//main.cpp

#include "widget.h"

#include <QApplication>

#include "maincontrol.h"

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

{

QApplication a(argc, argv);

MainControl app;

app.init();

return a.exec();

}

//maincontrol.h

#ifndef MAINCONTROL\_H

#define MAINCONTROL\_H

#include <QObject>

#include "communication/network.h"

#include "communication/commanage.h"

#include "ui/positionui.h"

#include "ui/uimanage.h"

class **MainControl** : public QObject

{

Q\_OBJECT

public:

explicit **MainControl**(QObject \*parent = nullptr);

~***MainControl***();

void **init**();

private:

ComManage \* m\_com;

UIManage \* m\_ui;

};

#endif // MAINCONTROL\_H

//maincontrol.cpp

#include "maincontrol.h"

MainControl::**MainControl**(QObject \*parent) : QObject(parent)

{

m\_com = new ComManage;

m\_ui = new UIManage;

// 连接状态

connect(m\_com,SIGNAL(sglTCPStatus(bool)),m\_ui,SLOT(slotTCPStatus(bool)));

connect(m\_com,SIGNAL(sglSerialStatus(bool)),m\_ui,SLOT(slotSerialStatus(bool)));

// 提示信息

connect(m\_com,SIGNAL(sglHintInfo(QString &)),m\_ui,SLOT(slotHintInfo(QString &)));

// 侦察标志

connect(m\_com,SIGNAL(sglScoutFlag()),m\_ui,SLOT(slotScoutFlag()));

// 串口名称

connect(m\_com,SIGNAL(sglSerialName(QStringList &)),m\_ui,SLOT(slotSerialName(QStringList &)));

// 位置信息

connect(m\_com,SIGNAL(sglPosInfo(STRUCT\_POS&)),m\_ui,SLOT(slotPosInfo(STRUCT\_POS&)));

// 通信连接信息

connect(m\_ui,SIGNAL(sglCreateTCP(QString &,quint16)),m\_com,SLOT(slotCreateTCP(QString &,quint16)));

connect(m\_ui,SIGNAL(sglCloseTCP()),m\_com,SLOT(slotCloseTCP()));

connect(m\_ui,SIGNAL(sglCreateSerial(QString&,quint32)),m\_com,SLOT(slotCreateSerial(QString&,quint32)));

connect(m\_ui,SIGNAL(sglCloseSerial()),m\_com,SLOT(slotCloseSerial()));

// 发送数据

connect(m\_ui,SIGNAL(sglSendData(STRUCT\_TARGET\_INFO &)),m\_com,SLOT(slotSendPosIno(STRUCT\_TARGET\_INFO &)));

}

MainControl::~***MainControl***()

{

if(nullptr != m\_com)

{

delete m\_com;

m\_com = nullptr;

}

if(nullptr != m\_ui)

{

delete m\_ui;

m\_ui = nullptr;

}

}

void MainControl::**init**()

{

m\_com->init();

m\_ui->showUI();

}

//public.h

#ifndef PUBLIC\_H

#define PUBLIC\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 接收信息：下发的车控信息 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SCOUT\_TASK\_INFO 0x10070 // 侦察任务信息

#define NAVIGATE\_INFO 0x10060 // 导航信息

#define LINK\_TEST\_INSTRUCT 0x10010 // 链路测试指令

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 发送信息：上报的火控信息 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define TARGET\_INFO 0x40020 // 战场目标信息

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 结构体定义 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//公有

// 报文头信息

typedef struct **\_sMsgHeadInfo**

{

unsigned int uMsgHead; // 标志头

unsigned int uLen; // 数据长度，从CRC开始至数据结束

unsigned int uCRC; // 校验和，从OPC开始计算，至数据结束

unsigned int uSNo; // 帧序号

unsigned short usOPC; // 操作码

unsigned short usATTR; // 属性

unsigned int uMsgID; // 消息编号

**\_sMsgHeadInfo**()

{

clear();

}

void **clear**()

{

uMsgHead = 0; // 标志头

uLen = 0; // 数据长度，从CRC开始至数据结束

uCRC = 0; // 校验和，从OPC开始计算，至数据结束

uSNo = 0; // 帧序号

usOPC = 0; // 操作码

usATTR = 0; // 属性

uMsgID = 0; // 消息编号

}

}STRUCT\_MSG\_HEAD\_INFO;

//位置信息

typedef struct **\_sPositionInfo**

{

double dLongitude; // 经度

double dLatitude; // 纬度

unsigned short usAltFlag; // 高度有效标志

unsigned short usAltitude; // 高度

**\_sPositionInfo**()

{

clear();

}

void **clear**()

{

dLongitude = 0.0;

dLatitude = 0.0;

usAltitude = 0;

usAltFlag = 0;

}

}STRUCT\_POSITION\_INFO;

// 时间

typedef struct **\_sTimeInfo**

{

unsigned short usYear;

unsigned short usMonth;

unsigned short usDay;

unsigned short usHour;

unsigned short usMinute;

unsigned short usSecond;

**\_sTimeInfo**()

{

clear();

}

void **clear**()

{

usYear = 0;

usMonth = 0;

usDay = 0;

usHour = 0;

usMinute = 0;

usSecond = 0;

}

}STRUCT\_TIME\_INFO;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//战场目标信息(上报)

typedef struct **\_sTargetInfo**

{

unsigned int uVMFCode; // 车辆的VMF编码值

unsigned long long ulTargetID; // 目标编号

STRUCT\_POSITION\_INFO sPosInfo; // 目标位置信息

unsigned short usTargetAttri; // 目标属性

unsigned short usTargetType; // 目标分类

STRUCT\_TIME\_INFO sFindTime; // 发现时间

unsigned short usFindMethod; // 发现手段

unsigned short usReliability; // 发现可靠性

unsigned short usReserve0; // 保留字段

unsigned short usTagMove; // 是否运动标识

float fMoveSpeed; // 目标运动速度

unsigned short usMoveOrientation; // 目标运动方向

unsigned short usReserve; // 保留字段

**\_sTargetInfo**()

{

clear();

}

void **clear**()

