УТВЕРЖДЕН

МГТУ.111111.001-01 12 01-ЛУ

**СПЕЦИАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ «ККМ»**

**Текст программы**

**МГТУ.111111.001-01 12 01**

**Листов 91**

2020

АННОТАЦИЯ

В данном программном документе приведен текст программы «ККМ», предназначенной для работы с кассовым аппаратом. Исходным языком данной разработки является С++. Среда разработки - QT, компилятор – MinGW (локализованная русская версия).

Основной функцией программы ККМ.exe является общение с кассиром, ввод и вывод информации, а также передача и получение данных по USB/COM порту от кассового устройства.

Оформление программного документа «Текст программы» произведено по требованиям ЕСПД (ГОСТ 19.101-77 [[1]](#footnote-1), ГОСТ 19.103-77 [[2]](#footnote-2), ГОСТ 19.104-78\* [[3]](#footnote-3), ГОСТ 19.105-78\* [[4]](#footnote-4), ГОСТ 19.106-78\* [[5]](#footnote-5), ГОСТ 19.401-78 [[6]](#footnote-6), ГОСТ 19.604-78\* [[7]](#footnote-7)).

СОДЕРЖАНИЕ

[1. Текст Программы ккт на исходном языке 4](#_Toc59704205)

[1.1 Текст GUI ЧАСТИ 5](#_Toc59704206)

[1.2 Текст Связной части с аппаратурой 44](#_Toc59704207)

[Перечень принятых сокращений 90](#_Toc59704208)

1. Текст Программы ккт на исходном языке

1.1 Текст GUI ЧАСТИ

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[]) {

    QApplication a(argc, argv);

    auto mainWindow = new MainWindow;

    mainWindow->setAttribute(Qt::WA\_DeleteOnClose);

    mainWindow->setWindowTitle("KKM");

    mainWindow->setWindowIcon(QIcon("C:/Users/Sergei/Desktop/7 сем/АСВТ/Курсовая/icon.jpg"));

    mainWindow->show();

    return a.exec();

}

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QTabBar>

#include <QWidget>

#include <QMessageBox>

#include <QDateTime>

#include <QDebug>

MainWindow::MainWindow(QWidget \*parent) :

    QMainWindow(parent),

    ui(new Ui::MainWindow),

    \_settingProgW(new SettingProgWindow),

    \_settingKKMW(new SettingKKMWindow),

    \_loginW(new LoginOperatorOFD),

    \_regW(new RegistrationWindow),

    \_regData(new RegistrationData),

    \_commandW(new CommandWindow),

    \_isReg(false),

    \_method\_calc(0),

    \_subject\_calc(0),

    \_nds(0),

    \_quantity(0),

    \_count\_check(0),

    \_posishion\_check(0),

    \_checkAmount(0),

    \_received(0),

    \_change(0),

    \_KKMAmount(0),

    \_cashAmount(0),

    \_cashlessAmount(0) {

    ui->setupUi(this);

    initKKMWindow();

    initNewChekWindow();

    initPDB();

    initHDB();

    initInspectWindow();

    initRegistrWindow();

    initCommandWindow();

    ui->new\_chek->setEnabled(false);

    ui->inspect->setEnabled(false);

    ui->setting\_prog->setEnabled(false);

    ui->other\_commands->setEnabled(false);

    ui->open\_shift->setEnabled(false);

    ui->close\_shift->setEnabled(false);

    ui->info->setEnabled(false);

    ui->buttonFiscalEnd->setEnabled(false);

    ui->buttonFiscal->setEnabled(false);

}

MainWindow::~MainWindow() {

    delete ui;

    delete \_settingProgW;

    delete \_settingKKMW;

    delete \_loginW;

    delete \_regW;

    delete \_commandW;

    delete \_pdb;

    delete \_hdb;

    delete \_modelP;

    delete \_modelH;

    delete \_regData;

}

void MainWindow::initPDB() {

    \_pdb = new ProductDB();

    \_pdb->connectToDataBase();

    \_modelP = new QSqlTableModel(this);

    // Инициализируем модель для представления данных

    this->setupModel(TABLE\_PRODUCT,

                     QStringList() << trUtf8("ID")

                                   << trUtf8("Штрих-код")

                                   << trUtf8("Название товара")

                                   << trUtf8("Цена"), \_modelP);

    this->createUI(ui->productDB, \_modelP); // Инициализируем внешний вид таблицы с данными

}

void MainWindow::initHDB() {

    \_hdb = new HistoryDB();

    \_hdb->connectToDataBase();

    \_modelH = new QSqlTableModel(this);

    // Инициализируем модель для представления данных

    this->setupModel(TABLE\_HISTORY,

                     QStringList() << trUtf8("Создан")

                                   << trUtf8("№ чека")

                                   << trUtf8("Операция")

                                   << trUtf8("Позиций")

                                   << trUtf8("Сумма нал.")

                                   << trUtf8("Сумма безнал.")

                                   << trUtf8("Получено")

                                   << trUtf8("Сдача")

                                   << trUtf8("№ фиск. док.")

                                   << trUtf8("Фиск. пр."), \_modelH);

    this->createUI(ui->historyDB, \_modelH); // Инициализируем внешний вид таблицы с данными

}

void MainWindow::initKKMWindow() {

    auto size = ui->tabWidget->width() / ui->tabWidget->count();

    ui->tabWidget->setStyleSheet(ui->tabWidget->styleSheet() +

                                        "QTabBar::tab {"

                                        "width: " + QString::number(size) +

                                        "px; height: 35px}" );

    ui->is\_open->setText(STATUS\_SHIFT[0]);

    ui->is\_redy\_kkm->setText(STATUS\_KKM[0]);

    ui->is\_fiscal->setText(STATUS\_FN[0]);

    ui->line\_summ->setText(QString::number(\_KKMAmount));

    \_settingProgW->setAttribute(Qt::WA\_DeleteOnClose);

    \_settingProgW->setDefaultSetting(\_method\_calc, \_subject\_calc, \_nds, \_quantity);

    \_settingProgW->setWindowFlags(Qt::Window | Qt::WindowTitleHint | Qt::CustomizeWindowHint);

    \_settingProgW->setWindowTitle("Настройка программы");

    \_settingKKMW->setAttribute(Qt::WA\_DeleteOnClose);

    \_settingKKMW->setWindowFlags(Qt::Window | Qt::WindowTitleHint | Qt::CustomizeWindowHint);

    \_settingKKMW->setWindowTitle("Настройка кассы");

    connect(\_settingProgW, SIGNAL(saveSetting(size\_t, size\_t, size\_t, size\_t)), this,

            SLOT(recieveProgSetting(size\_t, size\_t, size\_t, size\_t)));

    connect(\_settingKKMW, SIGNAL(saveSetting(const QString &, const QString &)), this,

            SLOT(initKKMSetting(const QString &, const QString &)));

