Interfacing Oxford Temperature sensor using Qt

Akshay Kumar

----------.pro file code----------

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-12-30T03:15:31
#
#-------------------------------------------------

QT       += core gui



greaterThan(QT_MAJOR_VERSION, 4) {
    QT       += widgets serialport
} else {

include($$QTSERIALPORT_PROJECT_ROOT/src/serialport/qt4support/serialpor
t.prf)
}


greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = OXFORD
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

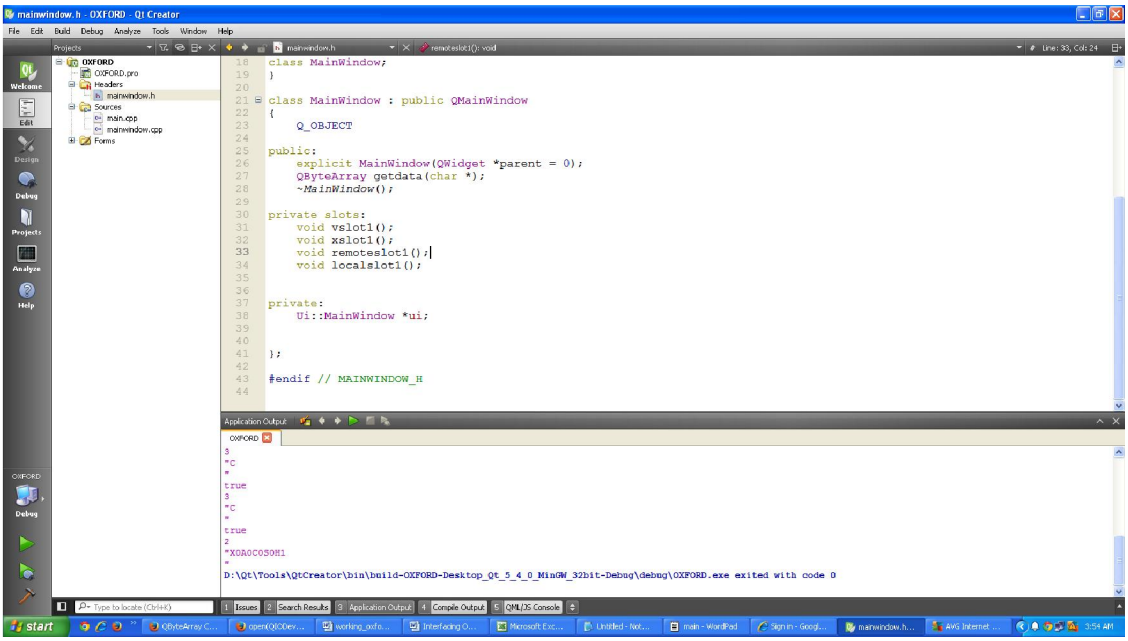
First you need to add serial port to your project widgets. For that, do the above command which adds the serial port according to the version of Qt you have.

Everything else in the .pro file is by default.

.pro file is basically used to establish connections, specify your targets and sources for your project.

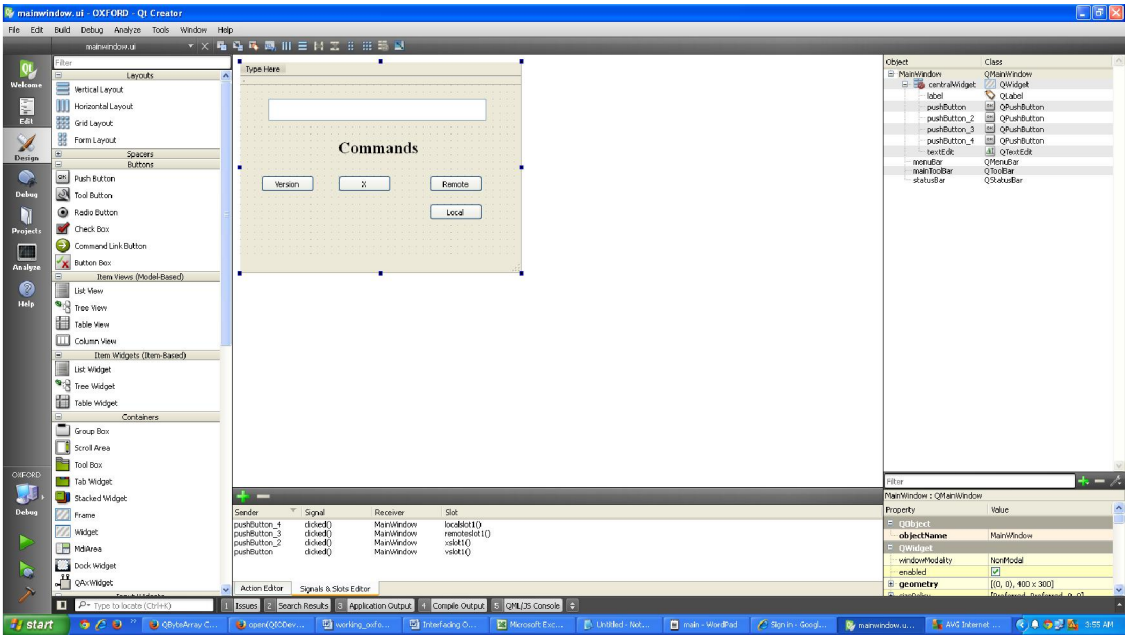
In the .h file add the <QtSerialPort/QSerialPort> library for accessing functions and classes related to serial port.