{

uVMFCode = 3; // VMF编码值

ulTargetID = 0; // 目标ID

sPosInfo.clear(); // 目标位置信息

usTargetAttri = 0; // 目标属性

usTargetType = 0; // 目标分类

sFindTime.clear(); // 发现时间

usFindMethod = 7; // 发现手段,无人机

usReliability = 0; // 发现可靠性，非常可靠

usReserve0 = 0; // 保留字段

usTagMove = 0; // 是否运动标识

fMoveSpeed = 0.0; // 目标运动速度

usMoveOrientation = 0; // 目标运动方向

usReserve = 0; // 保留字段

}

}STRUCT\_TARGET\_INFO;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//侦查要求信息（下发）

typedef struct **\_sInvestigationInfo**

{

unsigned long long ulVMFCode; // 车辆的VMF编码值

unsigned short usDevType; // 设备类型

unsigned short usUrgencyDegree; // 紧急程度

STRUCT\_TIME\_INFO sStartTime; // 开始时间

STRUCT\_TIME\_INFO sEndTime; // 结束时间

unsigned short usReserve; // 保留字段

unsigned short usTagTarget; // 侦察目标有效标识

unsigned long long ulTargetID; // 侦查目标ID

STRUCT\_POSITION\_INFO sTargetLocation; // 侦查目标位置信息

unsigned short usTagArea; // 侦察区域有效标识

unsigned short usVertexNum; // 多边形顶点数量

STRUCT\_POSITION\_INFO\* pPosInfo; // 多边形顶点位置信息

**\_sInvestigationInfo**()

{

clear();

}

void **clear**()

{

ulVMFCode = 0; // 车辆的VMF编码值

usDevType = 1; // 设备类型,无人机

usUrgencyDegree = 0; // 紧急程度,特急

sStartTime.clear(); // 开始时间

sEndTime.clear(); // 结束时间

usReserve = 0; // 保留字段

usTagTarget = 0; // 侦察目标有效标识,无效

ulTargetID = 0; // 侦查目标ID

sTargetLocation.clear(); // 侦查目标位置信息

usTagArea = 0; // 侦察区域有效标识,无效

usVertexNum = 0; // 多边形顶点数量

pPosInfo = nullptr; // 多边形顶点位置信息

}

}STRUCT\_INVESTIGATION\_INFO;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 与零度之间的协议 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct **\_sPos**

{

float fLongitude; // 经度

float fLatitude; // 纬度

unsigned short usAltitude; // 高度

unsigned short usDistance; // 目标与吊舱之间的距离

**\_sPos**()

{

clear();

}

void **clear**()

{

fLongitude = 0.0;

fLatitude = 0.0;

usAltitude = 0;

usDistance = 0;

}

}STRUCT\_POS;

#endif // PUBLIC\_H

//serialcom.h

#ifndef SERIALCOM\_H

#define SERIALCOM\_H

#include <QObject>

#include <QSerialPortInfo>

#include <QSerialPort>

#include <QString>

#include <QStringList>

#include <QByteArray>

class **SerialCom** : public QObject

{

Q\_OBJECT

public:

explicit **SerialCom**(QObject \*parent = nullptr);

public:

void **getName**(QStringList &listName);

bool **createLink**(QString & portName, qint32 nBaudRate);

void **closeLink**();

void **sendData**(QByteArray &data);

QByteArray **recvData**();

signals:

void **sglRecvData**();

private:

QSerialPort \*m\_serial;

};

#endif // SERIALCOM\_H

//serialcom.cpp

#include "serialcom.h"

SerialCom::**SerialCom**(QObject \*parent) : QObject(parent)

{

m\_serial = new QSerialPort;

}

void SerialCom::**getName**(QStringList &listName)

{

listName.clear();

QList<QSerialPortInfo> listInfo = QSerialPortInfo::availablePorts();

for(int i = 0; i < listInfo.size(); i++)

{

listName.append(listInfo.at(i).portName());

}

}

bool SerialCom::**createLink**(QString & portName, qint32 nBaudRate)

{

if(m\_serial == nullptr)

return false;

if(m\_serial->isOpen())

m\_serial->*close*();

//设置串口名

m\_serial->setPortName(portName);

//设置波特率

m\_serial->setBaudRate(nBaudRate);

//设置数据位数

m\_serial->setDataBits(QSerialPort::Data8);

//设置奇偶校验

m\_serial->setParity(QSerialPort::NoParity);

//设置停止位

m\_serial->setStopBits(QSerialPort::OneStop);

//设置流控制

m\_serial->setFlowControl(QSerialPort::NoFlowControl);

//打开串口

bool isOpen = m\_serial->*open*(QIODevice::ReadWrite);

if(isOpen)

connect(m\_serial,&QSerialPort::readyRead,this,&SerialCom::sglRecvData);

else

return false;

return true;

}

void SerialCom::**closeLink**()

{

if(m\_serial->isOpen())

m\_serial->*close*();

}

void SerialCom::**sendData**(QByteArray &data)

{

m\_serial->write(data);

}

QByteArray SerialCom::**recvData**()

{

return m\_serial->readAll();

}

//network.h

#ifndef NETWORK\_H

#define NETWORK\_H

#include <QObject>

#include <QTcpSocket>

#include <QString>

#include <QByteArray>

#include "public.h"

#include "dataDeal/datapack.h"

class **NetWork** : public QObject

{

Q\_OBJECT

public:

explicit **NetWork**(QObject \*parent = nullptr);

~***NetWork***();

/\*\*\*\*\*\*\*\*\*\* 外部接口 \*\*\*\*\*\*\*\*\*\*\*\*/

public:

void **closeLink**();

bool **createLink**(QString &addr, quint16 nPort);

QByteArray **recvData**();

bool **sendData**(QByteArray &data);

// 信号与槽

signals:

// to manage

void **sglRecvData**();

private:

QTcpSocket \*m\_socket;

};

#endif // NETWORK\_H

//network.cpp

#include "network.h"

#include <QDebug>

#include <QDateTime>

NetWork::**NetWork**(QObject \*parent) : QObject(parent)

{

m\_socket = new QTcpSocket;

}

NetWork::~***NetWork***()

{

if(m\_socket->isOpen())

m\_socket->*close*();

delete m\_socket;

m\_socket = nullptr;

}

void NetWork::**closeLink**()

{

if(m\_socket != nullptr && m\_socket->isOpen())

{

m\_socket->*close*();

}

}

bool NetWork::**createLink**(QString &addr,quint16 nPort)

{

if(nullptr != m\_socket)

{

if(m\_socket->isOpen())

m\_socket->*close*();

m\_socket->bind(28004);

m\_socket->*connectToHost*(addr,nPort);

bool isConnect = m\_socket->*waitForConnected*();

if(isConnect)

{

connect(m\_socket,&QTcpSocket::readyRead,this,&NetWork::sglRecvData);

return true;

}

}

return false;

}

QByteArray NetWork::**recvData**()

{

return m\_socket->readAll();

}

bool NetWork::**sendData**(QByteArray &data)

{

if(nullptr != m\_socket)