}

void MainWindow::initNewChekWindow() {

    ui->line\_sum\_check->setText("0.00");

    ui->line\_drop\_check->setText("0.00");

    ui->line\_sdacha\_check->setText("0");

    ui->number\_check->setText(QString::number(\_count\_check));

    ui->comboBox\_CashAmount->addItems({"Наличные", "Безнал."});

    ui->comboBox\_sell->addItems({"Приход (продажа)", "Возврат прихода", "Расход", "Возврат расхода"});

    ui->label\_is\_pr->setText(METHOD\_OF\_CALC[\_method\_calc]);

    ui->label\_subject->setText(SUBJECT\_OF\_CALC[\_subject\_calc]);

    ui->label\_is\_nds->setText(NDS[\_nds]);

    ui->tableWidget\_check->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Stretch);

}

void MainWindow::initInspectWindow() {

    \_loginW->setAttribute(Qt::WA\_DeleteOnClose);

    \_loginW->setWindowFlags(Qt::Window | Qt::WindowTitleHint | Qt::CustomizeWindowHint);

    \_loginW->setWindowTitle("Вход инспектора ФНС");

    connect(\_loginW, SIGNAL(login(const QString &, const QString &)), this,

            SLOT(checkLoginFNS(const QString &, const QString &)));

}

void MainWindow::checkLoginFNS(const QString &login, const QString &pass) {

    if (login == "admin" && pass == "admin") {

        ui->inspect->setEnabled(true);

        \_loginW->hide();

    }

}

void MainWindow::initRegistrWindow() {

    \_regW->setAttribute(Qt::WA\_DeleteOnClose);

    \_regW->setWindowFlags(Qt::Window | Qt::WindowTitleHint | Qt::CustomizeWindowHint);

    \_regW->setWindowTitle("Регистрация ККМ");

    connect(\_regW, SIGNAL(saveData(const RegistrationData \*)), this,

            SLOT(registration(const RegistrationData \*)));

}

void MainWindow::registration(const RegistrationData \*data) {

    \_regData->user = data->user;

    \_regData->addres = data->addres;

    \_regData->placeSettlement = data->placeSettlement;

    \_regData->regNumberKKT = data->regNumberKKT;

    \_regData->INN = data->INN;

    \_regData->siteFNS = data->siteFNS;

    \_regData->nc = data->nc;

    \_regData->wm = data->wm;

    \_regData->operatorOFD = data->operatorOFD;

    \_regData->INNoperator = data->INNoperator;

    \_regData->IP = data->IP;

    \_regData->port = data->port;

    if (ui->is\_redy\_kkm->text() != STATUS\_KKM[0]) {

        auto response = \_service->\_\_30\_\_GetFNStatus();

        if (response->ErrorMsg != "") {

            QMessageBox::warning(this, tr("Ошибка"),

                                       tr(response->ErrorMsg.c\_str()),

                                       QMessageBox::Ok);

            return;

        } else {

            if (response->PhaseOfLife == FN\_LIFE\_PHASES.at(3)) {

                ui->is\_fiscal->setText(STATUS\_FN[2]);

            } else if (response->PhaseOfLife == FN\_LIFE\_PHASES.at(1)) {

                auto response = \_service->\_\_02\_\_StartFiscalisation();

                if (response->ErrorMsg != "") {

                    QMessageBox::warning(this, tr("Ошибка"),

                                               tr(response->ErrorMsg.c\_str()),

                                               QMessageBox::Ok);

                    return;

                } else {

                    auto doc = CommonData{};

                    doc.UserName = \_regData->user.toStdString();

                    doc.Cashier = \_nameCashir.toStdString();

                    doc.Address = \_regData->addres.toStdString();

                    doc.InnOFD = \_regData->INNoperator.toStdString();

                    auto response = \_service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

                    if (response->ErrorMsg != "") {

                        QMessageBox::warning(this, tr("Ошибка"),

                                                   tr(response->ErrorMsg.c\_str()),

                                                   QMessageBox::Ok);

                        return;

                    } else {

                        auto req = ApproveFiscalisationRequest{};

                        req.DateTime = time(0);

                        req.Inn\_cp866 = \_regData->INN.toStdString();

                        req.KKTNumber\_cp866 = \_regData->regNumberKKT.toStdString();

                        req.NalogCode = \_regData->nc;

                        req.WorkMode = \_regData->wm;

                        auto response = \_service->\_\_03\_\_ApproveFiscalisation(req);

                        if (response->ErrorMsg != "") {

                            QMessageBox::warning(this, tr("Ошибка"),

                                                       tr(response->ErrorMsg.c\_str()),

                                                       QMessageBox::Ok);

                            return;

                        }

                        ui->is\_fiscal->setText(STATUS\_FN[2]);

                        ui->line\_summ->setText(QString::number(10000.00));

                        addRowinHistory("", "Фискализация", "", "", "", "", "",

                                        QString::number(response->FiscDocNumber),

                                        QString::number(response->FiscSign));

                        QMessageBox::information(this, tr("Сообщение"),

                                                   tr("ФН фискализирована!"),

                                                   QMessageBox::Ok);

                        ui->setting\_prog->setEnabled(true);

                        ui->other\_commands->setEnabled(true);

                        ui->open\_shift->setEnabled(true);

                        ui->close\_shift->setEnabled(true);

                        ui->info->setEnabled(true);

                        ui->buttonFiscalEnd->setEnabled(true);

                    }

                }

            }

        }

    } else {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("ККМ не готова!"),

                                   QMessageBox::Ok);

    }

}

void MainWindow::initCommandWindow() {

    \_commandW->setAttribute(Qt::WA\_DeleteOnClose);

    \_commandW->setWindowFlags(Qt::Window | Qt::WindowTitleHint | Qt::CustomizeWindowHint);

    \_commandW->setWindowTitle("Команды ФН");

    connect(\_commandW, SIGNAL(request(COMMANDS\_FN)), this, SLOT(executeCommand(COMMANDS\_FN)));

    connect(this, SIGNAL(sendCommandInfo(const QString &)), \_commandW, SLOT(printInfo(const QString &)));

}

void MainWindow::on\_push\_exit\_clicked() {

    this->close();

}

bool MainWindow::initKKMSetting(const QString &name, const QString &port) {

    if (ui->is\_redy\_kkm->text() == STATUS\_KKM[1]) {

        QMessageBox::information(this, tr("Сообщение"),

                                   tr("Соединение уже установлено!"),

                                   QMessageBox::Ok);

        return true;

    }

    \_nameCashir = name;

    ui->nameCashir->setText(\_nameCashir);

    ui->label\_setSetting->hide();

    \_port = port;

    \_service = std::make\_shared<Hardware>(port.toStdWString(), 100000);

    auto connected = \_service->get\_connection\_status();

    // добавить таймаут

    if (connected) {

        ui->is\_redy\_kkm->setText(STATUS\_KKM[1]);

        ui->is\_fiscal->setText(STATUS\_FN[1]);

        // проверка фискализирована ли

        auto response = \_service->\_\_30\_\_GetFNStatus();

        if (response->ErrorMsg != "") {

            QMessageBox::warning(this, tr("Ошибка"),

                                       tr(response->ErrorMsg.c\_str()),

                                       QMessageBox::Ok);

            return false;