Whatever function you create yourself, to be defined in MainWindow class, specify their protoypes either in private or public section.

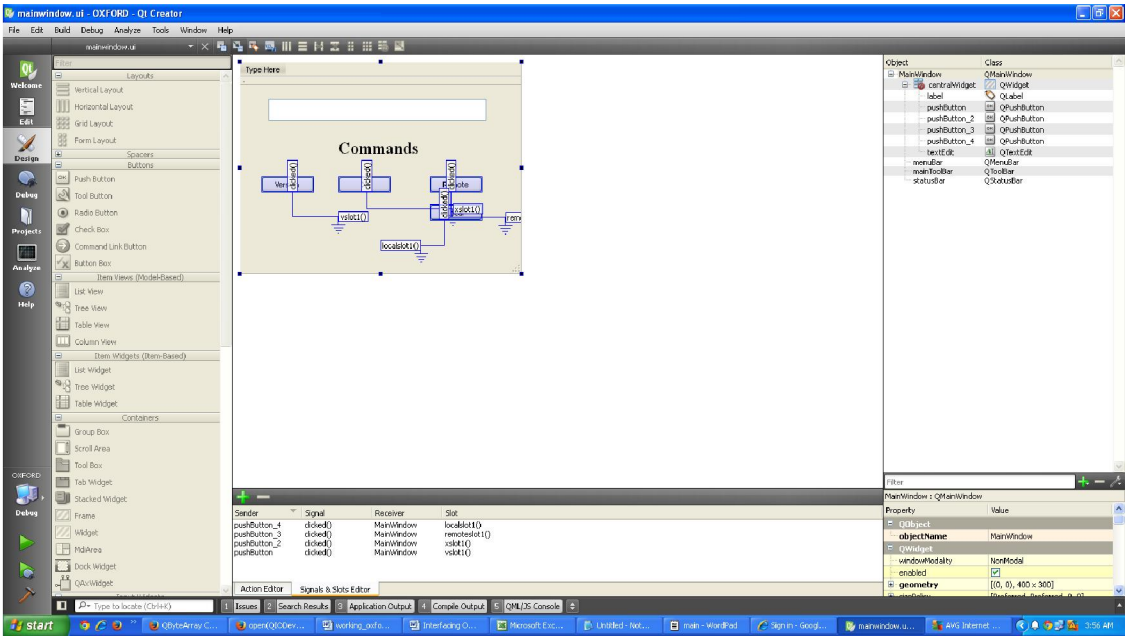


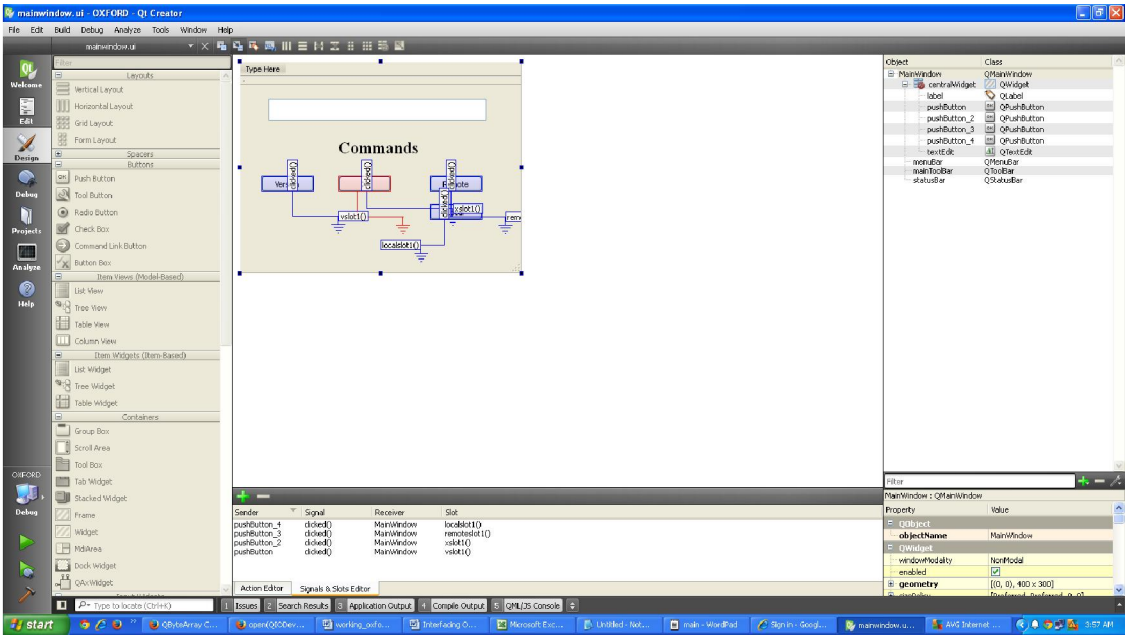
All this is done in the edit tab.