{

qint64 len = m\_socket->write(data);

bool isFlush = m\_socket->flush();

if(isFlush && len == data.size())

return true;

}

return false;

}

//commanage.h

#ifndef COMMANAGE\_H

#define COMMANAGE\_H

#include <QObject>

#include <QDateTime>

#include "dataDeal/datapack.h"

#include "dataDeal/dataparse.h"

#include "network.h"

#include "serialcom.h"

#include "public.h"

class **ComManage** : public QObject

{

Q\_OBJECT

public:

explicit **ComManage**(QObject \*parent = nullptr);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 成员函数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public:

void **init**();

private:

// 处理接收数据（总）

void **dealRecvData**(QByteArray &data);

// 处理串口接收数据

void **dealSerialData**(QByteArray &data);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 信号与槽接口 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

signals:

// to UI

void **sglTCPStatus**(bool bStatus);

void **sglSerialStatus**(bool bStatus);

void **sglHintInfo**(QString &info);

void **sglScoutFlag**();

void **sglSerialName**(QStringList & listName);

void **sglPosInfo**(STRUCT\_POS &info);

public slots:

// tcp

void **slotCreateTCP**(QString &addr, quint16 nPort);

void **slotCloseTCP**();

void **slotSendPosIno**(STRUCT\_TARGET\_INFO &info);

// serial

void **slotCreateSerial**(QString &name, quint32 baud);

void **slotCloseSerial**();

private slots:

void **recvTCP**();

void **recvSerial**();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 成员变量 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private:

NetWork \* m\_201; // 201通信接口

SerialCom \*m\_serial; // 零度通信接口

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 网络信息 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 侦察目标编号

unsigned long long m\_TargetID;

/\*\*\*\*\*\*\*\*\*\*\*\* 串口信息 \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//位置信息

STRUCT\_POS m\_sPosInfo;

// 缓存

QByteArray m\_dataBuf;

int m\_nDataIndex;

};

#endif // COMMANAGE\_H

//commanage.cpp

#include "commanage.h"

ComManage::**ComManage**(QObject \*parent) : QObject(parent)

{

m\_201 = new NetWork;

m\_serial = new SerialCom;

m\_TargetID = 1;

m\_dataBuf.resize(20);

m\_nDataIndex = 0;

connect(m\_201,&NetWork::sglRecvData,this,&ComManage::recvTCP);

connect(m\_serial,&SerialCom::sglRecvData,this,&ComManage::recvSerial);

}

void ComManage::**init**()

{

QStringList name;

m\_serial->getName(*name*);

emit sglSerialName(*name*);

}

void ComManage::**slotCloseTCP**()

{

m\_201->closeLink();

// status

emit sglTCPStatus(false);

QString netInfo = "TCP Close Success .";

emit sglHintInfo(*netInfo*);

}

void ComManage::ComManage::**slotCreateTCP**(QString &addr, quint16 nPort)

{

QString netInfo;

bool isLink = m\_201->createLink(*addr*,nPort);

if(isLink)

netInfo = "TCP Link Success .";

else

netInfo = "TCP Link Failed .";

emit sglTCPStatus(isLink);

emit sglHintInfo(*netInfo*);

}

void ComManage::**slotSendPosIno**(STRUCT\_TARGET\_INFO &info)

{

//自定义

info.ulTargetID = m\_TargetID++;//侦察目标ID

//位置信息，从零度获取

// info.sPosInfo.dLongitude = m\_sPosInfo.fLongitude;

// info.sPosInfo.dLatitude = m\_sPosInfo.fLatitude;

// info.sPosInfo.usAltitude = m\_sPosInfo.usAltitude;

// info.sPosInfo.usAltFlag = 1;

//当前时间

QDateTime curDateTime = QDateTime::currentDateTime();

info.sFindTime.usYear = curDateTime.date().year();

info.sFindTime.usMonth = curDateTime.date().month();

info.sFindTime.usDay = curDateTime.date().day();

info.sFindTime.usHour = curDateTime.time().hour();

info.sFindTime.usMinute = curDateTime.time().minute();

info.sFindTime.usSecond = curDateTime.time().second();

QByteArray data;

DataPack::getInstance()->packTargetInfo(*data*,*info*);

// 发送数据

if(!m\_201->sendData(*data*))

{

QString netInfo = "TCP Send Data Failed .";

emit sglTCPStatus(false);

emit sglHintInfo(*netInfo*);

}

}

void ComManage::**recvTCP**()

{

QByteArray data = m\_201->recvData();

if(data.size() >= 24)

dealRecvData(*data*);

}

/\*\*\*\*\*\*\*\*\* private \*\*\*\*\*\*\*\*\*\*\*\*\*/

void ComManage::**dealRecvData**(QByteArray &data)

{

// 解析网络数据

STRUCT\_MSG\_HEAD\_INFO info;

int num = DataParse::getInstance()->parseMsgHeadData(data,*info*);

if(num != 0)

{

QString statusInfo = QString("TCP Recv Data Error , ErrorCode is %1").arg(num);

emit sglHintInfo(*statusInfo*);

}

switch (info.uMsgID)

{

case LINK\_TEST\_INSTRUCT:

case NAVIGATE\_INFO:

break;

case SCOUT\_TASK\_INFO:

{

emit sglScoutFlag();

break;

}

default://无效报文，不做处理

{

QString statusInfo = QString("Don't deal data ,msgId is 0x%1").arg(QString::number(info.uMsgID,16));

emit sglHintInfo(*statusInfo*);

return;

}

}

// 回复ACK

DataPack::getInstance()->packACK(*data*,info.uSNo,0x80,info.uMsgID,info.usOPC);

// 发送数据

if(!m\_201->sendData(*data*))

{

QString netInfo = "TCP Send Data Failed .";

emit sglTCPStatus(false);

emit sglHintInfo(*netInfo*);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 串口 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void ComManage::**slotCreateSerial**(QString &name, quint32 baud)

{

QString netInfo;

bool isLink = m\_serial->createLink(*name*,baud);

if(isLink)

netInfo = "Serial Link Success .";

else

netInfo = "Serial Link Failed .";

emit sglSerialStatus(isLink);

emit sglHintInfo(*netInfo*);

}

void ComManage::**slotCloseSerial**()

{

m\_serial->closeLink();

// status

emit sglSerialStatus(false);

QString netInfo = "Serial Close Success .";

emit sglHintInfo(*netInfo*);

}

void ComManage::**recvSerial**()

{

QByteArray data = m\_serial->recvData();

int nSize = data.size();

unsigned char \*buf = (unsigned char \*)data.data();

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

{

switch (m\_nDataIndex)

{

case 0:

{

if('$' == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 1;