        } else {

            if (response->PhaseOfLife == FN\_LIFE\_PHASES.at(3))

                ui->is\_fiscal->setText(STATUS\_FN[2]);

            ui->buttonFiscal->setEnabled(true);

            QMessageBox::information(this, tr("Сообщение"),

                                       tr("Соединение установлено!"),

                                       QMessageBox::Ok);

            ui->buttonFiscal->setEnabled(true);

        }

    } else {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("Проверьте соединение!"),

                                   QMessageBox::Ok);

        return false;

    }

}

void MainWindow::on\_buttonFiscal\_clicked() {

    if (\_isReg) {

        QMessageBox::information(this, tr("Сообщение"),

                                   tr("Регистрация выполнена!"),

                                   QMessageBox::Ok);

        return;

    }

    \_isReg = true;

    ui->tabWidget->setEnabled(true);

    ui->label\_acceess->hide();

    \_regW->show();

}

void MainWindow::on\_buttonFiscalEnd\_clicked() {

    if (ui->is\_fiscal->text() == STATUS\_FN[2]) {

        if (ui->is\_open->text() == STATUS\_SHIFT[1]) {

            QMessageBox::information(this, tr("Сообщение"),

                                       tr("Закройте смену!"),

                                       QMessageBox::Ok);

            return;

        }

        auto response = \_service->\_\_04\_\_StartCloseFiscalisation();

        if (response->ErrorMsg != "") {

            QMessageBox::warning(this, tr("Ошибка"),

                                       tr(response->ErrorMsg.c\_str()),

                                       QMessageBox::Ok);

            return;

        } else {

            auto doc = CommonData{};

            doc.Cashier = \_nameCashir.toStdString();

            doc.Address = \_regData->addres.toStdString();

            auto response = \_service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

            if (response->ErrorMsg != "") {

                QMessageBox::warning(this, tr("Ошибка"),

                                           tr(response->ErrorMsg.c\_str()),

                                           QMessageBox::Ok);

                return;

            } else {

                auto req = CloseFiscalisationRequest{};

                req.DateTime = time(0);

                req.KKTNumber\_cp866 = \_regData->regNumberKKT.toStdString();

                auto response = \_service->\_\_05\_\_CloseFiscalisation(req);

                if (response->ErrorMsg != "") {

                    QMessageBox::warning(this, tr("Ошибка"),

                                               tr(response->ErrorMsg.c\_str()),

                                               QMessageBox::Ok);

                    return;

                }

                ui->is\_fiscal->setText(STATUS\_FN[3]);

                addRowinHistory("", ""

                                    "Закрытие фискального режима", "", "", "", "", "",

                                QString::number(response->FiscDocNumber),

                                QString::number(response->FiscSign));

                QMessageBox::information(this, tr("Сообщение"),

                                           tr("Фискальный режим закрыт!"),

                                           QMessageBox::Ok);

                ui->kkm->setEnabled(false);

                ui->new\_chek->setEnabled(false);

                ui->inspect->setEnabled(true);

            }

        }

    } else {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("ККМ не фискализирована!"),

                                   QMessageBox::Ok);

    }

}

void MainWindow::on\_open\_shift\_clicked()

{

    if (ui->is\_fiscal->text() != STATUS\_FN[0]) {

        if (ui->is\_open->text() == STATUS\_SHIFT[1]) {

            QMessageBox::information(this, tr("Сообщение"),

                                       tr("Смена уже открыта!"),

                                       QMessageBox::Ok);

            return;

        }

        auto req = StartOpeningShiftRequest{};

        req.DateTime = time(0);

        auto response = \_service->\_\_11\_\_StartOpeningShift(req);

        if (response->ErrorMsg != "") {

            QMessageBox::warning(this, tr("Ошибка"),

                                       tr(response->ErrorMsg.c\_str()),

                                       QMessageBox::Ok);

            return;

        } else {

            auto doc = CommonData{};

            doc.UserName = \_regData->user.toStdString();

            doc.Cashier = \_nameCashir.toStdString();

            doc.Address = \_regData->addres.toStdString();

            auto response = \_service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

            if (response->ErrorMsg != "") {

                QMessageBox::warning(this, tr("Ошибка"),

                                           tr(response->ErrorMsg.c\_str()),

                                           QMessageBox::Ok);

                return;

            } else {

                auto response = \_service->\_\_12\_\_ApproveOpeningShift();

                if (response->ErrorMsg != "") {

                    QMessageBox::warning(this, tr("Ошибка"),

                                               tr(response->ErrorMsg.c\_str()),

                                               QMessageBox::Ok);

                    return;

                }

                ui->is\_open->setText(STATUS\_SHIFT[1]);

                ui->new\_chek->setEnabled(true);

                ui->val\_sum\_nal->setText("0.00");

                ui->val\_sum\_beznal->setText("0.00");

                ui->shiftNumber->setText(QString::number(response->ShiftNum));

                addRowinHistory("", "Открытие смены", "", "", "", "", "",

                                QString::number(response->FiscDocNumber),

                                QString::number(response->FiscSign));

            }

        }

    } else {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("ККМ не фискализирована!"),

                                   QMessageBox::Ok);

    }

}

void MainWindow::on\_close\_shift\_clicked()

{

    if (ui->is\_fiscal->text() != STATUS\_FN[0]) {

        if (ui->is\_open->text() == STATUS\_SHIFT[0]) {

            QMessageBox::information(this, tr("Сообщение"),

                                       tr("Смена не открыта!"),

                                       QMessageBox::Ok);

            return;

        }

        auto req = StartCloseShiftRequest{};

        req.DateTime = time(0);

        auto response = \_service->\_\_13\_\_StartCloseShift(req);

        if (response->ErrorMsg != "") {

            QMessageBox::warning(this, tr("Ошибка"),

                                       tr(response->ErrorMsg.c\_str()),

                                       QMessageBox::Ok);

            return;

        } else {

            auto doc = CommonData{};

            doc.UserName = \_regData->user.toStdString();

            doc.Cashier = \_nameCashir.toStdString();

            doc.Address = \_regData->addres.toStdString();

            auto response = \_service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

            if (response->ErrorMsg != "") {

                QMessageBox::information(this, tr("Сообщение"),

                                           tr("Смена не открыта!"),

                                           QMessageBox::Ok);

                return;

            } else {

                auto response = \_service->\_\_14\_\_ApproveCloseShift();

                if (response->ErrorMsg != "") {

                    QMessageBox::information(this, tr("Сообщение"),

                                               tr("Смена не открыта!"),

                                               QMessageBox::Ok);

                    return;

                }

                ui->is\_open->setText(STATUS\_SHIFT[2]);

                ui->new\_chek->setEnabled(false);

                ui->shiftNumber->setText(QString::number(response->ShiftNum));

                ui->number\_check->setText(QString::number(0)); // обновляем

                ui->val\_sum\_nal->setText(QString::number(\_KKMAmount));

                ui->val\_sum\_beznal->setText(QString::number(\_KKMAcashlessAmount));

                addRowinHistory("", "Закрытие смены", "",

                                QString::number(\_KKMcashAmount, 'f', 2),

                                QString::number(\_KKMAcashlessAmount, 'f', 2), "", "",

                                QString::number(response->FiscDocNumber),

                                QString::number(response->FiscSign));

