#include <QtGui/QApplication>

#include <QtCore/QTextCodec>

#include <QtCore/QTranslator>

#include <QtCore/QDebug>

#include <QtGui/QSplashScreen>

#include <QtTest/QTest>

#include <QtCore/QThread>

#include "mainform.h"

#include "logger.h"

#include "logindialog.h"

#include "qtexdb.h"

MainForm \*g\_mainform;

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

{

QApplication app(argc, argv);

QFile qss(":/qtverify/qss/default.qss");

qss.open(QFile::ReadOnly);

app.setStyleSheet(qss.readAll());

qss.close();

QSplashScreen \*splash = new QSplashScreen;

splash->setPixmap(QPixmap(":/qtverify/images/facility.png"));

splash->show();

QTextCodec::setCodecForTr(QTextCodec::codecForName("GB2312"));

QTextCodec::setCodecForLocale(QTextCodec::codecForName("GB2312"));

QTextCodec::setCodecForCStrings( QTextCodec::codecForName("GB2312"));

qInstallMsgHandler(myMessageOutput);

Qt::Alignment align = Qt::AlignCenter | Qt::AlignBottom;

splash->showMessage(QObject::tr("load translator files ..."), align, Qt::blue);

QTest::qSleep(200);

QString lang = "zh"; //默认显示中文

if (argc == 2)

{

lang = QString::fromLocal8Bit(argv[1]);

}

QString runhome = QProcessEnvironment::systemEnvironment().value("RUNHOME");

QString file\_name = runhome + "\\ini\\tr\_qtverify.ini";

QFile file(file\_name );

if( file.open(QIODevice::ReadOnly | QIODevice::Text) )

{

QTranslator \*translator = NULL;

QTextStream text(&file);

QString line ;

while ( !text.atEnd() )

{

line = text.readLine().simplified();

if( line.length() == 0 )

continue;

if( line.at(0) == '#' )

continue;

QString i18nName = QProcessEnvironment::systemEnvironment().value("RUNHOME") + "\\uif\\i18n\\" + lang + "\\";

i18nName.append(line).append(QString("\_%1.qm").arg(lang));

translator = new QTranslator( 0 );

if (!translator->load( i18nName ))

{

qDebug()<<"load translator file"<<line<<"failed!";

}

app.installTranslator( translator );

}

file.close();

}

else

{

qDebug("no i18n ini file.\n");

}

qDebug()<<"qtverify main thread:"<<QThread::currentThreadId();

splash->showMessage(QObject::tr("connect database ..."), align, Qt::blue);

QTest::qSleep(200);

startdb(); //连接数据库

splash->showMessage(QObject::tr("setting up the mainwindow ..."), align, Qt::blue);

QTest::qSleep(200);

g\_mainform = new MainForm;

LoginDialog login;

if (login.exec() == QDialog::Accepted)

{

g\_mainform->showMaximized();

splash->finish(g\_mainform);

delete splash;

app.exec();

}

closedb();

return 0;

}

#include <QtGui/QMessageBox>

#include <QAxObject>

#include <QProcess>

#include <QtCore>

#include "mainform.h"

#include "dbmysql.h"

#include "flowresult.h"

#include "setcomfrm.h"

#include "datatestdlg.h"

#include "setportfrm.h"

#include "masterslaveset.h"

#include "algorithm.h"

#include "flowweight.h"

#include "flowstandard.h"

#include "totalweight.h"

#include "totalstandard.h"

#include "calcverify.h"

#include "stdparaset.h"

#include "stdcoecorrect.h"

MainForm::MainForm(QWidget \*parent, Qt::WFlags flags)

: QMainWindow(parent, flags)

{

qDebug()<<"MainForm thread:"<<QThread::currentThreadId();

ui.setupUi(this);

m\_mySql = NULL;

m\_flowResultDlg = NULL;

m\_alg = new CAlgorithm();

m\_setcom = NULL;

m\_datatestdlg = NULL;

m\_portSet = NULL;

m\_masterslave = NULL;

m\_comProcess = new QProcess(this);

m\_flowWeightDlg = NULL;

m\_flowStandardDlg = NULL;

m\_totalWeightDlg = NULL;

m\_totalStandardDlg = NULL;

m\_calcDlg = NULL;

m\_stdParaSet = NULL;

m\_stdCoeCorrect = NULL;

QLabel \*permanent = new QLabel(this);

permanent->setFrameStyle(QFrame::NoFrame | QFrame::Sunken);

permanent->setText(tr("<a href=\"http://www.sdm.com.cn\">Shandong Delu Measurement Co.,Ltd.</a>"));

permanent->setTextFormat(Qt::RichText);

permanent->setOpenExternalLinks(true);

ui.statusBar->addPermanentWidget(permanent);

}

MainForm::~MainForm()

{

}

void MainForm::closeEvent( QCloseEvent \* event)

{

int button = QMessageBox::question(this, tr("Question"), tr("Quit Really ?"), \

QMessageBox::Yes|QMessageBox::Default, QMessageBox::No|QMessageBox::Escape);

if (button == QMessageBox::No)

{

event->ignore();

}

else if (button == QMessageBox::Yes)

{

event->accept();

qDebug()<<"^^^^^MainForm::closeEvent";

if (m\_mySql)

{

delete m\_mySql;

m\_mySql = NULL;

}

if (m\_flowResultDlg)

{

delete m\_flowResultDlg;

m\_flowResultDlg = NULL;

}

if (m\_alg)

{

delete m\_alg;

m\_alg = NULL;

}

if (m\_setcom)

{

delete m\_setcom;

m\_setcom = NULL;

}

if (m\_portSet)

{

delete m\_portSet;

m\_portSet = NULL;

}

if (m\_datatestdlg)

{

delete m\_datatestdlg;

m\_datatestdlg = NULL;

}

if (m\_masterslave)

{

delete m\_masterslave;

m\_masterslave = NULL;

}

if (m\_comProcess)

{

m\_comProcess->kill();

}

if (m\_flowWeightDlg)

{

delete m\_flowWeightDlg;

m\_flowWeightDlg = NULL;

}

if (m\_flowStandardDlg)

{

delete m\_flowStandardDlg;

m\_flowStandardDlg = NULL;

}

if (m\_totalWeightDlg)

{

delete m\_totalWeightDlg;

m\_totalWeightDlg = NULL;

}

if (m\_totalStandardDlg)

{

delete m\_totalStandardDlg;

m\_totalStandardDlg = NULL;

}

if (m\_calcDlg)

{

delete m\_calcDlg;

m\_calcDlg = NULL;

}

if (m\_stdParaSet)

{

delete m\_stdParaSet;

m\_stdParaSet = NULL;

}

if (m\_stdCoeCorrect)

{

delete m\_stdCoeCorrect;

m\_stdCoeCorrect = NULL;

}

}

}

void MainForm::on\_actionComSet\_triggered()

{

if (NULL == m\_setcom)

{

m\_setcom = new SetComFrm();

}

m\_setcom->show();

}

void MainForm::on\_actionPortSet\_triggered()

{

if (NULL == m\_portSet)

{

m\_portSet = new SetPortFrm();

}

m\_portSet->show();

}

void MainForm::on\_actionDataTest\_triggered()

{

if (NULL == m\_datatestdlg)

{

m\_datatestdlg = new DataTestDlg();

}

else //目的是执行QualityDlg的构造函数

{

delete m\_datatestdlg;

m\_datatestdlg = NULL;

m\_datatestdlg = new DataTestDlg();

}

m\_datatestdlg->show();

}

void MainForm::on\_actionComDebuger\_triggered()

{

QStringList cmdlist;

cmdlist<<"-zh";

m\_comProcess->start("qcom", cmdlist);

}

void MainForm::on\_actionStdParaSet\_triggered()

{

if (NULL == m\_stdParaSet)

{

m\_stdParaSet = new StdParaSet();

}

else //目的是执行StdParaSet的构造函数

{

delete m\_stdParaSet;

m\_stdParaSet = NULL;

m\_stdParaSet = new StdParaSet();

}

m\_stdParaSet->show();

}

void MainForm::on\_actionStdCoeCorrect\_triggered()

{

if (NULL == m\_stdCoeCorrect)

{

m\_stdCoeCorrect = new StdCoeCorrect();

}

else //目的是执行StdCoeCorrect的构造函数

{

delete m\_stdCoeCorrect;

m\_stdCoeCorrect = NULL;

m\_stdCoeCorrect = new StdCoeCorrect();

}

m\_stdCoeCorrect->show();

}

void MainForm::on\_actionMasterSlaveSet\_triggered()

{

if (NULL == m\_masterslave)

{

m\_masterslave = new CMasterSlave();

}

m\_masterslave->show();

}

void MainForm::on\_actionMySql\_triggered()

{

if (NULL == m\_mySql)

{

m\_mySql = new DbMySql();

}

m\_mySql->show();

}

void MainForm::on\_actionFlowWeight\_triggered()

{

if (NULL == m\_flowWeightDlg)

{

m\_flowWeightDlg = new FlowWeightDlg();

}

else //目的是执行FlowWeightDlg的构造函数

{

delete m\_flowWeightDlg;

m\_flowWeightDlg = NULL;

m\_flowWeightDlg = new FlowWeightDlg();

}

m\_flowWeightDlg->show();

}

void MainForm::on\_actionFlowStandard\_triggered()

{

if (NULL == m\_flowStandardDlg)

{

m\_flowStandardDlg = new FlowStandardDlg();

}

else //目的是执行FlowStandardDlg的构造函数

{

delete m\_flowStandardDlg;

m\_flowStandardDlg = NULL;

m\_flowStandardDlg = new FlowStandardDlg();

}

m\_flowStandardDlg->show();

}

void MainForm::on\_actionCalculator\_triggered()

{

if (NULL == m\_calcDlg)

{

m\_calcDlg = new CalcDlg();

}

else //目的是执行CalcDlg的构造函数

{

delete m\_calcDlg;

m\_calcDlg = NULL;

m\_calcDlg = new CalcDlg();

}

m\_calcDlg->show();

}

void MainForm::on\_actionTotalWeight\_triggered()

{

if (NULL == m\_totalWeightDlg)

{

m\_totalWeightDlg = new TotalWeightDlg();

}

else //目的是执行TotalWeightDlg的构造函数

{

delete m\_totalWeightDlg;

m\_totalWeightDlg = NULL;

m\_totalWeightDlg = new TotalWeightDlg();

}

m\_totalWeightDlg->show();

}

void MainForm::on\_actionTotalStandard\_triggered()

{

if (NULL == m\_totalStandardDlg)

{

m\_totalStandardDlg = new TotalStandardDlg();

}

else //目的是执行TotalStandardDlg的构造函数

{

delete m\_totalStandardDlg;

m\_totalStandardDlg = NULL;

m\_totalStandardDlg = new TotalStandardDlg();

}

m\_totalStandardDlg->show();

}

void MainForm::on\_actionFlowResult\_triggered()

{

if (NULL == m\_flowResultDlg)

{

m\_flowResultDlg = new FlowResultDlg();

}

else

{

delete m\_flowResultDlg;

m\_flowResultDlg = new FlowResultDlg();

}

m\_flowResultDlg->show();

}

void MainForm::on\_actionQueryExcel\_triggered()

{

QAxObject \*excel = NULL;

excel = new QAxObject("Excel.Application");

if (!excel)

{

QMessageBox::critical(this, tr("Error"), tr("Excel object lose!"));

return;

}

QAxObject \*workbooks = NULL;

QAxObject \*workbook = NULL;

excel->dynamicCall("SetVisible(bool)", false);

workbooks = excel->querySubObject("WorkBooks");

QString xlsFile = QProcessEnvironment::systemEnvironment().value("RUNHOME") + "\\dat\\test.xlsx";

workbook = workbooks->querySubObject("Open(QString, QVariant)", xlsFile);

if (NULL==workbook)

{

return;

}

QAxObject \* worksheet = workbook->querySubObject("WorkSheets(int)", 1);

QAxObject \* usedrange = worksheet->querySubObject("UsedRange");

QAxObject \* rows = usedrange->querySubObject("Rows");

QAxObject \* columns = usedrange->querySubObject("Columns");

int intRowStart = usedrange->property("Row").toInt();

int intColStart = usedrange->property("Column").toInt();

int intCols = columns->property("Count").toInt();

int intRows = rows->property("Count").toInt();

for (int i = intRowStart; i < intRowStart + intRows; i++) //行

{

for (int j = intColStart; j < intColStart + intCols; j++) //列

{

QAxObject \* range = worksheet->querySubObject("Cells(int,int)", i, j ); //获取单元格

qDebug("row %d, col %d, value is %d\n", i, j, range->property("Value").toInt());

}

}

}

void MainForm::on\_actionExit\_triggered()

{

this->close();

}

void MainForm::on\_actionAbout\_triggered()

{

QMessageBox::aboutQt(this);

}

void MainForm::on\_actionDefault\_triggered()

{

QFile qss(":/qtverify/qss/default.qss");

qss.open(QFile::ReadOnly);

this->setStyleSheet(qss.readAll());

qss.close();

}

void MainForm::on\_actionClassic\_triggered()

{

QFile qss(":/qtverify/qss/classic.qss");

qss.open(QFile::ReadOnly);

this->setStyleSheet(qss.readAll());

qss.close();

}

void MainForm::on\_actionFashion\_triggered()

{

QFile qss(":/qtverify/qss/fashion.qss");

qss.open(QFile::ReadOnly);

this->setStyleSheet(qss.readAll());

qss.close();