}

break;

}

case 1:

{

if('C' == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 2;

}

else

{

m\_nDataIndex = 0;

}

break;

}

case 2:

{

if('O' == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 3;

}

else

{

m\_nDataIndex = 0;

}

break;

}

case 3:

{

if('M' == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 4;

}

else

{

m\_nDataIndex = 0;

}

break;

}

case 4:

{

if('M' == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 5;

}

else

{

m\_nDataIndex = 0;

}

break;

}

case 5:

{

if(0x14 == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 6;

}

else

{

m\_nDataIndex = 0;

}

break;

}

case 6:

{

if(0xd8 == buf[i])

{

m\_dataBuf[m\_nDataIndex] = data[i];

m\_nDataIndex = 7;

}

else

{

m\_nDataIndex = 0;

}

break;

}

default:

{

if(m\_nDataIndex >= 7 && m\_nDataIndex <= 20)

{

if(i + 20 - m\_nDataIndex <= nSize)

{

memcpy(m\_dataBuf.data()+m\_nDataIndex,buf+i,20-m\_nDataIndex);

dealSerialData(*m\_dataBuf*);

i = i + 19 - m\_nDataIndex;

m\_nDataIndex = 0;

memset(m\_dataBuf.data(),0,20);

}

else

{

memcpy(m\_dataBuf.data()+m\_nDataIndex,buf+i,nSize - i);

m\_nDataIndex += (nSize - i);

i = nSize - 1;

}

}

break;

}

}// end switch

}//end for

}

void ComManage::**dealSerialData**(QByteArray &data)

{

bool isTrue = DataParse::getInstance()->parseSerialData(*data*,*m\_sPosInfo*);

if(!isTrue)

{

qDebug() << "------ Don't deal data ,data is " << data.toHex() << QTime::currentTime().msec();

return;

}

// qDebug() << "pos info is " << m\_sPosInfo.fLongitude << m\_sPosInfo.fLatitude << m\_sPosInfo.usAltitude << QTime::currentTime().second() << QTime::currentTime().msec();

emit sglPosInfo(*m\_sPosInfo*);

}

//dataparse.h

#ifndef DATAPARSE\_H

#define DATAPARSE\_H

#include <QObject>

#include "public.h"

#include <QByteArray>

#include <QDebug>

extern void **reverse**(char \* data,int nSize);

class **DataParse** : public QObject

{

Q\_OBJECT

public:

static DataParse \***getInstance**();

private:

explicit **DataParse**(QObject \*parent = nullptr);

static DataParse \*m\_dataParse;

/\*\*\*\*\*\*\*\* 解析串口数据 \*\*\*\*\*\*\*\*\*\*/

public:

bool **parseSerialData**(QByteArray &data, STRUCT\_POS &info);

/\*\*\*\*\*\*\*\* 解析网络数据 \*\*\*\*\*\*\*\*\*\*/

public:

int **parseMsgHeadData**(const QByteArray &data, STRUCT\_MSG\_HEAD\_INFO &info);

};

#endif // DATAPARSE\_H

//dataparse.cpp

#include "dataparse.h"

DataParse \* DataParse::m\_dataParse = nullptr;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*共有接口

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* public(外部接口) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DataParse \*DataParse::**getInstance**()

{

if(nullptr == m\_dataParse)

m\_dataParse = new DataParse;

return m\_dataParse;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* private \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DataParse::**DataParse**(QObject \*parent) : QObject(parent)

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 解析网络数据 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 解析报文头数据

int DataParse::**parseMsgHeadData**(const QByteArray &data,STRUCT\_MSG\_HEAD\_INFO &info)

{

unsigned int uTemp = 0;

unsigned short usTemp = 0;

unsigned char \*pData = (unsigned char\*)data.data();

int nSize = data.size();

// 总长度

if(nSize < 24)

return 1;

// 标志头

memcpy(&uTemp,pData,sizeof (uTemp));

reverse((char \*)(&uTemp),sizeof (uTemp));

if(uTemp != 0xDDCCBBAA)

{

info.clear();

return 2;

}

info.uMsgHead = uTemp;

// 数据长度

memcpy(&uTemp,pData+4,sizeof (uTemp));

reverse((char \*)(&uTemp),sizeof (uTemp));

if(uTemp != (nSize - 8))

{

info.clear();

return 3;

}

info.uLen = uTemp;

// CRC

unsigned short usCheckSum = 0;

unsigned int uCheckSum = 0;

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

{

usCheckSum += pData[i];

}

usCheckSum = ~(usCheckSum & 0xFFFF) + 1;

uCheckSum = (usCheckSum & 0xFFFF);

memcpy(&uTemp,pData+8,sizeof (uTemp));

reverse((char \*)(&uTemp),sizeof (uTemp));

if(uTemp != uCheckSum)

{

info.clear();

return 4;

}

info.uCRC = uTemp;

// 帧序号

memcpy(&uTemp,pData+12,sizeof (uTemp));

reverse((char \*)(&uTemp),sizeof (uTemp));

info.uSNo = uTemp;

// OPC

memcpy(&usTemp,pData+16,sizeof (usTemp));

reverse((char \*)(&usTemp),sizeof (usTemp));

info.usOPC = usTemp;

// ATTR

memcpy(&usTemp,pData+18,sizeof (usTemp));

reverse((char \*)(&usTemp),sizeof (usTemp));

info.usATTR = usTemp;

// MsgID

memcpy(&uTemp,pData+20,sizeof (uTemp));

reverse((char \*)(&uTemp),sizeof (uTemp));

info.uMsgID = uTemp;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*解析串口数据

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool DataParse::**parseSerialData**(QByteArray &data, STRUCT\_POS &info)

{

// 获取位置信息

int nIndex = 0;

int nSize = data.size();

unsigned short usTemp = 0;

// char cTemp = 0;

// unsigned char ucTemp = 0;

unsigned char ucCheckSum = 0;

float fTemp = 0.0;

unsigned char\* pTemp = (unsigned char \*)data.data();

// 校验

for(int i = 0; i < nSize - 1; i++)

{

ucCheckSum += pTemp[i];

}

if(ucCheckSum != pTemp[nSize - 1])

return false;

nIndex = 7;

memcpy(&usTemp,pTemp+nIndex,sizeof (usTemp));

info.usDistance = usTemp;

nIndex += 2;

memcpy(&fTemp,pTemp+nIndex,sizeof (fTemp));

info.fLongitude = fTemp;

nIndex += 4;

memcpy(&fTemp,pTemp+nIndex,sizeof (fTemp));

info.fLatitude = fTemp;

nIndex += 4;

memcpy(&usTemp,pTemp+nIndex,sizeof (usTemp));

info.usAltitude = usTemp;

return true;