                \_KKMcashAmount = 0;

                \_KKMAcashlessAmount = 0;

            }

        }

    } else {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("ККМ не фискализирована!"),

                                   QMessageBox::Ok);

    }

}

void MainWindow::on\_get\_check\_clicked() // формирование чека

{

    if (\_checkAmount > \_received || \_checkAmount == 0) {

        int ret = QMessageBox::warning(this, tr("Ошибка"),

                                       tr("Недостаточно средств.\n"

                                          "Повторите попытку!"),

                                       QMessageBox::Ok);

        return;

    }

    auto req = StartCheckRequest{};

    req.DateTime = time(0);

    auto response = \_service->\_\_15\_\_StartCheck(req);

    if (response->ErrorMsg != "") {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr(response->ErrorMsg.c\_str()),

                                   QMessageBox::Ok);

        return;

    } else {

        auto doc = CommonData{};

        doc.UserName = \_regData->user.toStdString();

        doc.Cashier = \_nameCashir.toStdString();

        doc.Address = \_regData->addres.toStdString();

        doc.InnOFD = \_regData->INNoperator.toStdString();

        auto response = \_service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        if (response->ErrorMsg != "") {

            QMessageBox::warning(this, tr("Ошибка"),

                                       tr(response->ErrorMsg.c\_str()),

                                       QMessageBox::Ok);

            return;

        } else {

            auto req = CreateCheckRequest();

            req.DateTime = time(0);

            req.OperationType = OperationType(ui->comboBox\_sell->currentIndex());

            req.Total = \_checkAmount;

            auto response = \_service->\_\_16\_\_CreateCheck(req);

            if (response->ErrorMsg != "") {

                QMessageBox::warning(this, tr("Ошибка"),

                                           tr(response->ErrorMsg.c\_str()),

                                           QMessageBox::Ok);

                return;

            }

            if (ui->comboBox\_sell->currentIndex() == 0 || ui->comboBox\_sell->currentIndex() == 2)

                \_KKMAmount += \_checkAmount; // обновляем сумму всей кассы

            else

                \_KKMAmount -= \_checkAmount; // обновляем сумму всей кассы

            ui->line\_summ->setText(QString::number(\_KKMAmount)); // обновляем сумму кассы

            ui->number\_check->setText(QString::number(response->CheckNum)); // обновляем

            ui->tableWidget\_check->setRowCount(0); // очистка таблицы

            if (ui->comboBox\_CashAmount->currentIndex() == 0) {

                \_cashAmount = \_checkAmount;

                \_KKMcashAmount += \_checkAmount;

            } else {

                \_cashlessAmount = \_checkAmount;

                \_KKMAcashlessAmount += \_checkAmount;

            }

            // добавляем в историю

            addRowinHistory(QString::number(response->CheckNum),

                            ui->comboBox\_sell->currentText(),

                            QString::number(\_posishion\_check),

                            QString::number(\_cashAmount, 'f', 2),

                            QString::number(\_cashlessAmount, 'f', 2),

                            QString::number(\_received, 'f', 2),

                            QString::number(\_change, 'f', 2),

                            QString::number(response->FiscDocNumber),

                            QString::number(response->FiscSign));

            // чистим переменные

            ui->line\_sum\_check->setText("0.00");

            ui->line\_drop\_check->setText("0.00");

            ui->line\_sdacha\_check->setText("0");

            \_posishion\_check = 0;

            \_checkAmount = 0;

            \_cashAmount= 0;

            \_cashlessAmount= 0;

            \_received = 0;

            \_change = 0;

        }

    }

}

void MainWindow::executeCommand(COMMANDS\_FN command) {

    QString message;

    switch (command) {

    case COMMANDS\_FN::\_30h: {

        auto response = \_service->\_\_30\_\_GetFNStatus();

        message += "Команда: 30h - Запрос статуса ФН\n";

        if (response->ErrorMsg != "") {

            message += response->ErrorMsg.c\_str();

            return;

        }

        message += QString("%1%2%3").arg("Состояние фазы жизни: ").arg(response->PhaseOfLife.c\_str()).arg("\n");

        message += QString("%1%2%3").arg("Текущий документ: ").arg(response->CurrentDocument.c\_str()).arg("\n");

        QString documentData = response->DocumentDataRecived ? "данные документа получены" : "нет данных документа";

        message += QString("%1%2%3").arg("Данные документ: ").arg(documentData).arg("\n");

        QString shiftData = response->ShiftIsOpen ? "смена открыта" : "смена закрыта";

        message += QString("%1%2%3").arg("Состояние смены: ").arg(shiftData).arg("\n");

        message += QString("%1%2%3").arg("Флаги предупреждения: ").arg(response->Warnings.c\_str()).arg("\n");

        message += QString("%1%2%3").arg("Дата и время: ").arg(response->DateTime.c\_str()).arg("\n");

        message += QString("%1%2%3").arg("Номер ФН: ").arg(response->Number\_cp866.c\_str()).arg("\n");

        message += QString("%1%2%3").arg("Номер последнего ФД: ").arg(QString::number(response->LastFDNumber)).arg("\n");

        emit this->sendCommandInfo(message);

        break;

    }

    case COMMANDS\_FN::\_31h: {

        auto response = \_service->\_\_31\_\_GetFNNumber();

        message += "Команда: 31h - Запрос номера ФН\n";

        if (response->ErrorMsg != "") {

            message += response->ErrorMsg.c\_str();

            return;

        }

        message += QString("%1%2%3").arg("Номер ФН: ").arg(response->Number\_cp866.c\_str()).arg("\n");

        emit this->sendCommandInfo(message);

        break;

    }

    case COMMANDS\_FN::\_32h: {

        auto response = \_service->\_\_32\_\_GetFNEndDate();

        message += "Команда: 32h - Запрос срока действия ФН\n";

        if (response->ErrorMsg != "") {

            message += response->ErrorMsg.c\_str();

            return;

        }

        message += QString("%1%2%3").arg("Срок действия ФН: ").arg(response->Date.c\_str()).arg("\n");

        emit this->sendCommandInfo(message);

        break;

    }

    case COMMANDS\_FN::\_33h: {

        auto response = \_service->\_\_33\_\_GetFNVersion();

        message += "Команда: 33h - Запрос версии ФН\n";

        if (response->ErrorMsg != "") {

            message += response->ErrorMsg.c\_str();

            return;

        }

        message += QString("%1%2%3").arg("Версия программного обеспечения ФН: ").arg(response->VersionSoftWare\_crc866.c\_str()).arg("\n");

        message += QString("%1%2%3").arg("Тип программного обеспечения ФН: ").arg(response->TypeSoftWare.c\_str()).arg("\n");

        emit this->sendCommandInfo(message);

        break;

