```
#ifndef QTDRO_H
#define QTDRO_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <ctype.h>
#include <math.h>
#include <sys/types.h>
#include <inttypes.h>
#include <iostream>
#include <cstdlib>

#include <qapplication.h>
#include <qlabel.h>
#include <qpushbutton.h>
#include <qlcdnumber.h>
#include <qtimer.h>
#include <QApplication>
#include "qstring.h"
#include "qlayout.h"
#include "qfont.h"
#include <QTime>
#include <QTimer>
#include <QDebug>
#include <QThread>
#include <QTextStream>
#include <QTextStreamFunction>
#include <QTextStreamManipulator>


#include <QApplication>
#include <QtDebug>
#include <stdlib.h>
#include <unistd.h>
#include <sys/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <math.h>
#include <QtCore>
#include <QThread>
#include <QWidget>
#include <sys/types.h>
#include <QTimer>
#include <QDateTime>
#include <QTextStream>
#include <QFile>
#include <QProcess>
#include <iostream>
#include <sstream>
#include <QDial>
#include <QWidget>
#include <QPushButton>
#include <QDir>
#include <QComboBox>
#include <QStringList>
#include <QByteArray>
#include <QAbstractButton>
#include <QTime>
#include <QDate>
#include <QDateEdit>
#include <QTimeEdit>
#include <QGraphicsView>
#include <QGraphicsSceneHoverEvent>
#include <QGraphicsItem>
#include <QGraphicsScene>
#include <QPolygon>
#include <QPolygonF>
#include <QDoubleValidator>
#include <QIntValidator>
#include <QMessageBox>
#include <QPixmap>
#include <sqlite3.h>
//#include <QtSql/QSql>
#include <QtSql/QSqlDatabase>
#include <QtSql/QSqlDriver>
#include <QtSql/QSqlDriverPlugin>
#include <QtSql/QSqlError>
#include <QtSql/QSqlField>
#include <QtSql/QSqlIndex>
#include <QtSql/QSqlRelationalTableModel>
#include <QtSql/QSqlQuery>
#include <QtSql/QSqlQueryModel>
#include <QtSql/QSqlRecord>
#include <QtSql/QSqlRelation>
#include <QtSql/QSqlTableModel>
#include <QtSql/QtSqlDepends>
```

```
#include <QtSql/QtSqlVersion>
//#include <QtSerialBus>
#include <QtSql/QSqlRelationalDelegate>
#include <QtNetwork/QNetworkAccessManager>
#include <QIODevice>
#include <QMessageBox>
#include <QFont>
#include <QAbstractButton>
#include <QDebug>

#include "include/cmd_msg.hh"
#include "include/stat_msg.hh"
#include "include/emc_nml.hh"
#include <stat_msg.hh>
#include <cmd_msg.hh>
#include <nml.hh>
#include "rcs.hh"               // etime()
#include "emc.hh"               // EMC NML
#include "include/emcglb.h"     // EMC_NMLFILE, TRAJ_MAX_VELOCITY,

//  TOOL_TABLE_FILE
#include "include/emccfg.h"     // DEFAULT_TRAJ_MAX_VELOCITY
#include "inifile.hh"           // INIFILE

/*
extern  int  argc;
extern char **argv;
*/

class QtDro: public QThread {
  Q_OBJECT
    public:
        explicit QtDro(QObject *parent = 0, bool b = false);
        void run();
        ~QtDro();
        // if Stop = true, the thread will break
        // out of the loop, and will be disposed
        bool Stop;

    public slots:
        void thEme(bool emeCmd);
        void thOn(bool onCmd);
        // int emcCommandSend(RCS_CMD_MSG & cmd);
/*      int emcCommandSend(RCS_CMD_MSG & cmd) {
            // write command
            if (emcCommandBuffer->write(&cmd)) {
                return -1;
            }
            emcCommandSerialNumber = cmd.serial_number;
            return 0;
        }
*/

    signals:
        void thXYZABCDEF (double x, double y, double z, double a,  double b,
                          double c, double u, double v, double w);

    private:
        int acTh;
        char avTh;
        char state_msg[4096];

    protected:
};


#endif // QTDRO_H

********************************************************************************
#include "QtDro.h"

using namespace std;

// the NML channels to the EMC task