}

void MainForm::chaneLanguage(QString lang)

{

QString runhome = QProcessEnvironment::systemEnvironment().value("RUNHOME");

QString file\_name = runhome + "\\ini\\tr\_qtverify.ini";

QFile file(file\_name );

if( !file.open(QIODevice::ReadOnly | QIODevice::Text) )

{

qDebug("no i18n ini file.\n");

return;

}

QTranslator \*translator = NULL;

QTextStream text(&file);

QString line ;

while ( !text.atEnd() )

{

line = text.readLine().simplified();

if( line.length() == 0 )

continue;

if( line.at(0) == '#' )

continue;

QString i18nName = QProcessEnvironment::systemEnvironment().value("RUNHOME") + "\\uif\\i18n\\" + lang + "\\";

line = line + "\_" + lang + ".qm";

i18nName.append(line);//.append(QString("\_%1.qm").arg(lang));

translator = new QTranslator( 0 );

if (!translator->load( i18nName ))

{

qDebug()<<"load translator file"<<line<<"failed!";

continue;

}

qApp->installTranslator( translator );

ui.retranslateUi(this);

}

file.close();

delete []translator;

}

void MainForm::on\_actionEnglish\_triggered()

{

chaneLanguage("en");

}

void MainForm::on\_actionChinese\_triggered()

{

chaneLanguage("zh");

}

#include <QtCore/QDebug>

#include <QtCore/QDateTime>

#include <math.h>

#include "protocol.h"

CProtocol::CProtocol()

{

}

CProtocol::~CProtocol()

{

}

TempProtocol::TempProtocol()

{

m\_tempFrame = new Temp\_Frame\_Struct();

memset(m\_tempFrame, 0, sizeof(Temp\_Frame\_Struct));

m\_tempStr = "";

m\_sendBuf = "";

}

TempProtocol::~TempProtocol()

{

if (m\_tempFrame != NULL)

{

delete m\_tempFrame;

m\_tempFrame = NULL;

}

}

void TempProtocol::makeSendBuf()

{

m\_sendBuf = "";

m\_sendBuf.append(ADDR\_CODE\_FIRST).append(ADDR\_CODE\_FIRST);//地址代号

m\_sendBuf.append(READ\_CODE); //标准读代码

UINT8 paracode = 0x00; //参数代号

m\_sendBuf.append(paracode);

UINT8 datacode = 0x00;

m\_sendBuf.append(datacode).append(datacode);

UINT16 checkcode = paracode\*256 + READ\_CODE + ADDR\_FIRST;

QString checkstr;

checkstr = QString("%1").arg(checkcode, 4, 16).replace(" ", "0");

bool ok;

UINT8 lownum = checkstr.right(2).toUInt(&ok, 16);

UINT8 hightnum = checkstr.left(2).toUInt(&ok, 16);

m\_sendBuf.append(lownum).append(hightnum);

}

bool TempProtocol::readTemperComBuffer(QByteArray tmp)

{

memset(m\_tempFrame, 0, sizeof(Temp\_Frame\_Struct));

bool ret = false;

int state = PV\_STATE;

UINT8 uch = 0; //无符号8位数字

INT8 ch = 0; //有符号8位数字

int number = tmp.size();

int m=0;

int pv\_i=0, sv\_i=0, para\_i=0, check\_i=0;

INT16 ck=0; //程序计算的检验码

for (m=0; m<number; m++)

{

switch(state)

{

case PV\_STATE: //16位有符号

{

if (pv\_i == 0) //低字节

{

uch = (UINT8)tmp.at(m);

m\_tempFrame->pv = uch;

pv\_i++;

break;

}

if (pv\_i == 1) //高字节

{

ch = (INT8)tmp.at(m);

state = SV\_STATE;

m\_tempFrame->pv = ch\*256 + m\_tempFrame->pv;

pv\_i = 0;

}

break;

}

case SV\_STATE: //16位有符号

{

if (sv\_i == 0) //低字节

{

uch = (UINT8)tmp.at(m);

m\_tempFrame->sv = uch;

sv\_i++;

break;

}

if (sv\_i == 1) //高字节

{

ch = (INT8)tmp.at(m);

state = MV\_STATE;

m\_tempFrame->sv = ch\*256 + m\_tempFrame->sv;

sv\_i = 0;

}

break;

}

case MV\_STATE: //8位有符号

{

ch = (INT8)tmp.at(m);

m\_tempFrame->mv = ch;

state = WARN\_STATE;

break;

}

case WARN\_STATE: //8位无符号

{

ch = (INT8)tmp.at(m);

m\_tempFrame->warn = ch;

state = PARA\_STATE;

break;

}

case PARA\_STATE: //16位有符号

{

if (para\_i == 0) //低字节 无符号

{

uch = (UINT8)tmp.at(m);

m\_tempFrame->para = uch;

para\_i++;

break;

}

if (para\_i == 1) //高字节 有符号

{

ch = (INT8)tmp.at(m);

state = CHECK\_STATE;

m\_tempFrame->para = ch\*256 + m\_tempFrame->para;

para\_i = 0;

}

break;

}

case CHECK\_STATE: //16位

{

if (check\_i == 0) //低字节

{

uch = (UINT8)tmp.at(m);

m\_tempFrame->cs = uch;

check\_i++;

break;

}

if (check\_i == 1) //高字节

{

ch = (INT8)tmp.at(m);

m\_tempFrame->cs = ch\*256 + m\_tempFrame->cs;

check\_i = 0;

state = PV\_STATE;

ck = CountCheck(m\_tempFrame);

if (ck == m\_tempFrame->cs) //校验通过

{

analyseFrame();

ret = true;

}

}

break;

}

default :

{

state = PV\_STATE;

break;

}

} //END OF switch(state)

} //END OF for (m=0; m<number; m++)

return ret;

}

INT16 TempProtocol::CountCheck(Temp\_Frame\_Struct \*pFrame)

{

if (NULL == pFrame)

{

return 0;

}

INT16 cs = 0;

cs = pFrame->pv + pFrame->sv + (pFrame->warn\*256 + pFrame->mv) + pFrame->para + ADDR\_FIRST;

return cs;

}

void TempProtocol::analyseFrame()

{

float PV = ((float)m\_tempFrame->pv)/10;

float SV = ((float)m\_tempFrame->sv)/10;

QString pvStr = QString("%1").arg(PV, DATA\_WIDTH, 'f', DATA\_PRECISION);

QString svStr = QString("%1").arg(SV, DATA\_WIDTH, 'f', DATA\_PRECISION);

m\_tempStr = pvStr + svStr;

}

QString TempProtocol::getTempStr()

{

qDebug()<<"TempProtocol::getTempStr thread:"<<QThread::currentThreadId();

return m\_tempStr;

}

QByteArray TempProtocol::getSendBuf()

{

return m\_sendBuf;

}

BalanceProtocol::BalanceProtocol()

{

m\_balValue = 0.0;

}

BalanceProtocol::~BalanceProtocol()

{

}

bool BalanceProtocol::readBalanceComBuffer(QByteArray tmp)

{

QByteArray whtArray;

m\_balValue = 0.0;

bool ret = false;

int num = tmp.size();

if (num < BAL\_DATA\_LENGTH) //一帧通常是22字节；

{

return ret;

}

int m=0;

char ch;

UINT8 ch1, ch2;

bool ok;

for (int i=num; i>0; i--)

{

ch1 = (UINT8)tmp.at(i-1);

if (ch1==ASCII\_LF && i>=16) //0x0A换行（表示一帧结束）

{

for (m=i-16; m<i-6; m++)

{

ch = tmp.at(m);

whtArray.append(ch);

}

m\_balValue = whtArray.replace(" ", 0).toFloat(&ok);

if (ok)

{

ret = true;

break;

}

}

}

return ret;

}

float BalanceProtocol::getBalanceValue()

{

return m\_balValue;

}

CtrlProtocol::CtrlProtocol()

{

m\_sendBuf = "";

}

CtrlProtocol::~CtrlProtocol()

{

}

QByteArray CtrlProtocol::getSendBuf()

{

return m\_sendBuf;

}

NewCtrlProtocol::NewCtrlProtocol()

{

m\_ctrlFrame = new NewCtrl\_Frame\_Struct();

memset(m\_ctrlFrame, 0, sizeof(NewCtrl\_Frame\_Struct));

m\_balValue = 0.0;

};

NewCtrlProtocol::~NewCtrlProtocol()

{

if (m\_ctrlFrame)

{

delete m\_ctrlFrame;

m\_ctrlFrame = NULL;

}

};

void NewCtrlProtocol::makeFrameOfCtrlRelay(UINT8 portno, bool status)

{

m\_sendBuf = "";

m\_sendBuf.append(CTRL\_START\_CODE).append(CTRL\_FUNC\_RELAY);

UINT8 relay\_num = 0x01; //控制的继电器数量 1路

m\_sendBuf.append(relay\_num);

m\_sendBuf.append(portno); //第protno路继电器

float a = 2;

UINT8 code0 = 0x00;

UINT8 data;

UINT8 st = status ? 0xFF : 0x00;

if (portno>=1 && portno<=8)

{

data = (UINT8)pow(a, (portno-1)) & st;

m\_sendBuf.append(data).append(code0).append(code0);

}

else if (portno>=9 && portno<=16)

{

data = (UINT8)pow(a, (portno-9)) & st;;

m\_sendBuf.append(code0).append(data).append(code0);

}

else if (portno>=17 && portno<=24)

{

data = (UINT8)pow(a, (portno-17)) & st;

m\_sendBuf.append(code0).append(code0).append(data);

}

UINT8 cs = CTRL\_START\_CODE + CTRL\_FUNC\_RELAY + relay\_num + portno + code0 + code0 + data;

m\_sendBuf.append(cs).append(CTRL\_END\_CODE);

}

void NewCtrlProtocol::makeFrameOfCtrlRegulate(UINT8 portno, UINT16 degree)

{

m\_sendBuf = "";

m\_sendBuf.append(CTRL\_START\_CODE).append(CTRL\_FUNC\_REGULATE);

float a = 2;

UINT8 regulate\_num = (UINT8)pow(a, (portno-1)); //控制的调节阀数量 只控制1路

m\_sendBuf.append(regulate\_num);

QString degStr = QString("%1").arg(degree, 4, 16).replace(" ", "0");

bool ok;

UINT8 dataL = degStr.right(2).toUInt(&ok, 16);//开度 低字节 需要实验和计算得到

UINT8 dataH = degStr.left(2).toUInt(&ok, 16); //开度 高字节

m\_sendBuf.append(dataL).append(dataH);

UINT8 cs = CTRL\_START\_CODE + CTRL\_FUNC\_REGULATE + regulate\_num + dataL + dataH;

m\_sendBuf.append(cs).append(CTRL\_END\_CODE);

}

void NewCtrlProtocol::makeFrameOfCtrlQuery()

{

m\_sendBuf = "";

m\_sendBuf.append(CTRL\_START\_CODE).append(CTRL\_FUNC\_QUERY);

UINT8 code0 = 0x00;

m\_sendBuf.append(code0).append(code0).append(code0).append(code0);

UINT8 cs = CTRL\_START\_CODE + CTRL\_FUNC\_QUERY + code0 + code0 + code0 + code0;

m\_sendBuf.append(cs).append(CTRL\_END\_CODE);

}

UINT8 NewCtrlProtocol::readCtrlComBuffer(QByteArray tmp)

{

qDebug()<<"readControlComBuffer ControlProtocol thread:"<<QThread::currentThreadId();

UINT8 ret = 0x00;

int state = CTRL\_START\_STATE;

UINT8 ch = 0; //无符号8位数字

int number = tmp.size();

int m=0;

int num\_i=0;

UINT8 ck=0; //程序计算的检验码

for (m=0; m<number; m++)

{

ch = (UINT8)tmp.at(m);

qDebug()<<"read data is:"<<ch;

switch(state)

{

case CTRL\_START\_STATE: //8位无符号

{

if (ch == CTRL\_START\_CODE)

{

m\_ctrlFrame->startCode = CTRL\_START\_CODE;

state = CTRL\_FUNC\_STATE;

}

break;

}

case CTRL\_FUNC\_STATE: //8位无符号

{

if (ch == CTRL\_FUNC\_RELAY) //功能码-继电器控制

{

m\_ctrlFrame->funcCode = CTRL\_FUNC\_RELAY;

state = CTRL\_DATA\_STATE;

break;

}

if (ch == CTRL\_FUNC\_REGULATE) //功能码-调节阀控制

{

m\_ctrlFrame->funcCode = CTRL\_FUNC\_REGULATE;

state = CTRL\_DATA\_STATE;

break;

}

if (ch == CTRL\_FUNC\_QUERY) //功能码-查询

{

m\_ctrlFrame->funcCode = CTRL\_FUNC\_QUERY;

state = CTRL\_DATA\_STATE;

break;

}

if (ch == CTRL\_FUNC\_BALANCE) //功能码-天平采集

{

m\_ctrlFrame->funcCode = CTRL\_FUNC\_BALANCE;

state = CTRL\_DATA\_STATE;

break;

}

break;

}

case CTRL\_DATA\_STATE: //8位无符号

{

if (m\_ctrlFrame->funcCode == CTRL\_FUNC\_RELAY) //继电器

{

m\_ctrlFrame->data[num\_i++] = ch;

if (num\_i == RELAY\_DATA\_LENGTH)

{

state = CTRL\_CS\_STATE;

num\_i = 0;