    }

    case COMMANDS\_FN::\_10h: {

        auto response = \_service->\_\_10\_\_GetShiftStatus();

        message += "Команда: 10h - Запрос параметров текущей смены\n";

        if (response->ErrorMsg != "") {

            message += response->ErrorMsg.c\_str();

            return;

        }

        message += QString("%1%2%3").arg("Номер смены: ").arg(QString::number(response->ShiftNum)).arg("\n");

        QString shiftData = response->ShiftOpen ? "смена открыта" : "смена закрыта";

        message += QString("%1%2%3").arg("Состояние смены: ").arg(shiftData).arg("\n");

        message += QString("%1%2%3").arg("Номер чека: ").arg(QString::number(response->CheckAmmount)).arg("\n");

        emit this->sendCommandInfo(message);

        break;

    }

    }

}

void MainWindow::on\_comboBox\_sell\_currentIndexChanged(int index) {

    if (index == 0 || index == 3)

        ui->label\_pol\_vid->setText("Получено");

    else if (index == 1 || index == 2)

        ui->label\_pol\_vid->setText("Выдано");

}

void MainWindow::on\_setting\_prog\_clicked() {

    \_settingProgW->show();

}

void MainWindow::on\_setting\_kkm\_clicked() {

    \_settingKKMW->show();

}

void MainWindow::on\_other\_commands\_clicked() {

    \_commandW->show();

}

void MainWindow::recieveProgSetting(size\_t method\_calc, size\_t subject\_calc,

                    size\_t nds, size\_t quantity) {

    \_method\_calc = method\_calc;

    \_subject\_calc = subject\_calc;

    \_nds = nds;

    \_quantity = quantity;

    ui->label\_is\_pr->setText(METHOD\_OF\_CALC[\_method\_calc]);

    ui->label\_subject->setText(SUBJECT\_OF\_CALC[\_subject\_calc]);

    ui->label\_is\_nds->setText(NDS[\_nds]);

}

void MainWindow::setupModel(const QString &tableName, const QStringList &headers, QSqlTableModel \*model)

{

    model->setTable(tableName);

    for(int i = 0, j = 0; i < model->columnCount(); i++, j++){

        model->setHeaderData(i,Qt::Horizontal,headers[j]);

    }

    model->setSort(0,Qt::AscendingOrder);

}

void MainWindow::createUI(QTableView \*t, QSqlTableModel  \*model)

{

    t->setModel(model);     // Устанавливаем модель на TableView

    t->verticalHeader()->setVisible(false);

    // Разрешаем выделение строк

    t->setSelectionBehavior(QAbstractItemView::SelectRows);

    // Устанавливаем режим выделения лишь одно строки в таблице

    t->setSelectionMode(QAbstractItemView::SingleSelection);

    // Устанавливаем размер колонок по содержимому

    t->resizeColumnsToContents();

    t->setEditTriggers(QAbstractItemView::NoEditTriggers);

    ui->productDB->horizontalHeader()->setSectionResizeMode(2, QHeaderView::Stretch);

    ui->historyDB->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

    //ui\_->productDB->horizontalHeader()->resizeSections(QHeaderView::ResizeToContents);

    model->select(); // Делаем выборку данных из таблицы

}

void MainWindow::addRowinCheck(const QString &name, float prise) {

    int newRow = ui->tableWidget\_check->rowCount();

    for (auto i = 0; i < newRow; i++) { // проверяем есть ли данный товар в чеке

        if (ui->tableWidget\_check->item(i, 0)->data(Qt::DisplayRole).toString() == name) {

            auto count = ui->tableWidget\_check->item(i, 2)->data(Qt::DisplayRole).toInt() + 1;

            auto endPrise = ui->tableWidget\_check->item(i, 3)->data(Qt::DisplayRole).toFloat() + prise;

            ui->tableWidget\_check->setItem(i, 2, new QTableWidgetItem(QString::number(count)));

            ui->tableWidget\_check->setItem(i, 3, new QTableWidgetItem(QString::number(endPrise)));

            return;

        }

    }

    ui->tableWidget\_check->insertRow(newRow);

    ui->tableWidget\_check->setItem(newRow, 0, new QTableWidgetItem(name));

    ui->tableWidget\_check->setItem(newRow, 1, new QTableWidgetItem(QString::number(prise)));

    ui->tableWidget\_check->setItem(newRow, 2, new QTableWidgetItem(QString::number(1)));

    ui->tableWidget\_check->setItem(newRow, 3, new QTableWidgetItem(QString::number(prise)));

}

void MainWindow::addRowinHistory(const QString &numberCheck,

                                 const QString &operation,

                                 const QString &pos,

                                 const QString &cashAmount,

                                 const QString &cashlessAmount,

                                 const QString &received,

                                 const QString &change,

                                 const QString &fiscDoc,

                                 const QString &fiscSign) {

    QVariantList data;

    data.append(QDateTime::currentDateTime().toString("dd.MM.yyyy HH:mm:ss"));

    data.append(numberCheck);

    data.append(operation);

    data.append(pos);

    data.append(cashAmount);

    data.append(cashlessAmount);

    data.append(received);

    data.append(change);

    data.append(fiscDoc);

    data.append(fiscSign);

    \_hdb->inserIntoTable(data);

    \_modelH->select();

}

void MainWindow::on\_productDB\_doubleClicked(const QModelIndex &index) {

    auto name = \_modelP->data(\_modelP->index(index.row(), 2)).toString();

    auto prise = \_modelP->data(\_modelP->index(index.row(), 3)).toFloat();

    \_checkAmount += prise;

    ++\_posishion\_check; //увеличиваем количество позиций\_

    addRowinCheck(name, prise);

    auto newPrise = QString::number(\_checkAmount);

    ui->line\_sum\_check->setText(newPrise);

}

void MainWindow::on\_line\_drop\_check\_editingFinished() {

    \_received = ui->line\_drop\_check->text().toFloat();

    if (\_checkAmount < \_received) {

            \_change = \_received - \_checkAmount; // считаем сдачу

            ui->line\_sdacha\_check->setText(QString::number(\_change));

    }

}

void MainWindow::on\_tabWidget\_currentChanged(int index) {

    qDebug() << "окно" << index;

    if (index == 3) {

        \_loginW->clearData();

        \_loginW->show();

    } else {

        ui->inspect->setEnabled(false);

        \_loginW->hide();

    }

}

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QDebug>

#include <QSqlTableModel>

#include <QTableView>

#include <settingprogwindow.h>

#include <settingkkmwindow.h>

#include <loginoperatorofd.h>

#include <registrationwindow.h>

#include <commandwindow.h>

#include <productdb.h>

#include <historydb.h>

#include <memory>

#include <ComReader/utils.h>

#include <ComReader/Hardware.h>

#include <ComReader/TLV.h>

const QStringList STATUS\_KKM = {"Не готова", "Готова"};

const QStringList STATUS\_FN = {"Недоступна", "Готовность к фискализации", "Фискальный режим", "Фискальный режим закрыт"};

const QStringList STATUS\_SHIFT = {"Не открыта", "Открыта", "Закрыта"};

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow {

    Q\_OBJECT

public:

    MainWindow(QWidget \*parent = nullptr);

    ~MainWindow();

    void setProgSetting(size\_t method\_calc, size\_t subject\_calc,

                        size\_t nds, size\_t quantity);

signals:

    void sendCommandInfo(const QString &);

private slots:

    void recieveProgSetting(size\_t method\_calc, size\_t subject\_calc,

                        size\_t nds, size\_t quantity);

    bool initKKMSetting(const QString &name, const QString &port);

    void registration(const RegistrationData \*data);

    void executeCommand(COMMANDS\_FN c);

    void checkLoginFNS(const QString &login, const QString &pass);

    void on\_push\_exit\_clicked();

    void on\_comboBox\_sell\_currentIndexChanged(int index);

    void on\_setting\_prog\_clicked();

    void on\_get\_check\_clicked();

    void on\_productDB\_doubleClicked(const QModelIndex &index);

    void on\_line\_drop\_check\_editingFinished();

    void on\_setting\_kkm\_clicked();

    void on\_buttonFiscal\_clicked();

    void on\_open\_shift\_clicked();

    void on\_close\_shift\_clicked();

    void on\_tabWidget\_currentChanged(int index);

    void on\_other\_commands\_clicked();

    void on\_buttonFiscalEnd\_clicked();

private:

    Ui::MainWindow \*ui;

    SettingProgWindow \*\_settingProgW;

    SettingKKMWindow \*\_settingKKMW;

    LoginOperatorOFD \*\_loginW;

    RegistrationWindow \*\_regW;

    CommandWindow \* \_commandW;

    // Свзязь с proteus

    std::shared\_ptr<Hardware> \_service;

    // имя кассира

    QString \_nameCashir;

    RegistrationData \* \_regData;

    bool \_isReg;

    // настройки ккм

    QString \_port;

    // настройки программы

    size\_t \_method\_calc;

    size\_t \_subject\_calc;

    size\_t \_nds;

    size\_t \_quantity;

    size\_t \_count\_check;

    size\_t \_posishion\_check;

    float \_KKMAmount;          // сумма кассы

    float \_KKMcashAmount;      // сумма кассы

    float \_KKMAcashlessAmount; // сумма кассы

    float \_cashAmount;         // сумма кассы нал

    float \_cashlessAmount;     // сумма кассы безнал

    float \_checkAmount;        // сумма чека

    float \_received;           // полученная сумма

    float \_change;             // сдача

    // БД товаров, истории

    ProductDB \*\_pdb;

    HistoryDB \*\_hdb;

    QSqlTableModel  \*\_modelP;

    QSqlTableModel  \*\_modelH;

    void initPDB();

    void initHDB();

    void setupModel(const QString &tableName, const QStringList &headers, QSqlTableModel \*model);

    void createUI(QTableView \*t, QSqlTableModel \*model);

    void initKKMWindow();

    void initNewChekWindow();

    void initInspectWindow();

    void initRegistrWindow();

    void initCommandWindow();

    void addRowinCheck(const QString &name, float prise);

    void addRowinHistory(const QString &numberCheck,

                         const QString &operation,

                         const QString &pos,

                         const QString &cashAmount,

                         const QString &cashlessAmount,

                         const QString &received,

                         const QString &change,

                         const QString &fiscDoc,

                         const QString &fiscSign);

};

#endif // MAINWINDOW\_H

#include "productdb.h"

ProductDB::ProductDB(QObject \*parent) : QObject(parent) {

}

/\* Методы для подключения к базе данных

 \* \*/

void ProductDB::connectToDataBase() {

    /\* Перед подключением к базе данных производим проверку на её существование.

     \* В зависимости от результата производим открытие базы данных или её восстановление

     \* \*/

    if (!QFile("C:/Users/Sergei/Documents/KKM/" DATABASE\_NAME).exists()){

        this->restoreDataBase();

    } else {

        this->openDataBase();

    }

}

/\* Методы восстановления базы данных

 \* \*/

bool ProductDB::restoreDataBase() {

    if(this->openDataBase()){

        if(!this->createTable()){

            return false;

        } else {

            return true;

        }

    } else {

        qDebug() << "Не удалось восстановить базу данных";

        return false;

    }

    return false;

}

/\* Метод для открытия базы данных

 \* \*/

bool ProductDB::openDataBase()

{

    /\* База данных открывается по заданному пути

     \* и имени базы данных, если она существует

     \* \*/

    db = QSqlDatabase::addDatabase("QSQLITE");

    db.setHostName(DATABASE\_HOSTNAME);

    db.setDatabaseName("C:/Users/Sergei/Documents/KKM/" DATABASE\_NAME);

    if(db.open()){

        return true;

    } else {

        return false;

    }

}

/\* Методы закрытия базы данных

 \* \*/

void ProductDB::closeDataBase()

{

    db.close();

}

/\* Метод для создания таблицы в базе данных

 \* \*/

bool ProductDB::createTable()

{

    /\* В данном случае используется формирование сырого SQL-запроса

     \* с последующим его выполнением.

     \* \*/

    QSqlQuery query;

    if(!query.exec( "CREATE TABLE " TABLE\_PRODUCT " ("

                            TABLE\_ID        " INTEGER         NOT NULL,"

                            TABLE\_BARCODE   " VARCHAR(255)    NOT NULL,"

                            TABLE\_NAME      " VARCHAR(255)    NOT NULL,"

                            TABLE\_PRICE     " FLOAT           NOT NULL"

                        " )"

                    )){

        qDebug() << "DataBase: error of create " << TABLE\_PRODUCT;

        qDebug() << query.lastError().text();

        return false;

    } else {

        return true;

    }

    return false;

}

/\* Метод для вставки записи в базу данных"

""

""

 \* \*/

bool ProductDB::inserIntoTable(const QVariantList &data)

{

    /\* Запрос SQL формируется из QVariantList,

     \* в который передаются данные для вставки в таблицу.

     \* \*/

    QSqlQuery query;

    /\* В начале SQL запрос формируется с ключами,

     \* которые потом связываются методом bindValue

     \* для подстановки данных из QVariantList

     \* \*/

    query.prepare("INSERT INTO " TABLE\_PRODUCT " ( " TABLE\_ID ", "

                                                     TABLE\_NAME ", "

                                                     TABLE\_BARCODE ", "

                                                     TABLE\_PRICE " ) "

                  "VALUES (:ID, :Barcode, :Name, :Prise )");

    query.bindValue(":ID",       data[0].toInt());

    query.bindValue(":Barcode",  data[1].toString());

    query.bindValue(":Name",     data[2].toString());

    query.bindValue(":Prise",    data[3].toFloat());

    // После чего выполняется запросом методом exec()

    if(!query.exec()){

        qDebug() << "error insert into " << TABLE\_PRODUCT;

        qDebug() << query.lastError().text();

        return false;

    } else {

        return true;

    }

    return false;