}

//datapack.h

#ifndef DATAPACK\_H

#define DATAPACK\_H

#include <QObject>

#include <QByteArray>

#include <QDebug>

#include "public.h"

void **reverse**(char \* data,int nSize);

class **DataPack** : public QObject

{

Q\_OBJECT

public:

static DataPack \***getInstance**();

void **packTargetInfo**(QByteArray & data,STRUCT\_TARGET\_INFO & info);

void **packACK**(QByteArray & data, uint uSNO, ushort usATTR, uint uMSGID, ushort usOPC);

private:

//打包数据头、数据段消息头、数据位

void **packHeadData**(uint uMNUM, uint usTLEN, uint uSNO, ushort usOPC, ushort usATTR, uint uMSGID, QByteArray & data);

//打包校验和

void **packCRC**(QByteArray & data);

//打包数据内容

void **packDataTarget**(STRUCT\_TARGET\_INFO & info,QByteArray & data);

private:

explicit **DataPack**(QObject \*parent = nullptr);

static DataPack \*m\_dataPack;

private:

uint m\_uSNO;//发送的帧序号

};

#endif // DATAPACK\_H

//datapack.cpp

#include "datapack.h"

DataPack \*DataPack::m\_dataPack = nullptr;

void **reverse**(char \* data,int nSize)

{

char temp;

for (int i = 0; i < nSize / 2; i++)

{

temp = data[i];

data[i] = data[nSize - i - 1];

data[nSize - i - 1] = temp;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\* public 接口\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DataPack \*DataPack::**getInstance**()

{

if(nullptr == m\_dataPack)

m\_dataPack = new DataPack;

return m\_dataPack;

}

//打包目标信息

void DataPack::**packTargetInfo**(QByteArray & data,STRUCT\_TARGET\_INFO & info)

{

data.clear();

packHeadData(0xDDCCBBAA,80,m\_uSNO++,4,0xFFFF,TARGET\_INFO,*data*);

packDataTarget(*info*,*data*);

packCRC(*data*);

}

// 打包ACK消息

void DataPack::**packACK**(QByteArray & data, uint uSNO, ushort usATTR,uint uMSGID,ushort usOPC)

{

data.clear();

packHeadData(0xDDCCBBAA,16,uSNO,usOPC,usATTR,uMSGID,*data*);

packCRC(*data*);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* private \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DataPack::**DataPack**(QObject \*parent) : QObject(parent)

{

m\_uSNO = 0;

}

//标志头，总长度，操作码，帧序号

void DataPack::**packHeadData**(uint uMNUM, uint usTLEN, uint uSNO, ushort usOPC, ushort usATTR, uint uMSGID, QByteArray & data)

{

//标志头

reverse((char \*)(&uMNUM),sizeof(uMNUM));

data.append((char \*)(&uMNUM),sizeof(uMNUM));

//总长度

reverse((char \*)(&usTLEN),sizeof(usTLEN));

data.append((char \*)(&usTLEN),sizeof(usTLEN));

uint uCRC = 0;

//校验和

reverse((char \*)(&uCRC),sizeof(uCRC));

data.append((char \*)(&uCRC),sizeof(uCRC));

//帧序号

reverse((char \*)(&uSNO),sizeof(uSNO));

data.append((char \*)(&uSNO),sizeof(uSNO));

//操作码

reverse((char \*)(&usOPC),sizeof(usOPC));

data.append((char \*)(&usOPC),sizeof(usOPC));

//属性

reverse((char \*)(&usATTR),sizeof(usATTR));

data.append((char \*)(&usATTR),sizeof(usATTR));

//消息编号

reverse((char \*)(&uMSGID),sizeof(uMSGID));

data.append((char \*)(&uMSGID),sizeof(uMSGID));

}

//打包校验和

void DataPack::**packCRC**(QByteArray & data)

{

unsigned char \*pData = (unsigned char\*)data.data();

int nSize = data.size();

unsigned int uCheckSum = 0;

unsigned short checkSum = 0;

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

{

checkSum += pData[i];

}

checkSum = ~(checkSum & 0xFFFF) + 1;

uCheckSum = (checkSum & 0xFFFF);

reverse((char \*)(&uCheckSum),sizeof(uCheckSum));

memcpy(pData+8,(char \*)(&uCheckSum),sizeof(uCheckSum));

}

//战场目标信息

void DataPack::**packDataTarget**(STRUCT\_TARGET\_INFO & info,QByteArray & data)

{

unsigned long long ulTemp = 0;

unsigned int uTemp = 0;

unsigned short usTemp = 0;

float fTemp = 0.0;

double dTemp = 0.0;

// VMF编码值

uTemp = info.uVMFCode;

reverse((char \*)(&uTemp),sizeof(uTemp));

data.append((char \*)(&uTemp),sizeof(uTemp));

// 目标ID

ulTemp = info.ulTargetID;

reverse((char \*)(&ulTemp),sizeof(ulTemp));

data.append((char \*)(&ulTemp),sizeof(ulTemp));

// 目标位置信息

// 经度

dTemp = info.sPosInfo.dLongitude;

reverse((char \*)(&dTemp),sizeof(dTemp));

data.append((char \*)(&dTemp),sizeof(dTemp));

// 纬度

dTemp = info.sPosInfo.dLatitude;

reverse((char \*)(&dTemp),sizeof(dTemp));

data.append((char \*)(&dTemp),sizeof(dTemp));

// 高度有效标识

usTemp = info.sPosInfo.usAltFlag;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 高度

usTemp = info.sPosInfo.usAltitude;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 目标属性

usTemp = info.usTargetAttri;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 目标分类

usTemp = info.usTargetType;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 发现时间

// 年

usTemp = info.sFindTime.usYear;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 月

usTemp = info.sFindTime.usMonth;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 日

usTemp = info.sFindTime.usDay;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 时

usTemp = info.sFindTime.usHour;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 分

usTemp = info.sFindTime.usMinute;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 秒

usTemp = info.sFindTime.usSecond;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 发现手段,无人机

usTemp = info.usFindMethod;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 发现可靠性，非常可靠

usTemp = info.usReliability;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 保留字

usTemp = info.usReserve0;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 是否运动标识

usTemp = info.usTagMove;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 目标运动速度

fTemp = info.fMoveSpeed;

reverse((char \*)(&fTemp),sizeof(fTemp));

data.append((char \*)(&fTemp),sizeof(fTemp));

// 目标运动方向

usTemp = info.usMoveOrientation;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