For creating the user interface, click on the form subsection in edit tab. Create a UI with different widgets you want to add as shown:

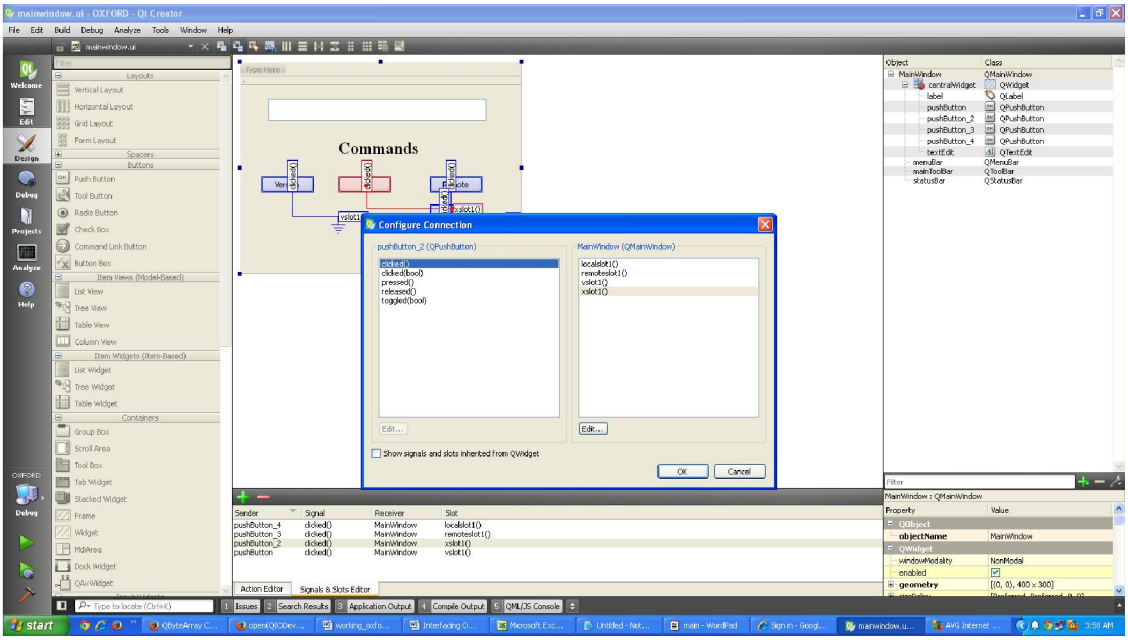


If you want to trigger an action from a widget, like on clicking a button, add slots to each button.

