TCP Socket Example 1

- In this example, we will learn how to use **QTcpSocket**. We're taking blocking approach (synchronous). In other words, we use **waitFor...()** functions (e.g., **QTcpSocket::waitForConnected()**) until the operation has completed, instead of connecting to signals.

- The **QTcpSocket** class provides a TCP socket.

- TCP (Transmission Control Protocol) is a reliable, stream-oriented, connection-oriented transport protocol. It is especially well suited for continuous transmission of data.

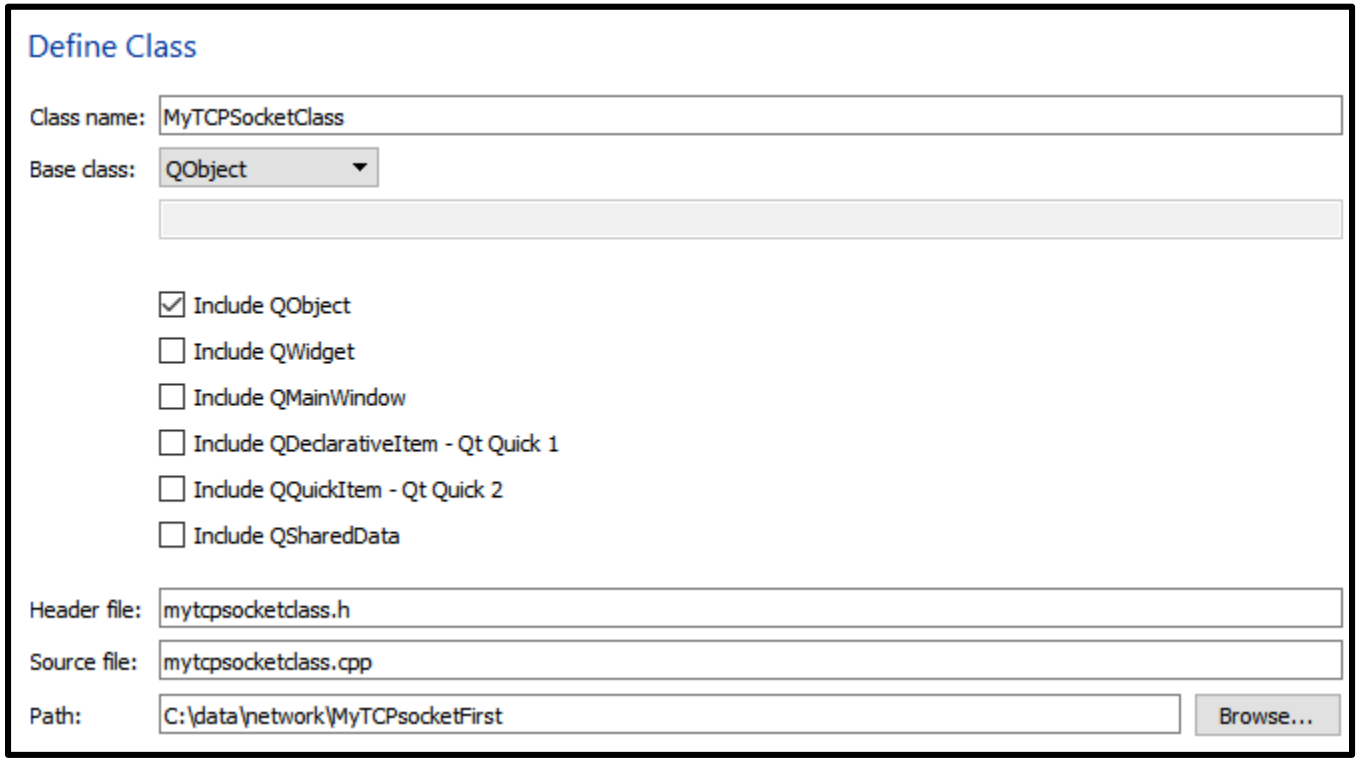
- **QTcpSocket** is a convenience subclass of **QAbstractSocket** that allows you to establish a TCP connection and transfer streams of data. See the **QAbstractSocket** documentation for details.