// 保留字段

usTemp = info.usReserve;

reverse((char \*)(&usTemp),sizeof(usTemp));

data.append((char \*)(&usTemp),sizeof(usTemp));

}

//comsetuo.h

#ifndef COMSETUP\_H

#define COMSETUP\_H

#include <QWidget>

namespace **Ui** {

class **ComSetup**;

}

class **ComSetup** : public QWidget

{

Q\_OBJECT

public:

explicit **ComSetup**(QWidget \*parent = 0);

~***ComSetup***();

public:

void **setSerialName**(QStringList & listName);

void **setTCPLink**(bool bLink);

void **setSerialLink**(bool bLink);

signals:

// TCP

void **sglCloseTCP**();

void **sglCreateTCP**(QString &addr,quint16 port);

// Serial

void **sglCloseSerial**();

void **sglCreateSerial**(QString &serialName,quint32 baud);

private slots:

void **on\_pb\_link\_TCP\_clicked**();

void **on\_pb\_close\_TCP\_clicked**();

void **on\_pb\_link\_Serial\_clicked**();

void **on\_pb\_close\_Serial\_clicked**();

private:

void **setComboBox**();

void **setRegExp**();

void **setPushButton**();

private:

Ui::ComSetup \*ui;

bool m\_bTCPStatus;

bool m\_bSerialStatus;

};

#endif // COMSETUP\_H

//comsetup.cpp

#include "comsetup.h"

#include "ui\_comsetup.h"

ComSetup::**ComSetup**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ComSetup)

{

ui->setupUi(this);

m\_bTCPStatus = false;

m\_bSerialStatus = false;

setComboBox();

setRegExp();

setPushButton();

}

ComSetup::~***ComSetup***()

{

delete ui;

}

void ComSetup::**setSerialName**(QStringList & listName)

{

ui->cB\_serialName->addItems(listName);

}

void ComSetup::**setTCPLink**(bool bLink)

{

m\_bTCPStatus = bLink;

}

void ComSetup::**setSerialLink**(bool bLink)

{

m\_bSerialStatus = bLink;

}

void ComSetup::**setComboBox**()

{

QStringList listBaud;

listBaud << QString::number(115200)

<< QString::number(57600)

<< QString::number(38400)

<< QString::number(19200)

<< QString::number(9600)

<< QString::number(4800)

<< QString::number(2400)

<< QString::number(1200);

ui->cB\_baud->addItems(listBaud);

}

void ComSetup::**setRegExp**()

{

QString strIP = QString("^(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|[1-9])\\.")

+ QString("(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|\\d)\\.")

+ QString("(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|\\d)\\.")

+ QString("(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|\\d)$");

QString strPort = QString("(^\\d+$)");

//IP

QRegExp rxIP;

rxIP.setPattern(strIP);

QValidator \*valIp = new QRegExpValidator(rxIP);

ui->lE\_IP->setValidator(valIp);

//端口号

QRegExp rxPort;

rxPort.setPattern(strPort);

QValidator \*valPort = new QRegExpValidator(rxPort);

ui->lE\_port->setValidator(valPort);

}

void ComSetup::**setPushButton**()

{

// 连接按钮

ui->pb\_link\_TCP->setStyleSheet("QPushButton:pressed"

"{"

"background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}"

"QPushButton"

"{"

"background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}");

// 断开连接按钮

ui->pb\_close\_TCP->setStyleSheet("QPushButton:pressed"

"{"

"background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}"

"QPushButton"

"{"

"background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}");

// 连接按钮

ui->pb\_link\_Serial->setStyleSheet("QPushButton:pressed"

"{"

"background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}"

"QPushButton"

"{"

"background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}");

// 断开连接按钮

ui->pb\_close\_Serial->setStyleSheet("QPushButton:pressed"

"{"

"background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}"

"QPushButton"

"{"

"background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}");

}

// slot

void ComSetup::**on\_pb\_link\_TCP\_clicked**()

{

if(m\_bTCPStatus)

return;

QString addr = ui->lE\_IP->text();

quint16 port = ui->lE\_port->text().toUShort();

if(addr.isEmpty() || 0 == port)

{

addr = "127.0.0.1";

port = 28000;

}

emit sglCreateTCP(*addr*,port);

}

void ComSetup::**on\_pb\_close\_TCP\_clicked**()

{

if(m\_bTCPStatus)

{

emit sglCloseTCP();

m\_bTCPStatus = false;

}

}

void ComSetup::**on\_pb\_link\_Serial\_clicked**()

{

if(m\_bSerialStatus)

return;

QString name = ui->cB\_serialName->currentText();

qint32 baud = ui->cB\_baud->currentText().toInt();

if(name.isEmpty() || baud == 0)

return;

emit sglCreateSerial(*name*,baud);

}

void ComSetup::**on\_pb\_close\_Serial\_clicked**()

{

if(m\_bSerialStatus)

{

emit sglCloseSerial();

m\_bSerialStatus = false;

}

}

//posinfo.h

#ifndef POSINFO\_H

#define POSINFO\_H

#include <QWidget>

#include "public.h"

namespace **Ui** {

class **PosInfo**;

}

class **PosInfo** : public QWidget

{

Q\_OBJECT

public:

explicit **PosInfo**(QWidget \*parent = 0);

~***PosInfo***();

void **setLinkStatus**(bool bLink);

void **setPosInfo**(STRUCT\_POS &info);

private:

void **setComboBox**();

void **setRegExp**();

void **setPushButton**();

signals:

// send data

void **sglSendData**(STRUCT\_TARGET\_INFO & info);

private slots:

void **on\_pb\_sendPos\_clicked**();

private:

Ui::PosInfo \*ui;

bool m\_bLinkStatus;

//combo box 描述

QStringList m\_listTargetAttr; // 目标属性

QStringList m\_listTargetType; // 目标分类

QStringList m\_listFindMethod; // 发现手段

QStringList m\_listReliability; // 发现可靠性

//combo box 值

QList<ushort> m\_listTargetAttrValue; // 目标属性

QList<ushort> m\_listTargetTypeValue; // 目标分类

QList<ushort> m\_listFindMethodValue; // 发现手段

QList<ushort> m\_listReliabilityValue; // 发现可靠性

};

#endif // POSINFO\_H

//posinfo.cpp

#include "posinfo.h"

#include "ui\_posinfo.h"

#include <QDebug>

PosInfo::**PosInfo**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::PosInfo)

{

ui->setupUi(this);

m\_bLinkStatus = false;

