate listening
- Typically used with an event loop but can block using **waitForNewConnection()**
 - Can specify timeout interval

QTcpServer

- **nextPendingConnection()**
 - Returns next connection as a pointer to a **QTcpSocket** object
- **newConnection()**
 - This *signal* is emitted when a client connects to the server
- **isListening()**
 - Returns true if currently listening; false otherwise
- **hasPendingConnections()**
 - Returns true if there are connections pending; false otherwise

QTcpSocket

- **connectToHost()** allows one to establish a connection to a particular host using
 - **QHostAddress** object
 - **QString** representation of an IP address or host name (lookup will be performed)
- **disconnectFromHost()**
 - Attempts to close socket; waits for any pending data to be written

QTcpSocket

- **connected()**
 - Signal emitted after **connectToHost()** has been called and a connection has been successfully established
- **disconnected()**
 - This signal is emitted when the socket has been disconnected
- **readyRead()**
 - Signal emitted once every time new data is available for reading from the device.
 - Only emitted again once new data is available
- **error()**
 - Emitted after error occurs
 - Includes error description

Example: TCP/QDataStream

```
// TCP Example -- Greeting Server Application, main.cpp
```

```
#include <QCoreApplication>
```

```
#include <QDebug>
```

```
#include "greetingserver.h"
```

```
int main(int argc, char* argv[])
```

```
{
```

```
    QCoreApplication myApp(argc, argv);
```

```
    GreetingServer server;
```

```
    qDebug() << "Server running";
```

```
    return myApp.exec();
```

```
} // End main()
```


Example: TCP/QDataStream

```
// TCP Example -- Greeting Server Application, greetingserver.h

#ifndef GREETINGSERVER_H
#define GREETINGSERVER_H

#include <QTcpServer>

class GreetingServer : public QTcpServer
{
    Q_OBJECT

public:
    GreetingServer(QObject* parent = 0);

private:
    QString greetings[4];

private slots:
    void sendGreeting();
};

#endif
```

Example: TCP/QDataStream

```
// TCP Example -- Greeting Server Application, greetingserver.cpp

#include <QtNetwork>
#include "greetingserver.h"

GreetingServer::GreetingServer(QObject* parent) : QTcpServer(parent)
{
    greetings[0] = "Hello";           // Initialize greetings array
    greetings[1] = "Howdy";
    greetings[2] = "Salutations";
    greetings[3] = "Aloha";

    // Initialize random number generator to number of seconds
    // between 00:00:00 and now
    qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));

    // Send greeting upon new connection
    connect(this, SIGNAL(newConnection()), this, SLOT(sendGreeting()));

    // Listen for incoming connects on port 1234 of this machine
    this->listen(QHostAddress::LocalHost, 1234);
} // End GreetingServer::GreetingServer()
```

Example: TCP/QDataStream

```
// TCP Example -- Greeting Server Application, greetingserver.cpp - continued
```

```
void GreetingServer::sendGreeting()
{
    // Socket created as child of GreetingServer object
    QTcpSocket* client = this->nextPendingConnection();

    QByteArray block;
    QDataStream outgoingMessage(&block, QIODevice::WriteOnly);

    outgoingMessage.setVersion(QDataStream::Qt_4_1);

    // Select random greeting
    outgoingMessage << greetings[qrand() % 4];

    // Write greeting to socket
    client->write(block);

    // Attempt to close socket but wait until pending data written
    client->disconnectFromHost();
} // End GreetingServer::sendGreeting()
```

Example: TCP/QDataStream

```
// TCP Example -- Client, main.cpp

#include <QApplication>
#include "client.h"

int main(int argc, char* argv[])
{
    QApplication myApp(argc, argv);

    Client c;
    c.show();

    return myApp.exec();
} // End main()
```

Example: TCP/QDataStream

```
// client.h

#ifndef CLIENT_H
#define CLIENT_H

#include <QDialog>
#include <QPushButton>
#include <QLabel>
#include <QTcpSocket>
#include <QVBoxLayout>

class Client : public QDialog
{
    Q_OBJECT

public:
    Client(QWidget* parent = 0);

private:
    QTcpSocket*    socket;
    QLabel*        label;
    QPushButton*   button;
    QVBoxLayout*   mainLayout;

private slots:
    void updateLabel();
    void requestGreeting();
};

#endif
```

Example: TCP/QDataStream

```
// TCP Example -- Client, main.cpp

#include "client.h"
#include <QtNetwork>

Client::Client(QWidget* parent) : QDialog(parent)
{
    mainLayout = new QVBoxLayout(this);
    label = new QLabel("*** Ready ***");
    button = new QPushButton("Greet Me");
    mainLayout->addWidget(label);
    mainLayout->addStretch();
    mainLayout->addWidget(button);

    connect(button, SIGNAL(clicked()), this, SLOT(requestGreeting()));
} // End Client::Client()
```

Example: TCP/QDataStream

```
// TCP Example -- Client, main.cpp - continued
```

```
void Client::requestGreeting()
{
    socket = new QTcpSocket(this);
    connect(socket, SIGNAL(readyRead()), this, SLOT(updateLabel()));
    connect(socket, SIGNAL(disconnected()), socket, SLOT(deleteLater()));

    socket->connectToHost(QHostAddress::LocalHost, 1234);
} // End Client::requestGreeting()

void Client::updateLabel()
{
    QDataStream incomingMessage(socket);
    incomingMessage.setVersion(QDataStream::Qt_4_1);

    QString msg;
    incomingMessage >> msg;
    label->setText(msg);

    delete socket;
} // End Client::updateLabel()
```

Observations

- The previous approach works for a small number of incoming connections
- To handle a large number of connections, create a separate thread for each connection

Key Points

- **QTcpSocket/QTcpServer** and **QUdpSocket** networking classes may be used to interface your Qt program with applications running locally or running on remote systems
 - Option One:
 - Declare, configure, and employ objects of these types
 - Option Two:
 - Derive customized versions of the types
 - Employ objects of your derived class types