- Open link http://doc.qt.io/qt-5/index.html and type **QTcpSocket** to Search field

- Go through basic information about **QTcpSocket** Class (more link, header, qmake, inherits, …)

TCP Socket Example 1

- We'll start with **Application->Qt Console Application**. Create project named **MyTCPSocketFirst**.
- When project is done, add network module to project file **MyTCPSocketFirst.pro**: QT += network.
- Run **qmake (Build->Run qmake)** and Build project! You can close project file.
- Add C++ Class **MyTCPSocketClass** to project as shown below (you can choose your own Path)



More about qmake
http://doc.qt.io/archives/qt-4.8/qmake-manual.html#qmake

- Open **mytcpsocketclass.h** file, add lines below which are red to file

```
#ifndef MYTCPSOCKETCLASS_H
#define MYTCPSOCKETCLASS_H

#include <QObject>
#include <QTcpSocket>
#include <QDebug>

class MyTcpSocketClass : public QObject
{
   Q_OBJECT
public:
    MyTcpSocketClass(QObject *parent = nullptr);
   ~MyTCPSocketClass();
   void connectToServer();

private:
   QTcpSocket *socket; // composition

};
#endif // MYTCPSOCKETCLASS_H
```

TCP Socket Example 1 (continues…)

- Open **mytcpsocketclass.cpp** file
- Add lines below to constructor function

```
socket = new QTcpSocket(); // composition object creation
qDebug() << "1: Socket Created" << endl;
```

- Create implementation of destructor function and add lines below in it

```
delete socket; // composition object deleted
socket = nullptr;
qDebug() << "11: Socket Deleted";
```

- Create skeleton implementation of function **void connectToServer()**
- Build the project

```
void MyTCPSocketClass::connectToServer()
{
   qDebug() << "2: Connecting To Server";
   socket->connectToHost("oamk.fi", 80);
   if(socket->waitForConnected(5000))
   {
      qDebug() << "3: Client Connected To Server!" << endl;
      qDebug() << "4: Write message To Server!";
      socket->write("Hello server\r\n");
      if (socket->waitForBytesWritten(1000))
      {
         qDebug() << "5: Wait For Bytes Written";
      }
      qDebug() << "6: Bytes Written" << endl;
      if(socket->waitForReadyRead(3000))
      {
         qDebug() << "7: Wait For Ready To Read ";
      }
      qDebug() << "8: Ready To Read";
      qDebug() << "9: Reading: " << socket->bytesAvailable();
      qDebug() << socket->readAll() << endl;
      socket->close();
      qDebug() << "10: Connection closed" << endl;
   }
   else
   {
      qDebug() << "Not connected!";
      qDebug() << "Error: " << socket->errorString();
   }
}
```

TCP Socket Example 1 (continues…)

- Implement function **connectToServer()** as on the left
- We're taking blocking approach (synchronous). In other words, we use **waitFor...()** functions (e.g., **QTcpSocket::waitForConnected()**) until the operation has completed, instead of connecting to signals.
- In Example 2 we will use the signals/slots functionality
- In next page all functions on the left are explained

TCP Socket Example 1 (continues…)

- **socket->*connectToHost*(”oamk.fi", 80);**
  http://doc.qt.io/qt-5/qabstractsocket.html#connectToHost
- **socket->waitForConnected(5000)**
  http://doc.qt.io/qt-5/qabstractsocket.html#waitForConnected
- **socket->write("Hello server\r\n\r\n");**
  http://doc.qt.io/qt-5/qiodevice.html#write
- **socket->waitForBytesWritten(1000);**
  http://doc.qt.io/qt-5/qabstractsocket.html#waitForBytesWritten
- **socket->waitForReadyRead(3000);**
  http://doc.qt.io/qt-5/qabstractsocket.html#waitForReadyRead
- **socket->bytesAvailable();**
  http://doc.qt.io/qt-5/qabstractsocket.html#bytesAvailable
- **socket->readAll();**
  http://doc.qt.io/qt-5/qiodevice.html#readAll

qint64 QIODevice::write(const char *data, qint64 maxSize)

Writes at most maxSize bytes of data from data to the device. Returns the number of bytes that were actually written, or -1 if an error occurred.

QByteArray QIODevice::readAll()