}

}

if (m\_ctrlFrame->funcCode == CTRL\_FUNC\_REGULATE) //调节阀

{

m\_ctrlFrame->data[num\_i++] = ch;

if (num\_i == REGU\_DATA\_LENGTH)

{

state = CTRL\_CS\_STATE;

num\_i = 0;

}

}

if (m\_ctrlFrame->funcCode == CTRL\_FUNC\_QUERY) //查询

{

m\_ctrlFrame->data[num\_i++] = ch;

if (num\_i == DATA\_LENGTH)

{

state = CTRL\_CS\_STATE;

num\_i = 0;

}

}

if (m\_ctrlFrame->funcCode == CTRL\_FUNC\_BALANCE) //天平

{

m\_ctrlFrame->data[num\_i++] = ch;

if (num\_i == BAL\_DATA\_LENGTH)

{

state = CTRL\_CS\_STATE;

num\_i = 0;

}

}

break;

}

case CTRL\_CS\_STATE: //8位无符号

{

m\_ctrlFrame->cs = ch;

state = CTRL\_END\_STATE;

break;

}

case CTRL\_END\_STATE: //8位无符号

{

m\_ctrlFrame->endCode = CTRL\_END\_CODE;

state = CTRL\_START\_STATE;

ck = CountCheck(m\_ctrlFrame);

if (ck == m\_ctrlFrame->cs) //校验通过

{

analyseFrame();

qDebug()<<"check is ok 校验通过";

ret = m\_ctrlFrame->funcCode; //以功能码返回，便于区分

}

break;

}

default :

{

state = CTRL\_START\_STATE;

break;

}

} //END OF switch(state)

} //END OF for (m=0; m<number; m++)

return ret;

}

UINT8 NewCtrlProtocol::CountCheck(NewCtrl\_Frame\_Struct \*pFrame)

{

if (NULL == pFrame)

{

return 0;

}

UINT8 cs = pFrame->startCode + pFrame->funcCode;

int i = 0;

if (pFrame->funcCode == CTRL\_FUNC\_RELAY)

{

for (i=0; i<RELAY\_DATA\_LENGTH; i++)

{

cs += pFrame->data[i];

}

}

if (pFrame->funcCode == CTRL\_FUNC\_REGULATE)

{

for (i=0; i<REGU\_DATA\_LENGTH; i++)

{

cs += pFrame->data[i];

}

}

if (pFrame->funcCode == CTRL\_FUNC\_QUERY)

{

for (i=0; i<DATA\_LENGTH; i++)

{

cs += pFrame->data[i];

}

}

if (pFrame->funcCode == CTRL\_FUNC\_BALANCE)

{

for (i=0; i<BAL\_DATA\_LENGTH; i++)

{

cs += pFrame->data[i];

}

}

return cs;

}

void NewCtrlProtocol::analyseFrame()

{

if (NULL==m\_ctrlFrame)

{

return;

}

if (m\_ctrlFrame->funcCode == CTRL\_FUNC\_BALANCE) //天平采集

{

QByteArray whtArray;

m\_balValue = 0.0; //先清零

char ch;

UINT8 ch1, ch2;

ch1 = m\_ctrlFrame->data[BAL\_DATA\_LENGTH-1];

ch2 = m\_ctrlFrame->data[BAL\_DATA\_LENGTH-2];

if (ch1==ASCII\_LF && ch2==ASCII\_CR) //0x0A换行; 0x0D回车（表示一帧结束）

{

for (int i=6; i<16; i++)

{

ch = m\_ctrlFrame->data[i];

whtArray.append(ch);

}

m\_balValue = whtArray.replace(" ", "0").toFloat();

}

}

}

NewCtrl\_Frame\_Struct \* NewCtrlProtocol::getConFrame()

{

return m\_ctrlFrame;

}

float NewCtrlProtocol::getBalanceValue()

{

return m\_balValue;

}

OldCtrlProtocol::OldCtrlProtocol()

{

portCloseMap.insert(1, 0xE1);

portCloseMap.insert(2, 0xE3);

portCloseMap.insert(3, 0xE5);

portCloseMap.insert(4, 0xE7);

portCloseMap.insert(5, 0xE9);

portCloseMap.insert(6, 0xEB);

portCloseMap.insert(7, 0xED);

portCloseMap.insert(8, 0xEF);

portOpenMap.insert(1, 0xE0);

portOpenMap.insert(2, 0xE2);

portOpenMap.insert(3, 0xE4);

portOpenMap.insert(4, 0xE6);

portOpenMap.insert(5, 0xE8);

portOpenMap.insert(6, 0xEA);

portOpenMap.insert(7, 0xEC);

portOpenMap.insert(8, 0xEE);

}

OldCtrlProtocol::~OldCtrlProtocol()

{

}

void OldCtrlProtocol::makeFrameOfCtrlRelay(UINT8 portno, bool status)

{

qDebug()<<"OldCtrlProtocol::makeFrameOfCtrlRelay";

UINT8 zeroCode = 0x00;

m\_sendBuf = "";

if (status) //true(阀门打开,闭合继电器)

{

m\_sendBuf.append(0xFF).append(portCloseMap[portno]).append(zeroCode)\

.append(zeroCode).append(0xFE);

}

else //false(阀门关闭，断开继电器)

{

m\_sendBuf.append(0xFF).append(portOpenMap[portno]).append(zeroCode)\

.append(zeroCode).append(0xFE);

}

}

void OldCtrlProtocol::makeFrameOfCtrlWaterPump(UINT8 portno, bool status)

{

UINT8 zeroCode = 0x00;

m\_sendBuf = "";

if (status) //true 开水泵

{

m\_sendBuf.append(0xFF).append(0xF9).append(zeroCode).append(zeroCode).append(0xFE);

}

else //false 关水泵

{

m\_sendBuf.append(0xFF).append(0xFA).append(zeroCode).append(zeroCode).append(0xFE);

}

}

void OldCtrlProtocol::makeFrameOfSetDriverFreq(int freq)

{

int data = int(freq\*4095/50);//0~50Hz对应0~4095

UINT8 dataH = data/256;

UINT8 dataL = data%256;

m\_sendBuf = "";

m\_sendBuf.append(0xFF).append(0xF8).append(dataH).append(dataL).append(0xFE);

}

UINT8 OldCtrlProtocol::readCtrlComBuffer(QByteArray tmp)

{

return 0;

}

void OldCtrlProtocol::analyseFrame()

{

}

void OldCtrlProtocol::makeFrameOfCtrlPressPump(bool status)

{

UINT8 zeroCode = 0x00;

m\_sendBuf = "";

if (status) //开打压泵

{

m\_sendBuf.append(0xFF).append(0xFB).append(zeroCode).append(zeroCode).append(0xFE);

}

else //false 关打压泵

{

m\_sendBuf.append(0xFF).append(0xFC).append(zeroCode).append(zeroCode).append(0xFE);

}

}

MeterProtocol::MeterProtocol()

{

m\_sendBuf = "";

}

MeterProtocol::~MeterProtocol()

{

}

QByteArray MeterProtocol::getSendFrame()

{

return m\_sendBuf;

}

QString MeterProtocol::getFullMeterNo()

{

return m\_fullMeterNo;

}

QString MeterProtocol::getFlow()

{

return m\_flow;

}

QString MeterProtocol::getInTemper()

{

return m\_inTemper;

}

QString MeterProtocol::getOutTemper()

{

return m\_outTemper;

}

QString MeterProtocol::getHeat()

{

return m\_heat;

}

QString MeterProtocol::getDate()

{

return m\_date;;

}

DeluMeterProtocol::DeluMeterProtocol()

{

m\_deluMeterFrame = new DeluMeter\_Frame\_Struct();

memset(m\_deluMeterFrame, 0, sizeof(DeluMeter\_Frame\_Struct));

}

DeluMeterProtocol::~DeluMeterProtocol()

{

if (m\_deluMeterFrame)

{

delete m\_deluMeterFrame;

m\_deluMeterFrame = NULL;

}

}

UINT8 DeluMeterProtocol::CountCheck(DeluMeter\_Frame\_Struct \*pFrame)

{

if (NULL == pFrame)

{

return 0;

}

UINT8 cs = 0;

cs = pFrame->startCode + pFrame->typeCode;

for (int i=0; i<METER\_ADDR\_LEN; i++)

{

cs += pFrame->addr[i];

}

cs += pFrame->ctrlCode;

cs += pFrame->dataLen;

for (int j=0; j<METER\_DATAID\_LEN; j++)

{

cs += pFrame->dataID[j];

}

cs += pFrame->sn;

for (int m=0; m<METER\_DATA\_LEN; m++)

{

cs += pFrame->data[m];

}

return cs;

}

UINT8 DeluMeterProtocol::readMeterComBuffer(QByteArray tmp)

{

qDebug()<<"readMeterComBuffer MeterProtocol thread:"<<QThread::currentThreadId();

UINT8 ret = 0x00;

int state = METER\_START\_STATE;

UINT8 ch = 0; //无符号8位数字

int number = tmp.size();

int m=0;

int addr\_num=0, dataID\_num=0, data\_num=0;

UINT8 ck=0; //程序计算的检验码

for (m=0; m<number; m++)

{

ch = (UINT8)tmp.at(m);

qDebug()<<"read data is:"<<ch;

if (ch == METER\_PREFIX\_CODE)

{

continue;

}

switch(state)

{

case METER\_START\_STATE: //

{

if (ch == METER\_START\_CODE)

{

m\_deluMeterFrame->startCode = METER\_START\_CODE;

state = METER\_TYPE\_STATE;

}

break;

}

case METER\_TYPE\_STATE: //

{

if (ch == METER\_TYPE\_ANSWER\_CODE) //响应

{

m\_deluMeterFrame->typeCode = ch;

state = METER\_ADDR\_STATE;

}

break;

}

case METER\_ADDR\_STATE: //

{

m\_deluMeterFrame->addr[addr\_num++] = ch;

if (addr\_num == METER\_ADDR\_LEN)

{

state = METER\_CTRL\_STATE;

addr\_num = 0;

}

break;

}

case METER\_CTRL\_STATE: //

{

m\_deluMeterFrame->ctrlCode = ch;

state = METER\_DATALEN\_STATE;

break;

}

case METER\_DATALEN\_STATE: //

{

m\_deluMeterFrame->dataLen = ch;

state = METER\_DATAID\_STATE;

break;

}

case METER\_DATAID\_STATE: //

{

m\_deluMeterFrame->dataID[dataID\_num++] = ch;

if (dataID\_num == METER\_DATAID\_LEN)

{

state = METER\_SN\_STATE;

dataID\_num = 0;

}

break;

}

case METER\_SN\_STATE: //序列号

{

m\_deluMeterFrame->sn = ch;

state = METER\_DATA\_STATE;

break;

}

case METER\_DATA\_STATE: //

{

m\_deluMeterFrame->data[data\_num++] = ch;

if (data\_num == METER\_DATA\_LEN)

{

state = METER\_CS\_STATE;

data\_num = 0;

}

break;

}

case METER\_CS\_STATE: //

{

m\_deluMeterFrame->cs = ch;

state = METER\_END\_STATE;

break;

}

case METER\_END\_STATE: //

{

m\_deluMeterFrame->endCode = ch;

state = METER\_START\_STATE;

ck = CountCheck(m\_deluMeterFrame);

if (ck == m\_deluMeterFrame->cs) //校验通过

{

analyseFrame();

ret = 1; //

qDebug()<<"check is ok 校验通过";

}

break;

}

default :

{

state = METER\_START\_STATE;

break;

}

} //END OF switch(state)

} //END OF for (m=0; m<number; m++)

return ret;

}

void DeluMeterProtocol::analyseFrame()

{

if (NULL == m\_deluMeterFrame)

{

return;

}

m\_fullMeterNo = "";

for (int i=METER\_ADDR\_LEN-1; i>=0; i--)

{

m\_fullMeterNo.append(QString("%1").arg(m\_deluMeterFrame->addr[i], 2, 16)).replace(' ', '0');

}

m\_inTemper = "";

m\_inTemper.append(QString("%1%2.%3").arg(m\_deluMeterFrame->data[2], 2, 16)\

.arg(m\_deluMeterFrame->data[1], 2, 16).arg(m\_deluMeterFrame->data[0], 2, 16));

m\_inTemper.replace(' ', '0');

m\_flow = "";

m\_flow.append(QString("%1.%2%3%4").arg(m\_deluMeterFrame->data[9], 2, 16)\

.arg(m\_deluMeterFrame->data[8], 2, 16).arg(m\_deluMeterFrame->data[7], 2, 16)\

.arg(m\_deluMeterFrame->data[6], 2, 16));

m\_flow.replace(' ', '0');

m\_heat = "";

m\_heat.append(QString("%1%2.%3%4").arg(m\_deluMeterFrame->data[14], 2, 16)\

.arg(m\_deluMeterFrame->data[13], 2, 16).arg(m\_deluMeterFrame->data[12], 2, 16)\

.arg(m\_deluMeterFrame->data[11], 2, 16));

m\_heat.replace(' ', '0');

m\_outTemper = "";

m\_outTemper.append(QString("%1%2.%3").arg(m\_deluMeterFrame->data[48], 2, 16)\

.arg(m\_deluMeterFrame->data[47], 2, 16).arg(m\_deluMeterFrame->data[46], 2, 16));

m\_outTemper.replace(' ', '0');

m\_date = "";

m\_date.append(QString("%1%2%3%4").arg(m\_deluMeterFrame->data[52], 2, 16)\

.arg(m\_deluMeterFrame->data[51], 2, 16).arg(m\_deluMeterFrame->data[50], 2, 16).\

arg(m\_deluMeterFrame->data[49], 2, 16));

m\_date.replace(' ', '0');