// 描述

m\_listTargetAttr.clear(); // 目标属性

m\_listTargetType.clear(); // 目标分类

m\_listFindMethod.clear(); // 发现手段

m\_listReliability.clear(); // 发现可靠性

// 值

m\_listTargetAttrValue.clear(); // 目标属性

m\_listTargetTypeValue.clear(); // 目标分类

m\_listFindMethodValue.clear(); // 发现手段

m\_listReliabilityValue.clear(); // 发现可靠性

setComboBox(); // 设置comboBox

setRegExp(); // 设置正则表达式

setPushButton(); // 设置按钮状态

}

PosInfo::~***PosInfo***()

{

delete ui;

}

void PosInfo::**setLinkStatus**(bool bLink)

{

m\_bLinkStatus = bLink;

}

void PosInfo::**setPosInfo**(STRUCT\_POS &info)

{

ui->lE\_lon->setText(QString::number(info.fLongitude,'f',8));

ui->lE\_lat->setText(QString::number(info.fLatitude,'f',8));

ui->lE\_alt->setText(QString::number(info.usAltitude));

}

void PosInfo::**setComboBox**()

{

// 目标属性

m\_listTargetAttr << "无报告" // 无报告

<< "敌方" // 敌方

<< "我方" // 我方

<< "友方" // 友方

<< "中立" // 中立

<< "不明"; // 不明

for(ushort i = 0; i < 6; i++)

m\_listTargetAttrValue.append(i);

ui->combox\_attri->addItems(m\_listTargetAttr);

ui->combox\_attri->setCurrentIndex(1);

// 目标分类

m\_listTargetType << "设施目标" // 设施目标

<< "指挥机构类" // 指挥机构类

<< "雷达站" // 雷达站

<< "直升机起降场" // 直升机起降场

<< "碉堡" // 碉堡

<< "弹药库" // 弹药库

<< "步兵" // 步兵

<< "装甲车辆" // 装甲车辆

<< "坦克" // 坦克

<< "汽车" // 汽车

<< "舟桥" // 舟桥

<< "列车" // 列车

<< "直升机" // 直升机

<< "无人机"; // 无人机

m\_listTargetTypeValue << 1 << 2 << 16 << 40 << 61 << 95 << 203

<< 229 << 238 << 239 << 240 << 241 << 257 << 258;

ui->combox\_type->addItems(m\_listTargetType);

ui->combox\_type->setCurrentIndex(13);

// 发现手段

m\_listFindMethod << "无线电侦听" // 无线电侦听

<< "无线电侦测" // 无线电侦测

<< "技术侦察" // 技术侦察

<< "观察" // 观察

<< "雷达" // 雷达

<< "照摄像" // 照摄像

<< "战场电视" // 战场电视

<< "无人机" // 无人机

<< "武装侦察" // 武装侦察

<< "捕俘" // 捕俘

// << "观察" // 观察

<< "监视" // 监视

<< "军调（谍报）" // 军调（谍报）

<< "各级通报"; // 各级通报

for(ushort i = 0; i < 14; i++)

{

if(i != 10) // 没有观察

m\_listFindMethodValue.append(i);

}

ui->combox\_method->addItems(m\_listFindMethod);

ui->combox\_method->setCurrentIndex(7);

// 发现可靠性

m\_listReliability << "完全可靠" // 完全可靠

<< "通常可靠" // 通常可靠

<< "相当可靠" // 相当可靠

<< "不总是可靠" // 不总是可靠

<< "不可靠" // 不可靠

<< "不能判断可靠性"; // 不能判断可靠性

for(ushort i = 0; i < 6; i++)

m\_listReliabilityValue.append(i);

ui->combox\_reliability->addItems(m\_listReliability);

ui->combox\_reliability->setCurrentIndex(0);

}

void PosInfo::**setRegExp**()

{

QString strtAlt = QString("(^\\d+$)");

QString strLon = QString("(^-?(180([.][0]{1,8})?|1?[0-7]?\\d(\\.\\d{1,8})?|[1-9]?\\d(\\.\\d{1,8})?|[0-9]?(\\.\\d{1,8})?)$)");

QString strLat = QString("(^-?(90([.][0]{1,8})?|[1-8]?\\d(\\.\\d{1,8})?|[0-9]?(\\.\\d{1,8})?)$)");

QString strSpeed = QString("(^((-?\\d+)(\\.\\d{1,8})?)$)");

QString strOrientation = QString("(^\\d+$)");

//高度

QRegExp rxAlt;

rxAlt.setPattern(strtAlt);

QValidator \*valAlt = new QRegExpValidator(rxAlt);

ui->lE\_alt->setValidator(valAlt);

//经度

QRegExp rxLon;

rxLon.setPattern(strLon);

QValidator \*valLon = new QRegExpValidator(rxLon);

ui->lE\_lon->setValidator(valLon);

//纬度

QRegExp rxLat;

rxLat.setPattern(strLat);

QValidator \*valLat = new QRegExpValidator(rxLat);

ui->lE\_lat->setValidator(valLat);

//速度

QRegExp rxSpeed;

rxSpeed.setPattern(strSpeed);

QValidator \*valSpeed = new QRegExpValidator(rxSpeed);

ui->lE\_speed->setValidator(valSpeed);

//方向

QRegExp rxOrientation;

rxOrientation.setPattern(strOrientation);

QValidator \*valOrientation = new QRegExpValidator(rxOrientation);

ui->lE\_fangxiang->setValidator(valOrientation);

}

void PosInfo::**setPushButton**()

{

// 发送按钮

ui->pb\_sendPos->setStyleSheet("QPushButton:pressed"

"{"

"background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}"

"QPushButton"

"{"

"background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';"

"}");

}

void PosInfo::**on\_pb\_sendPos\_clicked**()

{

if(!m\_bLinkStatus)

{

return;

}

STRUCT\_TARGET\_INFO info;

// 位置信息

info.sPosInfo.dLongitude = ui->lE\_lon->text().toDouble();

info.sPosInfo.dLatitude = ui->lE\_lat->text().toDouble();

info.sPosInfo.usAltitude = ui->lE\_alt->text().toUShort();

if(ui->rb\_alt->isChecked())

info.sPosInfo.usAltFlag = 1;

else

info.sPosInfo.usAltFlag = 0;

// 运动信息

if(ui->rb\_move->isChecked())

info.usTagMove = 1;

else

info.usTagMove = 0;

info.fMoveSpeed = ui->lE\_speed->text().toFloat();

info.usMoveOrientation = ui->lE\_fangxiang->text().toUShort();

// 目标信息

info.usTargetAttri = m\_listTargetAttrValue.at( ui->combox\_attri->currentIndex() );

info.usTargetType = m\_listTargetTypeValue.at( ui->combox\_type->currentIndex() );