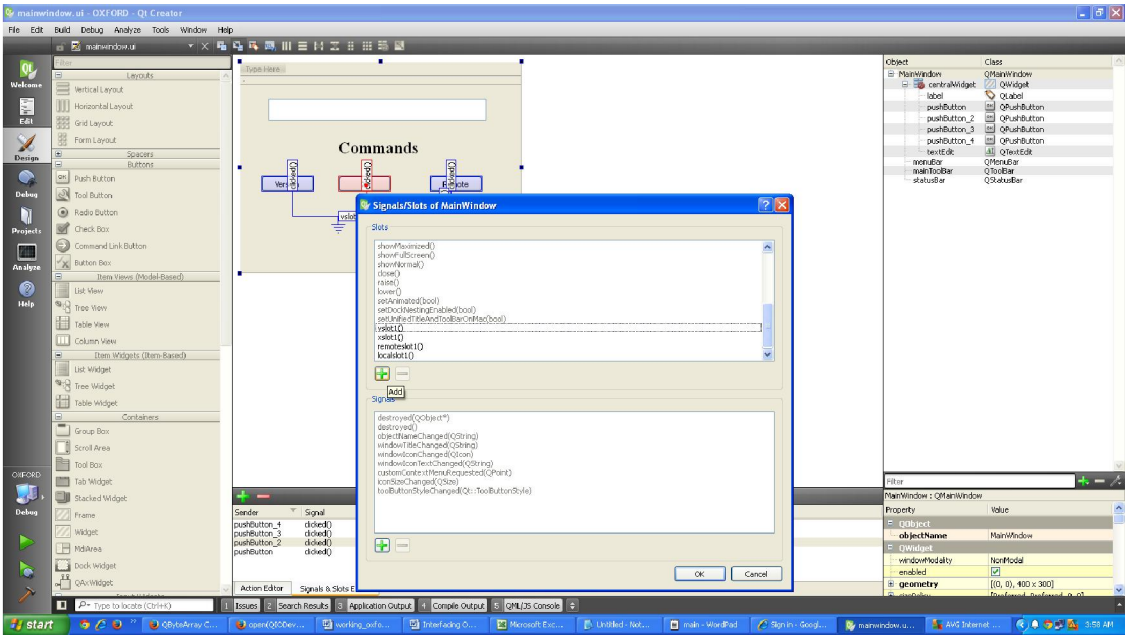




Slots are created by clicking on above edit slots/signals button.

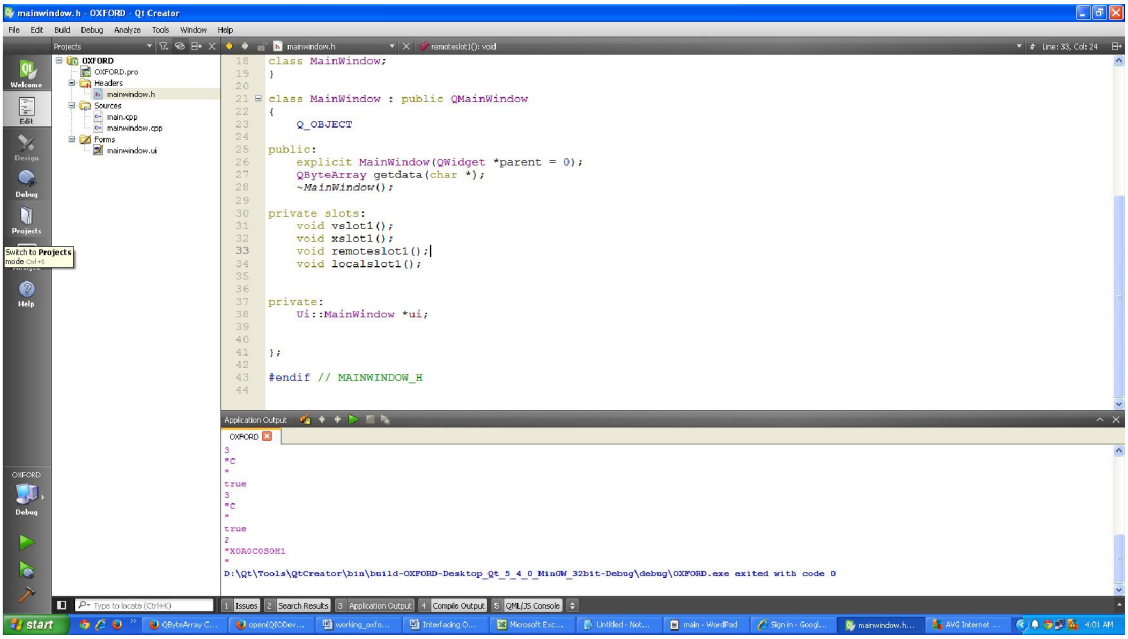


Click on the edit button.