}

void DeluMeterProtocol::makeFrameOfReadMeter()

{

m\_sendBuf = "";

for (int i=0; i<WAKEUP\_CODE\_NUM; i++)

{

m\_sendBuf.append(METER\_WAKEUP\_CODE);//唤醒红外

}

for (int j=0; j<PREFIX\_CODE\_NUM; j++)

{

m\_sendBuf.append(METER\_PREFIX\_CODE); //前导字节

}

m\_sendBuf.append(METER\_START\_CODE);//起始符

m\_sendBuf.append(METER\_TYPE\_ASK\_CODE); //仪表类型 请求

for (int m=0; m<METER\_ADDR\_LEN; m++)

{

m\_sendBuf.append(METER\_ADDR\_CODE); //广播地址

}

m\_sendBuf.append(METER\_CTRL\_CODE);//控制码

m\_sendBuf.append(0x03);//数据长度

m\_sendBuf.append(0x3F).append(0x90); //数据标识

m\_sendBuf.append(0x03);//序列号

UINT8 cs = METER\_START\_CODE + METER\_TYPE\_ASK\_CODE + METER\_ADDR\_CODE\*METER\_ADDR\_LEN + METER\_CTRL\_CODE\

+ 0x03 + 0x3F + 0x90 + 0x03;

m\_sendBuf.append(cs);//校验码

m\_sendBuf.append(METER\_END\_CODE);//结束符

}

void DeluMeterProtocol::makeFrameOfSetVerifyStatus()

{

m\_sendBuf = "";

for (int i=0; i<WAKEUP\_CODE\_NUM; i++)

{

m\_sendBuf.append(METER\_WAKEUP\_CODE);//唤醒红外

}

for (int j=0; j<PREFIX\_CODE\_NUM; j++)

{

m\_sendBuf.append(METER\_PREFIX\_CODE); //前导字节

}

m\_sendBuf.append(METER\_START\_CODE);//起始符

m\_sendBuf.append(METER\_TYPE\_ASK\_CODE); //仪表类型 请求

for (int m=0; m<METER\_ADDR\_LEN; m++)

{

m\_sendBuf.append(METER\_ADDR\_CODE); //广播地址

}

UINT8 code0 = 0x00;

m\_sendBuf.append(0x33).append(code0).append(0x61).append(0x16);

}

void DeluMeterProtocol::makeFrameOfModifyMeterNo(QString oldMeterNo, QString newMeterNo)

{

m\_sendBuf = "";

for (int i=0; i<WAKEUP\_CODE\_NUM; i++)

{

m\_sendBuf.append(METER\_WAKEUP\_CODE);//唤醒红外

}

for (int j=0; j<PREFIX\_CODE\_NUM; j++)

{

m\_sendBuf.append(METER\_PREFIX\_CODE); //前导字节

}

m\_sendBuf.append(METER\_START\_CODE);//起始符

m\_sendBuf.append(METER\_TYPE\_ASK\_CODE); //仪表类型 请求

UINT8 cs = METER\_START\_CODE + METER\_TYPE\_ASK\_CODE;

UINT8 oldNo;

bool ok;

for (int m=METER\_ADDR\_LEN-1; m>=0; m--)

{

oldNo = oldMeterNo.mid(2\*m, 2).toUInt(&ok, 16);

m\_sendBuf.append(oldNo); //旧表号

cs += oldNo;

}

UINT8 code1 = 0x39;

UINT8 code2 = 0x11;

UINT8 code3 = 0x18;

UINT8 code4 = 0xA0;

UINT8 code5 = 0xAA;

m\_sendBuf.append(code1).append(code2).append(code3).append(code4).append(code5);

cs += code1 + code2 + code3 + code4 + code5;

UINT8 newNo;

for (int n=METER\_ADDR\_LEN-1; n>=0; n--)

{

newNo = newMeterNo.mid(2\*n, 2).toUInt(&ok, 16);

m\_sendBuf.append(newNo); //新表号

cs += newNo;

}

UINT8 timeCode;

QString currentTime = QDateTime::currentDateTime().toString("yyyyMMddHHmmss");//"20150107125930"

for (int p=6; p>=0; p--)

{

timeCode = currentTime.mid(2\*p, 2).toUInt(&ok, 16);

m\_sendBuf.append(timeCode); //当前时间

cs += timeCode;

}

m\_sendBuf.append(cs).append(0x16);

}

#include <QtGui/QMessageBox>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlQuery>

#include <QtCore/QDebug>

#include "logindialog.h"

#include "commondefine.h"

LoginDialog::LoginDialog(QWidget \*parent)

: QDialog(parent)

{

ui.setupUi(this);

ui.passwordEdit->setFocus();

ui.passwordEdit->setEchoMode(QLineEdit::Password);

ui.loginButton->setDefault(true);

initUserComboBox();

}

LoginDialog::~LoginDialog()

{

qDebug()<<"!!! CLoginDialog destructor";

}

void LoginDialog::initUserComboBox()

{

int idx=0;

QSqlQuery query;

if (query.exec("select F\_Desc from T\_User\_Def\_Tab order by F\_ID"))

{

while(query.next())

{

ui.userComboBox->insertItem(idx++, query.value(0).toString());

}

}

}

void LoginDialog::on\_loginButton\_clicked()

{

if (ui.passwordEdit->text().isEmpty())

{

QMessageBox::warning(this, tr("Warning"), tr("please input password!"));

ui.passwordEdit->setFocus();

}

else

{

QSqlQuery query;

query.exec(QString("select F\_Password from T\_User\_Def\_Tab where F\_ID=%1").arg(ui.userComboBox->currentIndex()));

query.next();

if (query.value(0).toString() == ui.passwordEdit->text())

{

QDialog::accept();

}

else

{

QMessageBox::warning(this, tr("Warning"), tr("password error!"));

ui.passwordEdit->clear();

ui.passwordEdit->setFocus();

}

}

}

void LoginDialog::on\_quitButton\_clicked()

{

QDialog::reject();//隐藏对话框，发送拒绝信号

}

#include <QtGui/QMessageBox>

#include <QtCore/QDebug>

#include <QtCore/QTimer>

#include <QtCore/QThread>

#include <QTest>

#include <QtSql/QSqlTableModel>

#include <QtGui/QFileDialog>

#include <QtCore/QSignalMapper>

#include <math.h>

#include "flowweight.h"

#include "commondefine.h"

#include "algorithm.h"

#include "qtexdb.h"

#include "parasetdlg.h"

#include "readcomconfig.h"

FlowWeightDlg::FlowWeightDlg(QWidget \*parent, Qt::WFlags flags)

: QWidget(parent, flags)

{

qDebug()<<"FlowWeightDlg thread:"<<QThread::currentThreadId();

ui.setupUi(this);

ui.btnNext->hide(); //隐藏"下一步"按钮

ui.btnStart->hide();

m\_inputStartValue = false;

m\_inputEndValue = false;

m\_gradeErr[1] = 1.00f;

m\_gradeErr[2] = 2.00f;

m\_gradeErr[3] = 3.00f;

if (!getPortSetIni(&m\_portsetinfo)) //获取下位机端口号配置信息

{

QMessageBox::warning(this, tr("Warning"), tr("Warning:get port set info failed!"));

}

m\_stopFlag = false; //退出界面后，不再检查天平容量

m\_readComConfig = new ReadComConfig(); //读串口设置接口

m\_balanceObj = NULL;

initBalanceCom(); //初始化天平串口

m\_tempObj = NULL;

m\_tempTimer = NULL;

initTemperatureCom(); //初始化温度采集串口

m\_controlObj = NULL;

initControlCom(); //初始化控制串口

m\_balLastValue = 0.0; //用于判断天平值是否发生突变

m\_totalcount = 0;

m\_startWeight = 0.0;

m\_endWeight = 0.0;

memset(m\_deltaWeight, 0, sizeof(float)\*FLOW\_SAMPLE\_NUM);

m\_flowRateTimer = new QTimer();

connect(m\_flowRateTimer, SIGNAL(timeout()), this, SLOT(slotFreshFlowRate()));

m\_flowRateTimer->start(TIMEOUT\_FLOW\_SAMPLE);

m\_chkAlg = new CAlgorithm();//计算类接口

initValveStatus(); //映射关系；初始化阀门状态

m\_exaustTimer = new QTimer(this); //排气定时器

connect(m\_exaustTimer, SIGNAL(timeout()), this, SLOT(slotExaustFinished()));

m\_tempCount = 1; //计算平均温度用的累加计数器

m\_nowOrder = 0; //当前进行的检定序号

m\_nowParams = new Verify\_Params\_STR;

memset(m\_nowParams, 0, sizeof(Verify\_Params\_STR));

m\_continueVerify = true; //连续检定

m\_resetZero = false; //初值回零

m\_autopick = false; //自动采集

m\_flowPointNum = 0; //流量点个数

m\_maxMeterNum = 0; //某规格表最多支持的检表个数

m\_validMeterNum = 0; //实际检表个数

m\_exaustSecond = 45; //默认排气时间45秒

m\_manufac = 0; //制造厂商 默认德鲁

m\_meterStartValue = NULL;

m\_meterEndValue = NULL;

m\_meterTemper = NULL;

m\_meterDensity = NULL;

m\_meterStdValue = NULL;

m\_meterError = NULL;

m\_balStartV = 0;

m\_balEndV = 0;

m\_timeStamp = "";

m\_nowDate = "";

m\_validDate = "";

m\_flowPoint = 0; //流量(m3/h)

m\_recPtr = NULL;

QSqlTableModel \*model = new QSqlTableModel(this);

model->setTable("T\_Meter\_Standard");

model->select();

m\_meterStdMapper = new QDataWidgetMapper(this);

m\_meterStdMapper->setSubmitPolicy(QDataWidgetMapper::ManualSubmit);

m\_meterStdMapper->setModel(model);

m\_meterStdMapper->addMapping(ui.lnEditStandard, 1);

m\_paraSetDlg = NULL; //参数设置对话框

m\_paraSetReader = new ParaSetReader(); //读参数设置接口

if (!readNowParaConfig()) //获取当前检定参数

{

qWarning()<<"读取参数配置文件失败!";

}

if (!isComAndPortNormal())

{

qWarning()<<"串口、端口设置错误!";

}

m\_meterThread = NULL;

m\_meterObj = NULL;

initMeterCom(); //初始化热量表通讯串口

if (!isWaterOutValveOpen())

{

qDebug()<<"放水阀门未打开";

openWaterOutValve();

}

}

FlowWeightDlg::~FlowWeightDlg()

{

}

void FlowWeightDlg::closeEvent( QCloseEvent \* event)

{

qDebug()<<"^^^^^FlowWeightDlg::closeEvent";

m\_stopFlag = true;

if (m\_paraSetReader) //读检定参数

{

delete m\_paraSetReader;

m\_paraSetReader = NULL;

}

if (m\_paraSetDlg) //参数设置对话框

{

delete m\_paraSetDlg;

m\_paraSetDlg = NULL;

}

if (m\_readComConfig) //读串口设置

{

delete m\_readComConfig;

m\_readComConfig = NULL;

}

if (m\_tempObj) // 温度采集

{

delete m\_tempObj;

m\_tempObj = NULL;

m\_tempThread.exit(); //否则日志中会有警告"QThread: Destroyed while thread is still running"

}

if (m\_tempTimer) //计时器

{

if (m\_tempTimer->isActive())

{

m\_tempTimer->stop();

}

delete m\_tempTimer;

m\_tempTimer = NULL;

}

if (m\_balanceObj) //天平采集

{

delete m\_balanceObj;

m\_balanceObj = NULL;

m\_balanceThread.exit();

}

if (m\_controlObj) //阀门控制

{

delete m\_controlObj;

m\_controlObj = NULL;

m\_valveThread.exit();

}

if (m\_chkAlg)//计算类

{

delete m\_chkAlg;

m\_chkAlg = NULL;

}

if (m\_meterObj)

{

delete []m\_meterObj;

m\_meterObj = NULL;

for (int i=0; i<m\_maxMeterNum; i++)

{

m\_meterThread[i].exit();

}

}

}

void FlowWeightDlg::initBalanceCom()

{

ComInfoStruct balanceStruct = m\_readComConfig->ReadBalanceConfig();

m\_balanceObj = new BalanceComObject();

m\_balanceObj->moveToThread(&m\_balanceThread);

m\_balanceThread.start();

m\_balanceObj->openBalanceCom(&balanceStruct);

connect(m\_balanceObj, SIGNAL(balanceValueIsReady(const float &)), this, SLOT(slotFreshBalanceValue(const float &)));

}

void FlowWeightDlg::initTemperatureCom()

{

ComInfoStruct tempStruct = m\_readComConfig->ReadTempConfig();

m\_tempObj = new TempComObject();

m\_tempObj->moveToThread(&m\_tempThread);

m\_tempThread.start();

m\_tempObj->openTemperatureCom(&tempStruct);

connect(m\_tempObj, SIGNAL(temperatureIsReady(const QString &)), this, SLOT(slotFreshComTempValue(const QString &)));

m\_tempTimer = new QTimer();

connect(m\_tempTimer, SIGNAL(timeout()), m\_tempObj, SLOT(writeTemperatureComBuffer()));

m\_tempTimer->start(TIMEOUT\_TEMPER); //周期请求温度

}

void FlowWeightDlg::initControlCom()