}

#ifndef PRODUCTDB\_H

#define PRODUCTDB\_H

#include <QObject>

#include <QSql>

#include <QSqlQuery>

#include <QSqlError>

#include <QSqlDatabase>

#include <QFile>

#include <QDate>

#include <QDebug>

/\* Директивы имен таблицы, полей таблицы и базы данных \*/

#define DATABASE\_HOSTNAME   "ProdDB"

#define DATABASE\_NAME       "product.db"

#define TABLE\_PRODUCT           "TableProduct"

#define TABLE\_ID                "ID"

#define TABLE\_BARCODE           "Barcode"

#define TABLE\_NAME              "Name"

#define TABLE\_PRICE             "Prise"

class ProductDB : public QObject

{

    Q\_OBJECT

public:

    explicit ProductDB(QObject \*parent = 0);

    ~ProductDB() = default;

    /\* Методы для непосредственной работы с классом

     \* Подключение к базе данных и вставка записей в таблицу

     \* \*/

    void connectToDataBase();

    bool inserIntoTable(const QVariantList &data);

private:

    // Сам объект базы данных, с которым будет производиться работа

    QSqlDatabase    db;

private:

    /\* Внутренние методы для работы с базой данных

     \* \*/

    bool openDataBase();

    bool restoreDataBase();

    void closeDataBase();

    bool createTable();

};

#endif // PRODUCTDB\_H

#include "registrationwindow.h"

#include "ui\_registrationwindow.h"

RegistrationWindow::RegistrationWindow(QWidget \*parent) :

    QWidget(parent),

    ui(new Ui::RegistrationWindow),

    regData(new RegistrationData) {

    ui->setupUi(this);

    connect(ui->pushButton\_reg, SIGNAL(clicked()), this, SLOT(sendData()));

}

RegistrationWindow::~RegistrationWindow() {

    delete ui;

    delete regData;

}

void RegistrationWindow::on\_pushButton\_cansel\_clicked() {

    this->hide();

}

bool RegistrationWindow::checkData() {

    if (!ui->user->text().isEmpty() && !ui->addr->text().isEmpty() &&

        !ui->place->text().isEmpty() && !ui->regNumber->text().isEmpty() &&

        !ui->inn->text().isEmpty() && !ui->sitefns->text().isEmpty() &&

        !ui->nameOperator->text().isEmpty() && !ui->innOperator->text().isEmpty() &&

        !ui->ip->text().isEmpty() && !ui->port->text().isEmpty()) {

        if (ui->NalogCode1->isChecked() || ui->NalogCode2->isChecked() ||

            ui->NalogCode4->isChecked() || ui->NalogCode8->isChecked() ||

            ui->NalogCode16->isChecked() || ui->NalogCode32->isChecked()) {

            if (ui->WorkMode1->isChecked() || ui->WorkMode2->isChecked() ||

                ui->WorkMode4->isChecked() || ui->WorkMode8->isChecked() ||

                ui->WorkMode16->isChecked() || ui->WorkMode32->isChecked()) {

                return true;

            }

        }

    }

    return false;

}

void RegistrationWindow::sendData() {

    if (!checkData()) {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("Заполнены не все поля!"),

                                   QMessageBox::Ok);

        return;

    } else {

        regData->user = ui->user->text();

        regData->addres = ui->addr->text();

        regData->placeSettlement = ui->place->text();

        regData->regNumberKKT = ui->regNumber->text();

        regData->INN = ui->inn->text();

        regData->siteFNS = ui->sitefns->text();

        regData->operatorOFD = ui->nameOperator->text();

        regData->INNoperator = ui->innOperator->text();

        regData->IP = ui->ip->text();

        regData->port = ui->port->text();

        if (ui->NalogCode1->isChecked()) regData->nc = NalogCode::COMMON;

        if (ui->NalogCode2->isChecked()) regData->nc = NalogCode::SIMPLE\_INCOME;

        if (ui->NalogCode4->isChecked()) regData->nc = NalogCode::SIMPLE\_INCOME\_MINUS\_EXPENSE;

        if (ui->NalogCode8->isChecked()) regData->nc = NalogCode::SINGLE\_TAX\_ON\_IMPUTED\_INCOME;

        if (ui->NalogCode16->isChecked()) regData->nc = NalogCode::UNIFIED\_AGRICULTURAL\_TAX;

        if (ui->NalogCode32->isChecked()) regData->nc = NalogCode::PATENT\_TAXATION\_SYSTEM;

        if (ui->WorkMode1->isChecked()) regData->wm = WorkMode::ENCRYPTION;

        if (ui->WorkMode2->isChecked()) regData->wm = WorkMode::OFFLINE;

        if (ui->WorkMode4->isChecked()) regData->wm = WorkMode::AUTO;

        if (ui->WorkMode8->isChecked()) regData->wm = WorkMode::SERVICE\_APPLICATIONS;

        if (ui->WorkMode16->isChecked()) regData->wm = WorkMode::CHECK;

        if (ui->WorkMode32->isChecked()) regData->wm = WorkMode::INTERNET\_COMMERCE;

        emit this->saveData(regData);

        this->hide();

        QMessageBox::information(this, tr("Сообщение"),

                                   tr("Регистрация прошла успешно!"),

                                   QMessageBox::Ok);

    }

}

void RegistrationWindow::on\_toolButton\_clicked() {

    ui->user->setText("OOO \"Курсовая\"");

    ui->addr->setText("г. Москва");

    ui->place->setText("Там где сделанный курсач");

    ui->regNumber->setText("KKT-772-233-445-566");

    ui->inn->setText("409023738");

    ui->sitefns->setText("www.nalog.ru");

    ui->NalogCode1->setChecked(true);

    ui->WorkMode1->setChecked(true);

    ui->nameOperator->setText("ЗАО \"Панимаю\"");

    ui->innOperator->setText("4029013781");

    ui->ip->setText("127.0.0.1");

    ui->port->setText("4321");

}

#ifndef REGISTRATIONWINDOW\_H

#define REGISTRATIONWINDOW\_H

#include <QWidget>

#include <QMessageBox>

#include <ComReader/Hardware.h>

struct RegistrationData {

    QString user;

    QString addres;

    QString placeSettlement;

    QString regNumberKKT;

    QString INN;

    QString siteFNS;

    NalogCode nc;

    WorkMode wm;

    QString operatorOFD;

    QString INNoperator;

    QString IP;

    QString port;

}; // RegistrationData

namespace Ui {

class RegistrationWindow;

}

class RegistrationWindow : public QWidget

{

    Q\_OBJECT

public:

    explicit RegistrationWindow(QWidget \*parent = nullptr);

    ~RegistrationWindow();

signals:

    void saveData(const RegistrationData \*data);

private slots:

    void sendData();

    void on\_pushButton\_cansel\_clicked();

    void on\_toolButton\_clicked();

private:

    Ui::RegistrationWindow \*ui;

    RegistrationData \*regData;

    bool checkData();

};

#endif // REGISTRATIONWINDOW\_H

#include "settingkkmwindow.h"

#include "ui\_settingkkmwindow.h"

SettingKKMWindow::SettingKKMWindow(QWidget \*parent) :

    QWidget(parent),

    ui(new Ui::SettingKKMWindow) {

    ui->setupUi(this);

    initCB();