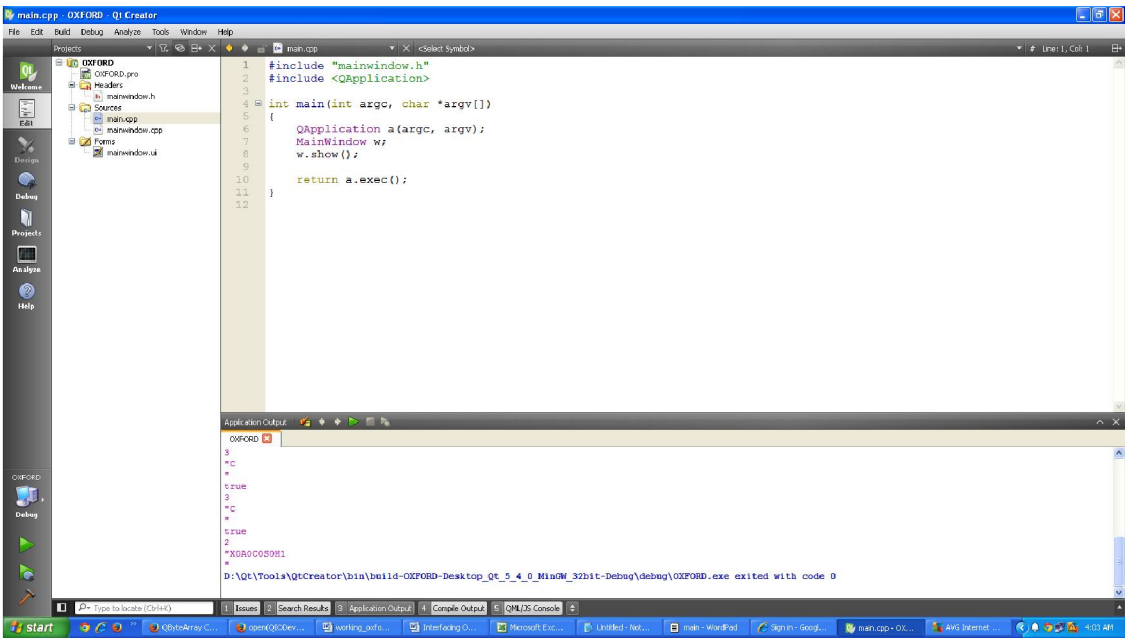
Click on the plus (add) button. Give a specific required callback function name. Now after naming the function, you need to link it to the button action. For that press OK.
Then double click on clicked(), select the function name from MainWindow section and press OK.

Once all such slots (call back functions are named and linked), return to mainwindow.h file.



Now, add all the callback functions created above as private slots: inside the mainwindow class definition. This is the prototype, needed because we will give the definition of the function later after the class, externally. You will see that.

main.cpp



You don’t have to do anything in the main.cpp

Inside the mainwindow.cpp the following code is added :

In the below code, define the definition of callback functions as:

```
void MainWindow::remoteslot1()
{

  replydata=getdata(remotecmd);
  if (replydata[0] == '?')
  {

      ui->textEdit->setPlainText("error");
  }
  else
  {

      ui->textEdit->setPlainText(replydata);
  }

}
```

Where remoteslot1() is the name of callbaback function. Give this function definition outside everything, independent of anything.

In our project, we have created a common QByteArray getdata(char *) function called inside every callback function. ‘remotecmd’ is is the command given as input, defined as #define remotecmd "C1\r" in mainwindow.h.

Now inside the getdata(char *cmmm) function, create a QSerialPort object and dynamically initialize it.

```
QSerialPort *serial2;
serial2 = new QSerialPort(this);

serial2->setPortName("com11");
bool aa=serial2->open(QIODevice::ReadWrite);
qDebug() << aa;
serial2->setBaudRate(9600);
serial2->setDataBits(QSerialPort::Data8);
serial2->setParity(QSerialPort::NoParity);
serial2->setStopBits(QSerialPort::OneStop);
serial2->setFlowControl(QSerialPort::NoFlowControl);
```

Now set the port parameters and open the port as shown above.
serial2->write(cmm); -> using this command, write the command onto the device.

```
ba2="";
          ba="";
          while (ba2 != "\r")
          {
            ba2=serial2->read(1);
            serial2->waitForReadyRead(1);
            ba=ba+ba2;
          }

          qDebug() << ba;

          serial2->close();
```

return(ba);

ba and ba2 both are QByteArray. Using the above method read the response from the device after sending the command. It is reading one character at a time and appending continuously in ba. This process continues upto it encounters carriage return, ie \r.

waitForReadyRead(1) is used to make it wait 1 ms before it reads next character to make every operation complete improper time.