{

ComInfoStruct valveStruct = m\_readComConfig->ReadValveConfig();

m\_controlObj = new ControlComObject();

m\_controlObj->setProtocolVersion(m\_portsetinfo.version);

m\_controlObj->moveToThread(&m\_valveThread);

m\_valveThread.start();

m\_controlObj->openControlCom(&valveStruct);

connect(m\_controlObj, SIGNAL(controlRelayIsOk(const UINT8 &, const bool &)), this, SLOT(slotSetValveBtnStatus(const UINT8 &, const bool &)));

connect(m\_controlObj, SIGNAL(controlRegulateIsOk()), this, SLOT(slotSetRegulateOk()));

}

void FlowWeightDlg::initMeterCom()

{

if (m\_maxMeterNum <= 0)

{

return;

}

m\_meterThread = new ComThread[m\_maxMeterNum];

m\_meterObj = new MeterComObject[m\_maxMeterNum];

int i=0;

for (i=0; i<m\_maxMeterNum; i++)

{

m\_meterObj[i].moveToThread(&m\_meterThread[i]);

m\_meterObj[i].setProtocolVersion(m\_manufac); //设置表协议类型

m\_meterThread[i].start();

m\_meterObj[i].openMeterCom(&m\_readComConfig->ReadMeterConfigByNum(QString("%1").arg(i+1)));

connect(&m\_meterObj[i], SIGNAL(readMeterNoIsOK(const QString&, const QString&)), this, SLOT(slotSetMeterNumber(const QString&, const QString&)));

connect(&m\_meterObj[i], SIGNAL(readMeterFlowIsOK(const QString&, const float&)), this, SLOT(slotSetMeterFlow(const QString&, const float&)));

}

}

void FlowWeightDlg::initValveStatus()

{

m\_nowPortNo = 0;

m\_valveBtn[m\_portsetinfo.bigNo] = ui.btnValveBig;

m\_valveBtn[m\_portsetinfo.smallNo] = ui.btnValveSmall;

m\_valveBtn[m\_portsetinfo.middle1No] = ui.btnValveMiddle1;

m\_valveBtn[m\_portsetinfo.middle2No] = ui.btnValveMiddle2;

m\_valveBtn[m\_portsetinfo.waterInNo] = ui.btnWaterIn;

m\_valveBtn[m\_portsetinfo.waterOutNo] = ui.btnWaterOut;

m\_valveBtn[m\_portsetinfo.pumpNo] = ui.btnWaterPump; //水泵

m\_valveStatus[m\_portsetinfo.bigNo] = VALVE\_CLOSE;

m\_valveStatus[m\_portsetinfo.smallNo] = VALVE\_CLOSE;

m\_valveStatus[m\_portsetinfo.middle1No] = VALVE\_CLOSE;

m\_valveStatus[m\_portsetinfo.middle2No] = VALVE\_CLOSE;

m\_valveStatus[m\_portsetinfo.waterInNo] = VALVE\_CLOSE;

m\_valveStatus[m\_portsetinfo.waterOutNo] = VALVE\_CLOSE;

m\_valveStatus[m\_portsetinfo.pumpNo] = VALVE\_CLOSE; //水泵

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.bigNo], m\_valveStatus[m\_portsetinfo.bigNo]);

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.smallNo], m\_valveStatus[m\_portsetinfo.smallNo]);

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.middle1No], m\_valveStatus[m\_portsetinfo.middle1No]);

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.middle2No], m\_valveStatus[m\_portsetinfo.middle2No]);

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.waterInNo], m\_valveStatus[m\_portsetinfo.waterInNo]);

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.waterOutNo], m\_valveStatus[m\_portsetinfo.waterOutNo]);

setValveBtnBackColor(m\_valveBtn[m\_portsetinfo.pumpNo], m\_valveStatus[m\_portsetinfo.pumpNo]);

}

void FlowWeightDlg::slotFreshBalanceValue(const float& balValue)

{

if (fabs(m\_balLastValue - balValue) > 1) //天平每次变化不可能大于1kg

{

m\_balLastValue = balValue;

return;

}

QString wht = QString::number(balValue, 'f', 3);

ui.lcdBigBalance->display(wht);

m\_balLastValue = balValue;

if (balValue > BALANCE\_CAPACITY) //防止天平溢出 暂设天平容量为100kg

{

m\_controlObj->askControlRelay(m\_portsetinfo.waterInNo, VALVE\_CLOSE);// 关闭进水阀

m\_controlObj->askControlRelay(m\_portsetinfo.waterOutNo, VALVE\_OPEN);// 打开放水阀

if (m\_portsetinfo.version == OLD\_CTRL\_VERSION) //老控制板 无反馈

{

slotSetValveBtnStatus(m\_portsetinfo.waterOutNo, VALVE\_OPEN);

slotSetValveBtnStatus(m\_portsetinfo.waterInNo, VALVE\_CLOSE);

}

}

}

void FlowWeightDlg::slotFreshComTempValue(const QString& tempStr)

{

ui.lcdInTemper->display(tempStr.left(DATA\_WIDTH)); //入口温度 PV

ui.lcdOutTemper->display(tempStr.right(DATA\_WIDTH)); //出口温度 SV

}

void FlowWeightDlg::slotFreshFlowRate()

{

qDebug()<<"FlowWeightDlg::slotFreshFlow thread:"<<QThread::currentThreadId();

if (m\_totalcount > 4294967290) //防止m\_totalcount溢出 32位无符号整数范围0~4294967295

{

m\_totalcount = 0;

m\_startWeight = 0.0;

m\_endWeight = 0.0;

memset(m\_deltaWeight, 0, sizeof(float)\*FLOW\_SAMPLE\_NUM);

}

if (m\_totalcount == 0) //记录天平初始重量

{

m\_startWeight = ui.lcdBigBalance->value();

m\_totalcount ++;

return;

}

float flowValue = 0.0;

float totalWeight = 0.0;

m\_endWeight = ui.lcdBigBalance->value();//取当前天平值, 作为当前运算的终值

float delta\_weight = m\_endWeight - m\_startWeight;

m\_deltaWeight[m\_totalcount%FLOW\_SAMPLE\_NUM] = delta\_weight;

for (int i=0; i<FLOW\_SAMPLE\_NUM; i++)

{

totalWeight += m\_deltaWeight[i];

}

flowValue = 3.6\*(totalWeight)\*1000/(FLOW\_SAMPLE\_NUM\*TIMEOUT\_FLOW\_SAMPLE); ui.lcdFlowRate->display(QString::number(flowValue, 'f', 3)); //在ui.lcdFlowRate中显示流速

m\_totalcount ++;//计数器累加

m\_startWeight = m\_endWeight;//将当前值保存, 作为下次运算的初值

}

int FlowWeightDlg::isComAndPortNormal()

{

return true;

}

int FlowWeightDlg::isWaterOutValveOpen()

{

return true;

}

int FlowWeightDlg::readNowParaConfig()

{

if (NULL == m\_paraSetReader)

{

return false;

}

m\_nowParams = m\_paraSetReader->getParams();

m\_continueVerify = m\_nowParams->bo\_converify; //连续检定

m\_resetZero = m\_nowParams->bo\_resetzero; //初值回零

m\_autopick = m\_nowParams->bo\_autopick; //自动采集

m\_flowPointNum = m\_nowParams->total\_fp; //有效流量点的个数

m\_exaustSecond = m\_nowParams->ex\_time; //排气时间

m\_standard = m\_nowParams->m\_stand; //表规格

m\_model = m\_nowParams->m\_model; //表型号

m\_meterType = m\_nowParams->m\_type;//表类型

m\_maxMeterNum = m\_nowParams->m\_maxMeters;//不同表规格对应的最大检表数量

m\_manufac = m\_nowParams->m\_manufac; //制造厂商

initTableWidget();

showNowKeyParaConfig();

return true;

}

void FlowWeightDlg::initTableWidget()

{

if (m\_maxMeterNum <= 0)

{

return;

}

ui.tableWidget->setRowCount(m\_maxMeterNum); //设置表格行数

QSignalMapper \*m\_signalMapper1 = new QSignalMapper();

QSignalMapper \*m\_signalMapper2 = new QSignalMapper();

QStringList vLabels;

for (int i=0; i< m\_maxMeterNum; i++)

{

vLabels<<QString(QObject::tr("meterPosNo%1").arg(i+1));

ui.tableWidget->setItem(i, COLUMN\_METER\_NUMBER, new QTableWidgetItem(QString()));

ui.tableWidget->setItem(i, COLUMN\_METER\_START, new QTableWidgetItem(QString()));

ui.tableWidget->setItem(i, COLUMN\_METER\_END, new QTableWidgetItem(QString()));

QPushButton \*btnModNo = new QPushButton(tr("Modify NO."));

ui.tableWidget->setCellWidget(i, COLUMN\_MODIFY\_METERNO, btnModNo);

m\_signalMapper1->setMapping(btnModNo, i);

connect(btnModNo, SIGNAL(clicked()), m\_signalMapper1, SLOT(map()));

QPushButton \*btnAdjErr = new QPushButton(tr("Adjust Err"));

ui.tableWidget->setCellWidget(i, COLUMN\_ADJUST\_ERROR, btnAdjErr);

m\_signalMapper2->setMapping(btnAdjErr, i);

connect(btnAdjErr, SIGNAL(clicked()), m\_signalMapper2, SLOT(map()));

}

connect(m\_signalMapper1, SIGNAL(mapped(const int &)),this, SLOT(slotModifyMeterNo(const int &)));

connect(m\_signalMapper2, SIGNAL(mapped(const int &)),this, SLOT(slotAdjustError(const int &)));

ui.tableWidget->setVerticalHeaderLabels(vLabels);

}

void FlowWeightDlg::showNowKeyParaConfig()

{

if (NULL == m\_nowParams)

{

return;

}

ui.cmbAutoPick->setCurrentIndex(m\_nowParams->bo\_autopick);

ui.cmbContinue->setCurrentIndex(m\_nowParams->bo\_converify);

ui.cmbResetZero->setCurrentIndex(m\_nowParams->bo\_resetzero);

m\_meterStdMapper->setCurrentIndex(m\_nowParams->m\_stand);

}

void FlowWeightDlg::on\_btnExhaust\_clicked()

{

if (!isDataCollectNormal())

{

qWarning()<<"数据采集不正常，请检查";

return;

}

if (!openAllValveAndPump())

{

qWarning()<<"打开所有阀门和水泵 失败!";

return;

}

m\_stopFlag = false;

clearTableContents();

m\_validMeterNum = 0;

m\_exaustSecond = m\_nowParams->ex\_time;

m\_exaustTimer->start(1000);//开始排气倒计时

ui.labelHintProcess->setText(tr("Exhaust countdown: %1 second").arg(m\_exaustSecond));

qDebug()<<"排气倒计时:"<<m\_exaustSecond<<"秒";

if (m\_autopick) //自动读表

{

readMeter();

}

else //手动读表

{

ui.labelHintPoint->setText(tr("Please input meter number!"));

}

return;

}

int FlowWeightDlg::isDataCollectNormal()

{

return true;

}

int FlowWeightDlg::openAllValveAndPump()

{

openValve(m\_portsetinfo.waterOutNo);

openValve(m\_portsetinfo.bigNo);

openValve(m\_portsetinfo.middle1No);

openValve(m\_portsetinfo.middle2No);

openValve(m\_portsetinfo.smallNo);

openValve(m\_portsetinfo.waterInNo);

m\_controlObj->askControlWaterPump(m\_portsetinfo.pumpNo, VALVE\_OPEN);

if (m\_portsetinfo.version == OLD\_CTRL\_VERSION) //老控制板 无反馈

{

slotSetValveBtnStatus(m\_portsetinfo.pumpNo, VALVE\_OPEN);

}

return true;

}

void FlowWeightDlg::slotExaustFinished()

{

m\_exaustSecond --;

ui.labelHintProcess->setText(tr("Exhaust countdown: %1 second").arg(m\_exaustSecond));

qDebug()<<"排气倒计时:"<<m\_exaustSecond<<"秒";

if (m\_exaustSecond > 1)

{

return;

}

m\_exaustTimer->stop(); //停止排气计时

ui.labelHintProcess->setText(tr("Exhaust countdown finished!"));

ui.labelHintProcess->clear();

if (!closeAllFlowPointValves()) //关闭所有流量点阀门 失败

{

if (!closeAllFlowPointValves()) //再尝试关闭一次

{

qWarning()<<"关闭所有流量点阀门失败，检定结束";

return;

}

}

ui.labelHintPoint->setText(tr("prepare balance ..."));

if (ui.lcdBigBalance->value() < BALANCE\_START\_VALUE)

{

if (!closeWaterOutValve())

{

qWarning()<<"关闭放水阀失败";

}

if (!openBigFlowValve())

{

qWarning()<<"打开大流量阀失败";

}

}

if (judgeBalanceInitValue(BALANCE\_START\_VALUE))

{

if (!closeBigFlowValve())

{

qWarning()<<"关闭大流量阀失败";

}

}

if (setMeterVerifyStatus()) //设置检定状态成功

{

startVerify();

}

}

int FlowWeightDlg::readMeter()

{

on\_btnReadMeter\_clicked();

return true;

}

int FlowWeightDlg::setMeterVerifyStatus()