// 发现信息

info.usFindMethod = m\_listFindMethodValue.at( ui->combox\_method->currentIndex() );

info.usReliability = m\_listReliabilityValue.at( ui->combox\_reliability->currentIndex() );

emit sglSendData(*info*);

}

//positionui.h

#ifndef POSITIONUI\_H

#define POSITIONUI\_H

#include <QWidget>

#include <QRegExp>

#include <QRegExpValidator>

#include <QValidator>

#include "public.h"

namespace **Ui** {

class **PositionUi**;

}

class **PositionUi** : public QWidget

{

Q\_OBJECT

public:

explicit **PositionUi**(QWidget \*parent = nullptr);

~***PositionUi***();

public:

void **addPosUI**(QWidget \* posUI);

void **addComUI**(QWidget \* comUI);

void **setScoutFlag**(); // 设置侦察标志

void **setLinkStatus**(bool bStatus);// 连接状态

void **showHintInfo**(QString &info); // 提示信息

signals:

void **sglCloseLink**();

void **sglCreateLink**(QString &addr,quint16 port);

void **sglSendData**(STRUCT\_TARGET\_INFO & info);

private slots:

void **on\_pb\_set\_clicked**();

void **on\_pb\_pos\_clicked**();

private:

Ui::PositionUi \*ui;

private:

void **hintShow**(QString &info);

};

#endif // POSITIONUI\_H

//positionui.cpp

#include "positionui.h"

#include "ui\_positionui.h"

#include <QDebug>

PositionUi::**PositionUi**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::PositionUi)

{

ui->setupUi(this);

// 设置

setWindowTitle("Position");

setWindowFlags( windowFlags() & ~Qt::WindowMinMaxButtonsHint);

// 显示

ui->wdg\_set->show();

ui->wdg\_pos->hide();

ui->pb\_set->setStyleSheet("background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';");

ui->pb\_pos->setStyleSheet("background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';");

}

PositionUi::~***PositionUi***()

{

delete ui;

}

void PositionUi::**addPosUI**(QWidget \* posUI)

{

posUI->setParent(ui->wdg\_pos);

}

void PositionUi::**addComUI**(QWidget \* comUI)

{

comUI->setParent(ui->wdg\_set);

}

void PositionUi::**hintShow**(QString &info)

{

ui->lb\_hint->setText(info);

}

void PositionUi::**setLinkStatus**(bool bStatus)

{

if(bStatus)

ui->lb\_status->setStyleSheet("background-color: rgb(0,255,0);border-radius:10px;");

else

ui->lb\_status->setStyleSheet("background-color: rgb(255,0,0);border-radius:10px;");

}

void PositionUi::**showHintInfo**(QString &info)

{

hintShow(*info*);

}

void PositionUi::**setScoutFlag**()

{

ui->lb\_scoutFlag->setStyleSheet("background-color: rgb(0,255,0);border-radius:10px;");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 按钮处理 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//设置界面

void PositionUi::**on\_pb\_set\_clicked**()

{

ui->pb\_set->setStyleSheet("background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';");

ui->pb\_pos->setStyleSheet("background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';");

ui->wdg\_set->show();

ui->wdg\_pos->hide();

}

//位置界面

void PositionUi::**on\_pb\_pos\_clicked**()

{

ui->pb\_set->setStyleSheet("background-color: rgb(22, 50, 71);"

"border:0px ;"

"font: 12pt '微软雅黑';");

ui->pb\_pos->setStyleSheet("background-color: rgb(170,170,255);"

"border:0px ;"

"font: 12pt '微软雅黑';");

ui->wdg\_set->hide();

ui->wdg\_pos->show();

}

//uimanage.h

#ifndef UIMANAGE\_H

#define UIMANAGE\_H

#include <QObject>

#include "comsetup.h"

#include "posinfo.h"

#include "positionui.h"

class **UIManage** : public QObject

{

Q\_OBJECT

public:

explicit **UIManage**(QObject \*parent = nullptr);

public:

void **showUI**();

signals:

// TCP

void **sglCloseTCP**();

void **sglCreateTCP**(QString &addr,quint16 port);

// Serial

void **sglCloseSerial**();

void **sglCreateSerial**(QString &serialName,quint32 baud);

// send data

void **sglSendData**(STRUCT\_TARGET\_INFO & info);

public slots:

// from com

void **slotTCPStatus**(bool bLink);

void **slotSerialStatus**(bool bLink);

void **slotHintInfo**(QString &info);

void **slotScoutFlag**();

void **slotSerialName**(QStringList & listName);

void **slotPosInfo**(STRUCT\_POS &info);

private:

PositionUi \*m\_mainUI; // 主界面

PosInfo \*m\_posUI; // 位置信息界面

ComSetup \*m\_comUI; // 通信设置界面

};

#endif // UIMANAGE\_H

//uimanage.cpp

#include "uimanage.h"

UIManage::**UIManage**(QObject \*parent) : QObject(parent)

{

// 创建对象

m\_mainUI = new PositionUi;

m\_posUI = new PosInfo;

m\_comUI = new ComSetup;

// 添加界面

m\_mainUI->addComUI(m\_comUI);

m\_mainUI->addPosUI(m\_posUI);

// connect Com

connect(m\_comUI,&ComSetup::sglCreateTCP,this,&UIManage::sglCreateTCP);

connect(m\_comUI,&ComSetup::sglCloseTCP,this,&UIManage::sglCloseTCP);

connect(m\_comUI,&ComSetup::sglCreateSerial,this,&UIManage::sglCreateSerial);

connect(m\_comUI,&ComSetup::sglCloseSerial,this,&UIManage::sglCloseSerial);

// connect pos

connect(m\_posUI,&PosInfo::sglSendData,this,&UIManage::sglSendData);

}

void UIManage::**showUI**()

{

m\_mainUI->show();

}

void UIManage::**slotTCPStatus**(bool bLink)

{

m\_mainUI->setLinkStatus(bLink);

m\_comUI->setTCPLink(bLink);

m\_posUI->setLinkStatus(bLink);

}

void UIManage::**slotSerialStatus**(bool bLink)

{

m\_comUI->setSerialLink(bLink);

}

void UIManage::**slotHintInfo**(QString &info)

{

m\_mainUI->showHintInfo(*info*);

}

void UIManage::**slotScoutFlag**()

{

m\_mainUI->setScoutFlag();

}

void UIManage::**slotSerialName**(QStringList & listName)

{

m\_comUI->setSerialName(*listName*);

}

void UIManage::**slotPosInfo**(STRUCT\_POS &info)

{

m\_posUI->setPosInfo(*info*);

}