RCS_CMD_CHANNEL *emcCommandBuffer ;
RCS_STAT_CHANNEL *emcStatusBuffer ;
EMC_STAT *emcStatus;
EMC_CMD_MSG *emcCmd;

// the NML channel for errors
static NML *emcErrorBuffer = 0;
static char error_string[NML_INTERNAL_CMS_ERROR] = "";

// the current command numbers, set up updateStatus(), used in main()
static int emcCommandSerialNumber = 0;

int counter = 0;


QtDro::QtDro (QObject *parent, bool b) : QThread(parent), Stop(b) {
```

```
}

QtDro::~QtDro() {
  //  inifile.Close();
  this->Stop;
}


void QtDro::run() {
    double thX = 0;
    double thY = 0;
    double thZ = 0;
    double thA = 0;
    double thB = 0;
    double thC = 0;
    double thU = 0;
    double thV = 0;
    double thW = 0;

    QTextStream(stdout) <<  "First run  function" << endl;

    const char *nmlfile = "/home/bkt/linuxcnc-dev/configs/sim/axis/emc.nml";
    RCS_STAT_CHANNEL *stat = new RCS_STAT_CHANNEL(emcFormat, "emcStatus", "xemc", nmlfile);
    RCS_CMD_CHANNEL *cmd = new RCS_CMD_CHANNEL(emcFormat, "emcCommand", "xemc", nmlfile);
    NML *error = new NML(emcFormat, "emcError", "xemc", nmlfile);

    //************ try to connect to EMC cmd
/*
   if (emcCommandBuffer == 0) {
        emcCommandBuffer =  new RCS_CMD_CHANNEL(emcFormat, "emcCommand", "xemc", nmlfile, 0);
        QTextStream(stdout) <<  "cmd buffer is valid  " << endl;
        if (!emcCommandBuffer->valid()) {
            delete emcCommandBuffer;
            emcCommandBuffer = 0;
            QTextStream(stdout) <<  "cmd buffer is NOT valid  " << endl;
            return;
        }
    }
*/
    while(1) {
        if((stat->peek() != EMC_STAT_TYPE) || (!stat->valid())) {
            QTextStream(stdout) <<  "while    these program NOT run  " << endl;
            break;
        }
        else {
            EMC_STAT *emcStatus = static_cast<EMC_STAT*>(stat->get_address());
            //EMC_CMD_MSG *emcCmd = static_cast<EMC_CMD_MSG*>(cmd->get_address());
            if ( (thX != emcStatus->motion.traj.position.tran.x) ||
                 (thY != emcStatus->motion.traj.position.tran.y) ||
                 (thZ != emcStatus->motion.traj.position.tran.z) ) {
                thX = emcStatus->motion.traj.position.tran.x;
                thY = emcStatus->motion.traj.position.tran.y;
                thZ = emcStatus->motion.traj.position.tran.z;
                thA = 0;
                thB = 0;
                thC = 0;
                thU = 0;
                thV = 0;
                thW = 0;
                emit thXYZABCDEF(thX, thY, thZ, thA, thB, thC, thU, thV, thW);
            }

            if(counter>=80 && counter<81) {
                 QTextStream(stdout) <<  "state_msg try zone" << endl;
                 EMC_TASK_SET_STATE  state_msg;
                // EMC_TASK_PLAN_PAUSE emc_task_plan_pause_msg;
                 QTextStream(stdout) <<  "spy 1" << endl;
                 state_msg.state = EMC_TASK_STATE_OFF;
                 QTextStream(stdout) <<  "spy 2" << endl;
                 state_msg.serial_number = ++emcCommandSerialNumber;
                 QTextStream(stdout) <<  "actual value of state_msg: " <<  state_msg.state << endl;
                 QTextStream(stdout) <<  "these steps seems right .. now try to write" << endl;
                 // cmd->write(&emc_task_plan_pause_msg);
                 cmd->write(&state_msg);
                 /* these NOT work because not
                    declare 'emcCommandBuffer' with automatic memory allocation with macro
                    "new"
                 */
                 QTextStream(stdout) <<  "just write on emcCommandBuffer" << endl;
                 counter = 0;
            }
            counter++;
        }
        this->msleep(100);
    }
}
```