{

for (int i=0; i<m\_maxMeterNum; i++)

{

m\_meterObj[i].askSetVerifyStatus();

}

return true;

}

int FlowWeightDlg::closeAllFlowPointValves()

{

closeValve(m\_portsetinfo.bigNo);

closeValve(m\_portsetinfo.middle1No);

closeValve(m\_portsetinfo.middle2No);

closeValve(m\_portsetinfo.smallNo);

return true;

}

int FlowWeightDlg::closeWaterOutValve()

{

closeValve(m\_portsetinfo.waterOutNo);

return true;

}

int FlowWeightDlg::openWaterOutValve()

{

openValve(m\_portsetinfo.waterOutNo);

return true;

}

int FlowWeightDlg::openBigFlowValve()

{

openValve(m\_portsetinfo.bigNo);

return true;

}

int FlowWeightDlg::closeBigFlowValve()

{

closeValve(m\_portsetinfo.bigNo);

return true;

}

int FlowWeightDlg::judgeBalanceInitValue(float v)

{

while (!m\_stopFlag && (ui.lcdBigBalance->value() < v))

{

qDebug()<<"天平重量 ="<<ui.lcdBigBalance->value()<<", 小于要求的重量 "<<v;

QTest::qWait(1000);

}

return true;

}

int FlowWeightDlg::judgeBalanceAndCalcAvgTemper(float targetV)

{

int second;

float nowFlow =m\_paraSetReader->getFpBySeq(m\_nowOrder).fp\_verify;;

while (!m\_stopFlag && (ui.lcdBigBalance->value() < targetV))

{

qDebug()<<"天平重量 ="<<ui.lcdBigBalance->value()<<", 小于要求的重量 "<<targetV;

m\_pipeInTemper += ui.lcdInTemper->value();

m\_pipeOutTemper += ui.lcdOutTemper->value();

m\_tempCount++;

second = 3.6\*(targetV - ui.lcdBigBalance->value())/nowFlow;

ui.labelHintPoint->setText(tr("NO. %1 flow point: %2 m3/h").arg(m\_nowOrder).arg(nowFlow));

ui.labelHintProcess->setText(tr("Verifying...\nPlease wait for about %1 second").arg(second));

QTest::qWait(1000);

}

m\_pipeInTemper = m\_pipeInTemper/m\_tempCount; //入口平均温度

m\_pipeOutTemper = m\_pipeOutTemper/m\_tempCount; //出口平均温度

ui.labelHintPoint->setText(tr("NO. %1 flow point: %2 m3/h").arg(m\_nowOrder).arg(nowFlow));

ui.labelHintProcess->setText(tr("Verify Finished!"));

if (m\_nowOrder == m\_flowPointNum)

{

ui.labelHintProcess->setText(tr("All flow points has verified !"));

ui.btnNext->hide();

}

return true;

}

void FlowWeightDlg::clearTableContents()

{

for (int i=0; i<m\_maxMeterNum; i++)

{

for (int j=1; j<ui.tableWidget->columnCount(); j++) //从第二列开始

{

if (ui.tableWidget->item(i,j) == 0)

{

continue;

}

ui.tableWidget->item(i,j)->setText("");

}

}

}

void FlowWeightDlg::on\_btnStart\_clicked()

{

startVerify();

ui.btnStart->hide();

}

void FlowWeightDlg::on\_btnNext\_clicked()

{

if (m\_nowOrder >= m\_flowPointNum)

{

QMessageBox::warning(this, tr("Warning"), tr("all flow points has verified!"));

return;

}

clearTableContents();

m\_nowOrder ++;

if (m\_nowOrder == m\_flowPointNum)

{

ui.btnNext->hide();

}

prepareVerifyFlowPoint(m\_nowOrder); // 开始进行下一次流量点的检定

}

void FlowWeightDlg::on\_btnStop\_clicked()

{

m\_stopFlag = true; //不再检查天平质量

m\_inputStartValue = false;

m\_inputEndValue = false;

m\_exaustTimer->stop();//停止排气定时器

ui.labelHintProcess->setText(tr("Verify has Stoped!"));

}

void FlowWeightDlg::startVerify()

{

m\_nowOrder = 1;

if (getValidMeterNum() <= 0)

{

QMessageBox::warning(this, tr("Warning"), tr("please input meter number, then click \"start\" button!"));//请输入表号！然后点击'开始'按钮

ui.btnStart->show();

return;

}

if (m\_recPtr != NULL)

{

delete []m\_recPtr;

m\_recPtr = NULL;

}

m\_recPtr = new Flow\_Verify\_Record\_STR[m\_validMeterNum];

memset(m\_recPtr, 0, sizeof(Flow\_Verify\_Record\_STR)\*m\_validMeterNum);

m\_flowPoint = m\_paraSetReader->getFpBySeq(1).fp\_verify;//第一个流量点

for (int m=0; m<m\_validMeterNum; m++) //

{

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_FLOW\_POINT, new QTableWidgetItem(QString::number(m\_flowPoint, 'f', 2)));//流量点

}

m\_timeStamp = QDateTime::currentDateTime().toString("yyyy-MM-dd HH:mm:ss.zzz"); m\_nowDate = QDateTime::currentDateTime().toString("yyyy-MM-dd"); //当前日期'2014-08-07'

m\_validDate=DateTime::currentDateTime().addYears(VALID\_YEAR).addDays(-1).toString("yyyy-MM-dd");

m\_startValueFlag = true; //默认是初值

m\_meterStartValue = new float[m\_validMeterNum]; //表初值

memset(m\_meterStartValue, 0, sizeof(float)\*m\_validMeterNum);

m\_meterEndValue = new float[m\_validMeterNum]; //表终值

memset(m\_meterEndValue, 0, sizeof(float)\*m\_validMeterNum);

m\_meterTemper = new float[m\_validMeterNum]; //表温度

memset(m\_meterTemper, 0, sizeof(float)\*m\_validMeterNum);

m\_meterDensity = new float[m\_validMeterNum]; //表密度

memset(m\_meterDensity, 0, sizeof(float)\*m\_validMeterNum);

m\_meterStdValue = new float[m\_validMeterNum]; //被检表的标准值

memset(m\_meterStdValue, 0, sizeof(float)\*m\_validMeterNum);

m\_meterError = new float[m\_validMeterNum]; //被检表的误差

memset(m\_meterError, 0, sizeof(float)\*m\_validMeterNum);

if (m\_continueVerify) //连续检定

{

if (!judgeBalanceCapacity()) //判断天平容量是否能够满足检定用量

{

openWaterOutValve();

while (!judgeBalanceCapacity())

{

QTest::qWait(1000);

}

closeWaterOutValve(); //若满足检定用量，则关闭放水阀

QTest::qWait(3000); //等待3秒钟(等待水流稳定)

}

}

if (m\_autopick) //自动采集

{

for (int j=0; j<m\_flowPointNum; j++)

{

m\_nowOrder = j+1;

prepareVerifyFlowPoint(j+1);

}

}

else //手动采集

{

if (prepareVerifyFlowPoint(1)) //第一个流量点检定

{

ui.btnNext->show();

ui.btnNext->setFocus();

}

}

}

int FlowWeightDlg::getValidMeterNum()

{

bool ok;

for (int i=0; i<m\_maxMeterNum; i++)

{

if (NULL == ui.tableWidget->item(i, COLUMN\_METER\_NUMBER)) //"表号"单元格为空

{

continue;

}

ui.tableWidget->item(i, COLUMN\_METER\_NUMBER)->text().toInt(&ok, 10);

if (!ok) //表号转换失败(非数字)

{

continue;

}

m\_meterPosMap[m\_validMeterNum] = i+1; //表位号从1开始

m\_validMeterNum++;

}

return m\_validMeterNum;

}

bool FlowWeightDlg::judgeBalanceCapacity()

{

bool ret = false;

float totalQuantity = 0;

int num = m\_paraSetReader->getParams()->total\_fp; //有效流量点的个数

for (int i=0; i<num; i++)

{

totalQuantity += m\_paraSetReader->getParams()->fp\_info[i].fp\_quantity;

}

ret = (ui.lcdBigBalance->value() + totalQuantity) < BALANCE\_CAPACITY;

return ret;

}

int FlowWeightDlg::judgeBalanceCapacitySingle(int order)

{

return true;

}

int FlowWeightDlg::prepareVerifyFlowPoint(int order)

{

if (order < 1)

{

return false;

}

if (m\_validMeterNum <= 0)

{

QMessageBox::warning(this, tr("Warning"), tr("please input meter number"));

return false;

}

if (!m\_continueVerify) //非连续检定，每次检定开始之前都要判断天平容量

{

if (!judgeBalanceCapacitySingle(order)) //天平容量不满足本次检定用量

{

openWaterOutValve(); //打开放水阀，天平放水

while (!judgeBalanceCapacitySingle(order)) //等待天平放水，直至满足本次检定用量

{

QTest::qWait(1000);

}

closeWaterOutValve(); //若满足检定用量，则关闭放水阀

QTest::qWait(3000); //等待3秒钟，等待水流稳定

}

}

int i=0;

if (m\_resetZero) //初值回零

{

if (m\_autopick || order==1 ) //自动采集或者是第一个检定点,需要等待热表初值回零

{

ui.labelHintPoint->setText(tr("Reset Zero"));

while (i < RESET\_ZERO\_TIME) //等待被检表初值回零

{

ui.labelHintProcess->setText(tr("please wait %1 seconds for reset zero").arg(RESET\_ZERO\_TIME-i));

i++;

QTest::qWait(1000);

}

}

memset(m\_meterStartValue, 0, sizeof(float)\*m\_validMeterNum);

for (int i=0; i<m\_validMeterNum; i++)

{

ui.tableWidget->item(m\_meterPosMap[i]-1, COLUMN\_METER\_START)->setText(QString("%1").arg(m\_meterStartValue[i]));

}

}

else //初值不回零

{

if (order >= 2) //第二个检定点之后

{

if (m\_autopick)

{

QTest::qWait(2000); //等2秒，供操作人员看上一次的检定结果

clearTableContents();

}

makeStartValueByLastEndValue(); //上一次的终值作为本次的初值

}

else //第一个检定点

{

if (!getMeterStartValue()) //获取表初值

{

return false;

}

}

}

startVerifyFlowPoint(order);

return true;

}

int FlowWeightDlg::startVerifyFlowPoint(int order)

{

m\_balStartV = ui.lcdBigBalance->value(); //记录天平初值

m\_pipeInTemper = ui.lcdInTemper->value();

m\_pipeOutTemper = ui.lcdOutTemper->value();

m\_tempCount = 1;

m\_flowPoint = m\_paraSetReader->getFpBySeq(order).fp\_verify;//order对应的流量点

int portNo = m\_paraSetReader->getFpBySeq(order).fp\_valve; //order对应的阀门端口号

float verifyQuantity = m\_paraSetReader->getFpBySeq(order).fp\_quantity; if (openValve(portNo))

{

if (judgeBalanceAndCalcAvgTemper(m\_balStartV + verifyQuantity))

{

closeValve(portNo); //关闭order对应的阀门

QTest::qWait(3000); //等待3秒钟，让天平数值稳定

m\_balEndV = ui.lcdBigBalance->value(); //记录天平终值

for (int m=0; m<m\_validMeterNum; m++) //

{

m\_meterTemper[m] = m\_chkAlg->getMeterTempByPos(m\_pipeInTemper, m\_pipeOutTemper, m\_meterPosMap[m]);

m\_meterDensity[m] = m\_chkAlg->getDensityByQuery(m\_meterTemper[m]); m\_meterStdValue[m] = m\_chkAlg->getStdVolByPos((m\_balEndV-m\_balStartV), m\_pipeInTemper, m\_pipeOutTemper, m\_meterPosMap[m]);

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_FLOW\_POINT, new QTableWidgetItem(QString::number(m\_flowPoint, 'f', 2)));//流量点

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_BAL\_START, new QTableWidgetItem(QString::number(m\_balStartV, 'f', 3)));//天平初值

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_BAL\_END, new QTableWidgetItem(QString::number(m\_balEndV, 'f', 3))); //天平终值

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_TEMPER, new QTableWidgetItem(QString::number(m\_meterTemper[m], 'f', 2))); //温度

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_DENSITY, new QTableWidgetItem(QString::number(m\_meterDensity[m], 'f', 3)));//密度

ui.tableWidget->setItem(m\_meterPosMap[m]-1, COLUMN\_STD\_VALUE, new QTableWidgetItem(QString::number(m\_meterStdValue[m], 'f', 3)));//标准值

}

if (!getMeterEndValue()) //获取表终值

{

return false;

}

if (m\_autopick) //自动采集

{

calcAllMeterError();

saveAllVerifyRecords();

}

}

}

return true;

}

int FlowWeightDlg::calcAllMeterError()

{

for (int m=0; m<m\_validMeterNum; m++)

{

m\_meterError[m] = 100\*(m\_meterEndValue[m] - m\_meterStartValue[m] - m\_meterStdValue[m])/m\_meterStdValue[m];//计算每个表的误差,单位%

ui.tableWidget->setItem(m\_meterPosMap[m]-1,COLUMN\_ERROR,new QTableWidgetItem(QString::number(m\_meterError[m], 'f', 4))); //误差

}

QString meterNoPrefix = getNumPrefixOfManufac(m\_nowParams->m\_manufac);

QString meterNoStr;

for (int i=0; i<m\_validMeterNum; i++)