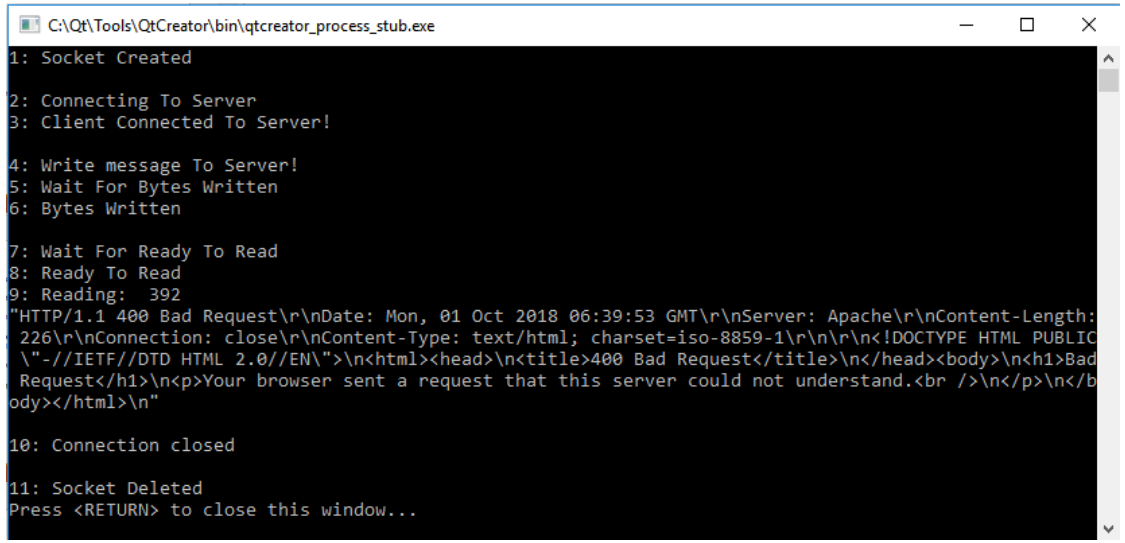
- Reads all remaining data from the device, and returns it as a byte array. This function has no way of reporting errors; returning an empty QByteArray can mean either that no data was currently available for reading, or that an error occurred.

TCP Socket Example 1 (continues…)

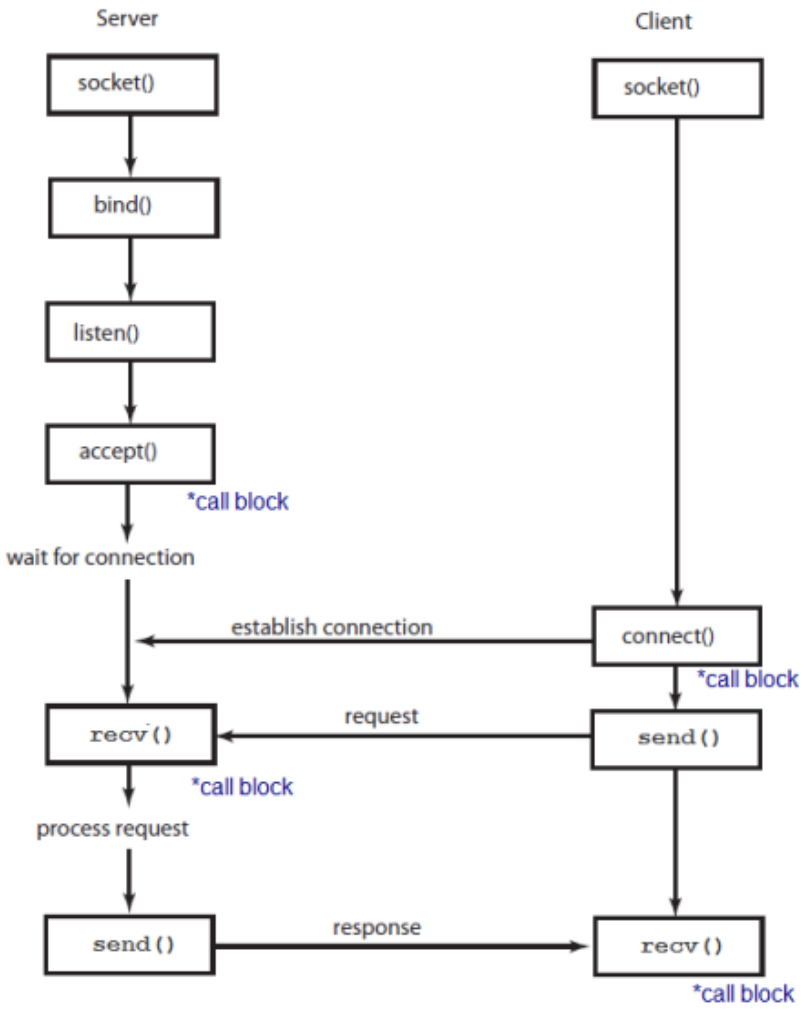
- Open file **main.cpp**. Add lines which are red to file.

```
#include <QCoreApplication>
#include "mytcpsocketclass.h"

int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);

   MyTCPSocketClass *objectMyTCPSocketClass;

   objectMyTCPSocketClass = new MyTCPSocketClass;

   objectMyTCPSocketClass->connectToServer();

   delete objectMyTCPSocketClass;
   objectMyTCPSocketClass= nullptr;

   return 1; // we don't use exec() function, because we don't need it in this example!
}
```

- Build and run the project! Output should be as picture on the right.



TCP Socket Example 1 (continues…)




```
socket = new QTcpSocket();
```

```
socket->connectToHost("oamk.fi",80);
```

```
socket->write("Hello server\r\n\r\n");
```

```
socket->readAll();
```