{

strncpy\_s(m\_recPtr[i].timestamp, m\_timeStamp.toAscii(), TIMESTAMP\_LEN);

m\_recPtr[i].flowPoint = m\_flowPoint;

meterNoStr=meterNoPrefix+ QString("%1").arg(ui.tableWidget->item(m\_meterPosMap[i]-1, 0)->text(), 8, '0');

strcpy\_s(m\_recPtr[i].meterNo, meterNoStr.toAscii());

m\_recPtr[i].flowPointIdx = m\_nowOrder;

m\_recPtr[i].methodFlag = WEIGHT\_METHOD; //质量法

m\_recPtr[i].meterValue0 = m\_meterStartValue[i];

m\_recPtr[i].meterValue1 = m\_meterEndValue[i];

m\_recPtr[i].balWeight0 = m\_balStartV;

m\_recPtr[i].balWeight1 = m\_balEndV;

m\_recPtr[i].pipeTemper = m\_meterTemper[i];

m\_recPtr[i].density = m\_meterDensity[i];

m\_recPtr[i].stdValue = m\_meterStdValue[i];

m\_recPtr[i].dispError = m\_meterError[i];

m\_recPtr[i].grade = m\_nowParams->m\_grade;

m\_recPtr[i].stdError = m\_gradeErr[m\_nowParams->m\_grade]; //二级表 标准误差

m\_recPtr[i].result = (fabs(m\_recPtr[i].dispError) <= fabs(m\_recPtr[i].stdError)) ? 1 : 0;

m\_recPtr[i].meterPosNo = m\_meterPosMap[i];

m\_recPtr[i].standard = m\_standard;

m\_recPtr[i].model = m\_model;

m\_recPtr[i].meterType = m\_meterType; //表类型

m\_recPtr[i].manufactDept = m\_nowParams->m\_manufac;

m\_recPtr[i].verifyDept = m\_nowParams->m\_vcomp;

m\_recPtr[i].verifyPerson = m\_nowParams->m\_vperson;

strncpy\_s(m\_recPtr[i].verifyDate, m\_nowDate.toAscii(), DATE\_LEN);

strncpy\_s(m\_recPtr[i].validDate, m\_validDate.toAscii(), DATE\_LEN);

}

return true;

}

int FlowWeightDlg::calcMeterError(int idx)

{

m\_meterError[idx] = 100\*(m\_meterEndValue[idx] - m\_meterStartValue[idx] - m\_meterStdValue[idx])/m\_meterStdValue[idx];//计算某个表的误差

ui.tableWidget->setItem(m\_meterPosMap[idx]-1, COLUMN\_ERROR, new QTableWidgetItem(QString::number(m\_meterError[idx], 'f', 4))); //误差

QString meterNoPrefix = getNumPrefixOfManufac(m\_nowParams->m\_manufac);

QString meterNoStr;

strncpy\_s(m\_recPtr[idx].timestamp, m\_timeStamp.toAscii(), TIMESTAMP\_LEN);

m\_recPtr[idx].flowPoint = m\_flowPoint;

meterNoStr = meterNoPrefix + QString("%1").arg(ui.tableWidget->item(m\_meterPosMap[idx]-1, 0)->text(), 8, '0');

strcpy\_s(m\_recPtr[idx].meterNo, meterNoStr.toAscii());

m\_recPtr[idx].flowPointIdx = m\_nowOrder; //

m\_recPtr[idx].methodFlag = WEIGHT\_METHOD; //质量法

m\_recPtr[idx].meterValue0 = m\_meterStartValue[idx];

m\_recPtr[idx].meterValue1 = m\_meterEndValue[idx];

m\_recPtr[idx].balWeight0 = m\_balStartV;

m\_recPtr[idx].balWeight1 = m\_balEndV;

m\_recPtr[idx].pipeTemper = m\_meterTemper[idx];

m\_recPtr[idx].density = m\_meterDensity[idx];

m\_recPtr[idx].stdValue = m\_meterStdValue[idx];

m\_recPtr[idx].dispError = m\_meterError[idx];

m\_recPtr[idx].grade = m\_nowParams->m\_grade;

m\_recPtr[idx].stdError = m\_gradeErr[m\_nowParams->m\_grade]; //二级表 标准误差

m\_recPtr[idx].result = (fabs(m\_recPtr[idx].dispError) <= fabs(m\_recPtr[idx].stdError)) ? 1 : 0;

m\_recPtr[idx].meterPosNo = m\_meterPosMap[idx];

m\_recPtr[idx].standard = m\_standard;

m\_recPtr[idx].model = m\_model;

m\_recPtr[idx].meterType = m\_meterType; //表类型

m\_recPtr[idx].manufactDept = m\_nowParams->m\_manufac;

m\_recPtr[idx].verifyDept = m\_nowParams->m\_vcomp;

m\_recPtr[idx].verifyPerson = m\_nowParams->m\_vperson;

strncpy\_s(m\_recPtr[idx].verifyDate, m\_nowDate.toAscii(), DATE\_LEN);

strncpy\_s(m\_recPtr[idx].validDate, m\_validDate.toAscii(), DATE\_LEN);

return true;

}

int FlowWeightDlg::openValve(UINT8 portno)

{

m\_controlObj->askControlRelay(portno, VALVE\_OPEN);

if (m\_portsetinfo.version==OLD\_CTRL\_VERSION) //老控制板 无反馈

{

slotSetValveBtnStatus(portno, VALVE\_OPEN);

}

return true;

}

int FlowWeightDlg::closeValve(UINT8 portno)

{

m\_controlObj->askControlRelay(portno, VALVE\_CLOSE);

if (m\_portsetinfo.version==OLD\_CTRL\_VERSION) //老控制板 无反馈

{

slotSetValveBtnStatus(portno, VALVE\_CLOSE);

}

return true;

}

int FlowWeightDlg::operateValve(UINT8 portno)

{

if (m\_valveStatus[portno]==VALVE\_OPEN) //阀门原来是打开状态

{

closeValve(portno);

}

else //阀门原来是关闭状态

{

openValve(portno);

}

return true;

}

void FlowWeightDlg::slotSetValveBtnStatus(const UINT8 &portno, const bool &status)

{

m\_valveStatus[portno] = status;

setValveBtnBackColor(m\_valveBtn[portno], m\_valveStatus[portno]);

}

void FlowWeightDlg::slotSetRegulateOk()

{

setRegBtnBackColor(m\_regBtn[m\_nowRegIdx], true);

}

void FlowWeightDlg::slotSetMeterNumber(const QString& comName, const QString& meterNo)

{

int meterPos = m\_readComConfig->getMeterPosByComName(comName);

if (meterPos < 1)

{

return;

}

ui.tableWidget->setItem(meterPos-1, COLUMN\_METER\_NUMBER, new QTableWidgetItem(meterNo.right(8))); //表号

}

void FlowWeightDlg::slotSetMeterFlow(const QString& comName, const float& flow)

{

int meterPos = m\_readComConfig->getMeterPosByComName(comName);

if (meterPos < 1)

{

return;

}

int idx = isMeterPosValid(meterPos);

if (m\_startValueFlag) //初值

{

ui.tableWidget->setItem(meterPos - 1, COLUMN\_METER\_START, new QTableWidgetItem(QString::number(flow)));

if (idx>=0 && m\_meterStartValue!=NULL)

{

m\_meterStartValue[idx] = flow;

}

}

else //终值

{

ui.tableWidget->setItem(meterPos - 1, COLUMN\_METER\_END, new QTableWidgetItem(QString::number(flow)));

if (idx>=0 && m\_meterEndValue!=NULL)

{

m\_meterEndValue[idx] = flow;

}

}

}

void FlowWeightDlg::setValveBtnBackColor(QToolButton \*btn, bool status)

{

if (NULL == btn)

{

return;

}

if (status) //阀门打开 绿色

{

btn->setStyleSheet("background-color:rgb(0,255,0);border:0px;border-radius:10px;");

}

else //阀门关闭 红色

{

btn->setStyleSheet("background-color:rgb(255,0,0);border:0px;border-radius:10px;");

}

}

void FlowWeightDlg::setRegBtnBackColor(QPushButton \*btn, bool status)

{

if (NULL == btn)

{

return;

}

if (status) //调节成功

{

btn->setStyleSheet("background:blue;border:0px;");

}

else //调节失败

{

btn->setStyleSheet("");

}

}

void FlowWeightDlg::on\_btnParaSet\_clicked()

{

if (NULL == m\_paraSetDlg)

{

m\_paraSetDlg = new ParaSetDlg();

}

else

{

delete m\_paraSetDlg;

m\_paraSetDlg = new ParaSetDlg();

}

connect(m\_paraSetDlg, SIGNAL(saveSuccessSignal()), this, SLOT(readNowParaConfig()));

m\_paraSetDlg->show();

}

void FlowWeightDlg::on\_btnWaterIn\_clicked() //进水阀

{

m\_nowPortNo = m\_portsetinfo.waterInNo;

operateValve(m\_nowPortNo);

}

void FlowWeightDlg::on\_btnWaterOut\_clicked() //出水阀

{

m\_nowPortNo = m\_portsetinfo.waterOutNo;

operateValve(m\_nowPortNo);

}

void FlowWeightDlg::on\_btnValveBig\_clicked() //大流量阀

{

m\_nowPortNo = m\_portsetinfo.bigNo;

operateValve(m\_nowPortNo);

}

void FlowWeightDlg::on\_btnValveMiddle1\_clicked() //中流一阀

{

m\_nowPortNo = m\_portsetinfo.middle1No;

operateValve(m\_nowPortNo);

}

void FlowWeightDlg::on\_btnValveMiddle2\_clicked() //中流二阀

{

m\_nowPortNo = m\_portsetinfo.middle2No;

operateValve(m\_nowPortNo);

}

void FlowWeightDlg::on\_btnValveSmall\_clicked() //小流量阀

{

m\_nowPortNo = m\_portsetinfo.smallNo;

operateValve(m\_nowPortNo);

}

void FlowWeightDlg::on\_btnWaterPump\_clicked()

{

m\_nowPortNo = m\_portsetinfo.pumpNo;

m\_controlObj->askControlWaterPump(m\_nowPortNo, !m\_valveStatus[m\_nowPortNo]);

if (m\_portsetinfo.version == OLD\_CTRL\_VERSION) //老控制板 无反馈

{

slotSetValveBtnStatus(m\_nowPortNo, !m\_valveStatus[m\_nowPortNo]);

}

}

void FlowWeightDlg::on\_btnSetFreq\_clicked()

{

m\_controlObj->askSetDriverFreq(ui.spinBoxFreq->value());

}

int FlowWeightDlg::getMeterStartValue()

{

if (m\_autopick) //自动采集

{

m\_startValueFlag = true;

readMeter();

QTest::qWait(2000); //等待串口返回数据

return true;

}

else //手动输入

{

m\_inputStartValue = true; //允许输入初值

QMessageBox::information(this, tr("Hint"), tr("please input init value of heat meter"));

ui.tableWidget->setCurrentCell(m\_meterPosMap[0]-1, COLUMN\_METER\_START); return false;

}

}

void FlowWeightDlg::makeStartValueByLastEndValue()

{

float flow = m\_paraSetReader->getFpBySeq(m\_nowOrder).fp\_verify;

for (int i=0; i<m\_validMeterNum; i++)

{

m\_meterStartValue[i] = m\_meterEndValue[i];

ui.tableWidget->item(m\_meterPosMap[i]-1, COLUMN\_METER\_START)->setText(QString("%1").arg(m\_meterStartValue[i]));

ui.tableWidget->item(m\_meterPosMap[i]-1, COLUMN\_FLOW\_POINT)->setText(QString("%1").arg(flow));

}

}

int FlowWeightDlg::getMeterEndValue()

{

if (m\_autopick) //自动采集

{

m\_startValueFlag = false;

readMeter();

QTest::qWait(2000); //等待串口返回数据

return true;

}

else //手动输入

{

m\_inputEndValue = true; //允许输入终值

QMessageBox::information(this, tr("Hint"), tr("please input end value of heat meter")); ui.tableWidget->setCurrentCell(m\_meterPosMap[0]-1, COLUMN\_METER\_END); return false;

}

}

void FlowWeightDlg::on\_tableWidget\_cellChanged(int row, int column)

{

if (m\_autopick) //自动采集

{

return;

}

if (NULL == ui.tableWidget->item(row, column))

{

return;

}

int meterPos = row + 1; //表位号

int idx = -1;

idx = isMeterPosValid(meterPos);

if (idx < 0)

{

return;

}

bool ok;

if (column==COLUMN\_METER\_START && m\_inputStartValue) //表初值列 且 允许输入初值

{

m\_meterStartValue[idx] = ui.tableWidget->item(row, column)->text().toFloat(&ok);

if (!ok)

{

QMessageBox::warning(this, tr("Warning"), tr("Error: please input digits")); return;

}

if (meterPos == m\_meterPosMap[m\_validMeterNum-1]) //输入最后一个表初值

{

m\_inputStartValue = false;

startVerifyFlowPoint(m\_nowOrder);

}

else //不是最后一个表初值,自动定位到下一个

{

ui.tableWidget->setCurrentCell(m\_meterPosMap[idx+1]-1, column);

}

}

if (column==COLUMN\_METER\_END && m\_inputEndValue) //表终值列 且 允许输入终值

{

m\_meterEndValue[idx] = ui.tableWidget->item(row, column)->text().toFloat(&ok);

if (!ok)

{

QMessageBox::warning(this, tr("Warning"), tr("Error: please input digits")); return;

}

calcMeterError(idx);

insertFlowVerifyRec(&m\_recPtr[idx], 1);

if (meterPos == m\_meterPosMap[m\_validMeterNum-1]) //输入最后一个表终值

{

m\_inputEndValue = false;

if (m\_autopick) //自动采集

{

ui.btnNext->hide();

}

else if ( !m\_autopick && (m\_nowOrder != m\_flowPointNum) )

{

ui.btnNext->show();

}

}

else //不是最后一个表终值,自动定位到下一个

{

ui.tableWidget->setCurrentCell(m\_meterPosMap[idx+1]-1, column);

}

}

}

int FlowWeightDlg::isMeterPosValid(int meterPos)

{

for (int i=0; i<m\_validMeterNum; i++)

{

if (m\_meterPosMap[i] == meterPos)

{

return i;

}

}

return -1;

}

int FlowWeightDlg::saveAllVerifyRecords()

{

insertFlowVerifyRec(m\_recPtr, m\_validMeterNum);

return true;

}

void FlowWeightDlg::on\_btnReadMeter\_clicked()

{

for (int j=0; j<m\_maxMeterNum; j++)

{

m\_meterObj[j].setProtocolVersion(m\_manufac); //设置热量表厂家

m\_meterObj[j].askReadMeter();

}

}

void FlowWeightDlg::slotModifyMeterNo(const int &row)

{

qDebug()<<"row ="<<row;

m\_meterObj[row].askModifyMeterNo("12345678",ui.tableWidget->item(row, COLUMN\_METER\_NUMBER)->text());

}

void FlowWeightDlg::slotAdjustError(const int &row)

{

qDebug()<<"adj row ="<<row;

}

void FlowWeightDlg::on\_btnExit\_clicked()

{

this->close();

}

#include <QtGui/QMessageBox>

#include <QtCore/QDebug>

#include <QFile>

#include <QtXml/QtXml>

#include <iostream>

#include "setcomfrm.h"

#include "commondefine.h"

#include "readcomconfig.h"

SetComFrm::SetComFrm(QWidget \*parent, Qt::WFlags flags)

: QWidget(parent, flags)

{

gui.setupUi(this);

m\_config = new ReadComConfig();

InstallConfigs();

QString runhome = QProcessEnvironment::systemEnvironment().value("RUNHOME");

#ifdef Q\_OS\_LINUX

ConfigFileName = runhome + "\/ini\/comconfig.xml";

#elif defined (Q\_OS\_WIN)

ConfigFileName = runhome + "\\ini\\comconfig.xml";

#endif

}

SetComFrm::~SetComFrm()

{

if (m\_config)

{

delete m\_config;

m\_config=NULL;

}

}

void SetComFrm::on\_btnExit\_clicked()

{

close();

}

void SetComFrm::on\_btnSave\_clicked()

{

WriteValveConfig(ReadValeSet());

WriteBalanceConfig(ReadBalanceSet());

WriteTempConfig(ReadTempSet());

WriteStdTempConfig(ReadStdTempSet());

WriteMetersConfig();

QMessageBox::information(this, tr("OK"), tr("Save Settings Successfully !"));

}

void SetComFrm::InstallConfigs()

{

InstallValeConfig();

InstallBalanceConfig();

InstallTempConfig();

InstallStdtmpConfig();

InstallMetersConfig();

}

void SetComFrm::InstallValeConfig()

{

QStringList valve\_index = m\_config->ReadIndexByName("valve");

gui.comboValveSerialNum->setCurrentIndex(valve\_index[0].toInt());

gui.comboValveBaudRate->setCurrentIndex(valve\_index[1].toInt());

gui.comboValveBits->setCurrentIndex(valve\_index[2].toInt());

gui.comboValveChkBit->setCurrentIndex(valve\_index[3].toInt());

gui.comboValveEndBit->setCurrentIndex(valve\_index[4].toInt());

}

void SetComFrm::InstallBalanceConfig()

{

QStringList valve\_index = m\_config->ReadIndexByName("balance");

gui.comboBalSerialNum->setCurrentIndex(valve\_index[0].toInt());

gui.comboBalBaudRate->setCurrentIndex(valve\_index[1].toInt());

gui.comboBalBits->setCurrentIndex(valve\_index[2].toInt());

gui.comboBalChkBit->setCurrentIndex(valve\_index[3].toInt());

gui.comboBalEndBit->setCurrentIndex(valve\_index[4].toInt());

}

void SetComFrm::InstallTempConfig()

{

QStringList valve\_index = m\_config->ReadIndexByName("temp");

gui.comboTempSerialNum->setCurrentIndex(valve\_index[0].toInt());

gui.comboTempBaudRate->setCurrentIndex(valve\_index[1].toInt());

gui.comboTempBits->setCurrentIndex(valve\_index[2].toInt());

gui.comboTempChkBit->setCurrentIndex(valve\_index[3].toInt());

gui.comboTempEndBit->setCurrentIndex(valve\_index[4].toInt());

}

void SetComFrm::InstallStdtmpConfig()

{

QStringList valve\_index = m\_config->ReadIndexByName("stdtemp");

gui.comboStdTmpSerialNum->setCurrentIndex(valve\_index[0].toInt());

gui.comboStdTmpBaudRate->setCurrentIndex(valve\_index[1].toInt());

gui.comboStdTmpBits->setCurrentIndex(valve\_index[2].toInt());

gui.comboStdTmpChkBit->setCurrentIndex(valve\_index[3].toInt());

gui.comboStdTmpEndBit->setCurrentIndex(valve\_index[4].toInt());

}

void SetComFrm::InstallMetersConfig()

{

const QObjectList list=gui.gBoxMeters->children();

foreach(QObject \*obj, list)

{

QString class\_name = QString::fromAscii( obj->metaObject()->className() );

if (class\_name == "QGroupBox")

{

InstallMeterConfigByNum((QGroupBox\*)obj);

}

}

}

void SetComFrm::InstallMeterConfigByNum(QGroupBox \*gBox)

{

const QObjectList list=gBox->children();

QString config\_id = "meter" + gBox->objectName().split("\_")[1];

QStringList configs = m\_config->ReadIndexByName(config\_id);

foreach(QObject\* obj, list)

{

QString class\_name = QString::fromAscii( obj->metaObject()->className() );

if(class\_name=="QComboBox")

{

QComboBox \*CBox=(QComboBox\*)obj;

QString object\_name = CBox->objectName();

if (object\_name .contains("SerialNum",Qt::CaseSensitive))

{

CBox->setCurrentIndex(configs[0].toInt());

}

else if (object\_name .contains("BaudRate",Qt::CaseSensitive))

{

CBox->setCurrentIndex(configs[1].toInt());

}

else if (object\_name .contains("Bits",Qt::CaseSensitive))

{

CBox->setCurrentIndex(configs[2].toInt());

}

else if (object\_name .contains("ChkBit",Qt::CaseSensitive))

{

CBox->setCurrentIndex(configs[3].toInt());

}

else if (object\_name .contains("EndBit",Qt::CaseSensitive))

{

CBox->setCurrentIndex(configs[4].toInt());

}

}

}

}

QVector<QString> SetComFrm::ReadValeSet()

{

QVector<QString> Configs = ReadGBoxSet(gui.gBoxValve);

return Configs;

}

QVector<QString> SetComFrm::ReadBalanceSet()

{

QVector<QString> Configs = ReadGBoxSet(gui.gBoxBalance);

return Configs;

}

QVector<QString> SetComFrm::ReadTempSet()

{

QVector<QString> Configs = ReadGBoxSet(gui.gBoxTempSenor);

return Configs;

}

QVector<QString> SetComFrm::ReadStdTempSet()

{

QVector<QString> Configs = ReadGBoxSet(gui.gBoxStdTmpSensor);

return Configs;

}

QVector<QString> SetComFrm::ReadMeterSetByNum(QString MeterNum)

{

QVector<QString> meter\_configs;

const QObjectList list=gui.gBoxMeters->children();

foreach(QObject \*obj, list)

{

QString class\_name = QString::fromAscii( obj->metaObject()->className() );

if (class\_name == "QGroupBox")

{

QGroupBox \*meter\_gbox = (QGroupBox\*)obj;

QString box\_name = meter\_gbox->objectName();

QStringList words = box\_name.split("\_");//获取控件名的后缀

if (words[1] == MeterNum)

{

meter\_configs = ReadGBoxSet(meter\_gbox);

break;

}

}

}

return meter\_configs;

}

QVector<QString> SetComFrm::ReadGBoxSet(QGroupBox \*gBox)

{

QString com\_num ;

QString baud\_rate ;

QString bits ;

QString chk\_bit ;

QString end\_bit ;

const QString sep="#SEP#";//分隔符，用于分隔界面值和索引值

const QObjectList list=gBox->children();

foreach(QObject\* obj, list)

{

QString class\_name = QString::fromAscii( obj->metaObject()->className() );

if(class\_name=="QComboBox")

{

QComboBox \*CBox=(QComboBox\*)obj;

QString object\_name = CBox->objectName();

if (bool occur = object\_name .contains("SerialNum",Qt::CaseSensitive))

{

com\_num = CBox->currentText() + sep + QString::number(CBox->currentIndex(), 10);

}

else if (occur = object\_name .contains("BaudRate",Qt::CaseSensitive))

{

baud\_rate = CBox->currentText()+ sep + QString::number(CBox->currentIndex(), 10);

}

else if (occur = object\_name .contains("Bits",Qt::CaseSensitive))

{

bits = CBox->currentText()+ sep + QString::number(CBox->currentIndex(), 10);

}

else if (occur = object\_name .contains("ChkBit",Qt::CaseSensitive))

{

chk\_bit = QString::number(CBox->currentIndex(), 10)+ sep + QString::number(CBox->currentIndex(), 10);

}

else if (occur = object\_name .contains("EndBit",Qt::CaseSensitive))

{

end\_bit = QString::number(CBox->currentIndex(), 10)+ sep + QString::number(CBox->currentIndex(), 10);

}

}

}

QVector<QString> strArray;

strArray.append(com\_num);

strArray.append(baud\_rate);

strArray.append(bits);

strArray.append(chk\_bit);

strArray.append(end\_bit);

return strArray;

}

bool SetComFrm::WriteValveConfig(QVector<QString> ValveConfigs)

{

return WriteConfigById("valve", ValveConfigs);

}

bool SetComFrm::WriteBalanceConfig(QVector<QString> BalanceConfigs)

{

return WriteConfigById("balance", BalanceConfigs);

}

bool SetComFrm::WriteTempConfig(QVector<QString> TempConfigs)

{

return WriteConfigById("temp", TempConfigs);

}

bool SetComFrm::WriteStdTempConfig(QVector<QString> StdTempConfigs)

{

return WriteConfigById("stdtemp", StdTempConfigs);

}

bool SetComFrm::WriteMetersConfig()

{

QVector<QString> Configs;//相应表号的界面配置

QString meter\_num;//表号

for (int i=1; i< 13; i++)

{

meter\_num = QString::number(i, 10);

Configs = ReadMeterSetByNum(meter\_num);

if (!WriteMeterConfigByNum(meter\_num, Configs))

{

QMessageBox::warning(this, tr("Warning"), tr("Save Settings of Meter: ") + meter\_num + tr("# Error!"));

return false;

}

}

return true;

}

bool SetComFrm::WriteMeterConfigByNum(QString MeterNum, QVector<QString> MeterConfigs)

{

QString ConfigId = "meter" + MeterNum;

return WriteConfigById(ConfigId, MeterConfigs);

}

bool SetComFrm::WriteConfigById(QString ConfigId, QVector<QString> Configs)

{

if (!OpenConfigFile())

return false;

QDomElement root = m\_doc.documentElement();

if(root.tagName()!= "configs")

return false;

QDomNode n;

if (ConfigId.contains("meter"))

n = root.lastChild().firstChild();

else

n = root.firstChild();

while ( !n.isNull() )

{

QDomElement e = n.toElement();

if( !e.isNull())

{

if( e.tagName() == "config" && e.attribute("id") == ConfigId)//查找id号

{

QDomNodeList list = e.childNodes();

for(int i=0; i<list.count(); i++) //遍历该列表

{

QDomNode node = list.at(i);

node.firstChild().setNodeValue(Configs[i]);

}

}

}

n = n.nextSibling();

}

return WriteConfigFile();

}

bool SetComFrm::OpenConfigFile()

{

QFile file( ConfigFileName );

if( !file.open( QFile::ReadOnly | QFile::Text ) )

{

QMessageBox::critical(this, tr("Error"), tr("Can not open file: ") + ConfigFileName, QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);

return false;

}

if( !m\_doc.setContent( &file ) )

{

QMessageBox::critical(this, tr("Error"), tr("Can not set Content file: ") + ConfigFileName, QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);

file.close();

return false;

}

file.close();

return true;

}

bool SetComFrm::WriteConfigFile()

{

QFile filexml(ConfigFileName);

if( !filexml.open( QFile::WriteOnly | QFile::Truncate) )

{

QMessageBox::critical(this, tr("Error"), tr("Can not write to file: ") + ConfigFileName, QMessageBox::Yes | QMessageBox::No, QMessageBox::Yes);

return false;

}

QTextStream ts(&filexml);

ts.reset();

ts.setCodec("utf-8");

m\_doc.save(ts, 4, QDomNode::EncodingFromTextStream);

filexml.close();

return true;

}