}

void SettingKKMWindow::initCB() {

    ui->cb\_model->addItem("Модель ККМ");

    ui->cb\_link->addItems({"USB", "COM"});

    ui->cb\_com->addItems({"COM1", "COM2", "COM3"});

    ui->cb\_speed->addItems({"115200"});

    ui->cb\_bit->addItems({"8", "9"});

    ui->cb\_paraty->addItems({"Нет", "Четно", "Нечетно"});

    ui->cb\_stopbit->addItems({"1 бит", "2 бита"});

    connect(ui->pushButton\_ok, SIGNAL(clicked()), this, SLOT(sendSetting()));

}

SettingKKMWindow::~SettingKKMWindow() {

    delete ui;

}

void SettingKKMWindow::on\_pushButton\_cansel\_clicked() {

    this->hide();

}

void SettingKKMWindow::sendSetting() {

    if (ui->cashir->text().isEmpty()) {

        QMessageBox::warning(this, tr("Ошибка"),

                                   tr("Введите имя кассира!"),

                                   QMessageBox::Ok);

    } else {

        emit this->saveSetting(ui->cashir->text(), ui->cb\_com->currentText());

        this->hide();

    }

}

void SettingKKMWindow::on\_cb\_link\_currentIndexChanged(int index) {

    if (index == 0) {

        ui->cb\_com->setEnabled(false);

        ui->cb\_speed->setEnabled(false);

        ui->cb\_bit->setEnabled(false);

        ui->cb\_paraty->setEnabled(false);

        ui->cb\_stopbit->setEnabled(false);

    } else {

        ui->cb\_com->setEnabled(true);

        ui->cb\_speed->setEnabled(true);

        ui->cb\_bit->setEnabled(true);

        ui->cb\_paraty->setEnabled(true);

        ui->cb\_stopbit->setEnabled(true);

    }

}

#ifndef SETTINGKKMWINDOW\_H

#define SETTINGKKMWINDOW\_H

#include <QWidget>

#include <QMessageBox>

namespace Ui {

class SettingKKMWindow;

}

class SettingKKMWindow : public QWidget {

    Q\_OBJECT

public:

    explicit SettingKKMWindow(QWidget \*parent = nullptr);

    ~SettingKKMWindow();

signals:

    void saveSetting(const QString &name, const QString &port);

private slots:

    void sendSetting();

    void on\_pushButton\_cansel\_clicked();

    void on\_cb\_link\_currentIndexChanged(int index);

private:

    Ui::SettingKKMWindow \*ui;

    void initCB();

};

#endif // SETTINGKKMWINDOW\_H

#include "settingprogwindow.h"

#include "ui\_settingprogwindow.h"

SettingProgWindow::SettingProgWindow(QWidget \*parent) :

    QWidget(parent),

    ui(new Ui::SettingProgWindow) {

    ui->setupUi(this);

    init();

    connect(ui->pushButton\_ok, SIGNAL(clicked()), this, SLOT(sendSetting()));

}

SettingProgWindow::~SettingProgWindow() {

    delete ui;

}

void SettingProgWindow::init() {

    ui->comboBox\_way->addItems(METHOD\_OF\_CALC);

    ui->comboBox\_think->addItems(SUBJECT\_OF\_CALC);

    ui->comboBox\_nds->addItems(NDS);

    ui->comboBox\_template->addItems(QUANTITY\_PATTERN);

}

void SettingProgWindow::setDefaultSetting(size\_t method\_calc,

                                          size\_t subject\_calc,

                                          size\_t nds,

                                          size\_t quantity) {

    ui->comboBox\_way->setCurrentIndex(method\_calc);

    ui->comboBox\_think->setCurrentIndex(subject\_calc);

    ui->comboBox\_nds->setCurrentIndex(nds);

    ui->comboBox\_template->setCurrentIndex(quantity);

}

void SettingProgWindow::on\_pushButton\_cansel\_clicked() {

    this->hide();

}

void SettingProgWindow::sendSetting() {

    emit this->saveSetting(ui->comboBox\_way->currentIndex(),

                           ui->comboBox\_think->currentIndex(),

                           ui->comboBox\_nds->currentIndex(),

                           ui->comboBox\_template->currentIndex());

    this->hide();

}

#ifndef SETTINGPROGWINDOW\_H

#define SETTINGPROGWINDOW\_H

#include <QWidget>

const QStringList METHOD\_OF\_CALC = {

    "Полный расчет",

    "Предоплата",

    "Аванс",

    "Частич.расчет"

};

const QStringList SUBJECT\_OF\_CALC = {

    "Товар",

    "Подакциз.товар",

    "Работа",

    "Услуга",

    "Платеж"

};

const QStringList NDS = {

    "Без НДС",

    "0%",

    "10%",

    "20%"

};

const QStringList QUANTITY\_PATTERN = {

    "999999"

};

namespace Ui {

class SettingProgWindow;

}

class SettingProgWindow : public QWidget {

    Q\_OBJECT

public:

    explicit SettingProgWindow(QWidget \*parent = nullptr);

    ~SettingProgWindow();

    void setDefaultSetting(size\_t method\_calc, size\_t subject\_calc,

                           size\_t nds, size\_t quantity);

signals:

    void saveSetting(size\_t method\_calc, size\_t subject\_calc,

                     size\_t nds, size\_t quantity);

private slots:

    void sendSetting();

    void on\_pushButton\_cansel\_clicked();

private:

    void init();

    Ui::SettingProgWindow \*ui;

};

#endif // SETTINGPROGWINDOW\_H

#include "loginoperatorofd.h"

#include "ui\_loginoperatorofd.h"

LoginOperatorOFD::LoginOperatorOFD(QWidget \*parent) :

    QWidget(parent),

    ui(new Ui::LoginOperatorOFD),

    counter(0) {

    ui->setupUi(this);

    connect(ui->pushButton\_ok, SIGNAL(clicked()), this, SLOT(checklogin()));

}

LoginOperatorOFD::~LoginOperatorOFD() {

    delete ui;

}

void LoginOperatorOFD::clearData() {

    ui->login->clear();

    ui->password->clear();

    ui->labelMess->clear();

}

void LoginOperatorOFD::checklogin() {

    counter++;

    if (counter > 2) {

        counter = 0;

        this->hide();

    } else {

        if (ui->login->text() != "admin" || ui->password->text() != "admin")

            ui->labelMess->setText("Неверный логин или пароль");

        emit this->login(ui->login->text(), ui->password->text());

    }

}

void LoginOperatorOFD::on\_pushButton\_close\_clicked() {

    this->hide();

}

#ifndef LOGINOPERATOROFD\_H

#define LOGINOPERATOROFD\_H

#include <QWidget>

namespace Ui {

class LoginOperatorOFD;

}

class LoginOperatorOFD : public QWidget

{

    Q\_OBJECT

public:

    explicit LoginOperatorOFD(QWidget \*parent = nullptr);

    ~LoginOperatorOFD();

    void clearData();

signals:

    void login(const QString &, const QString &);

private slots:

    void checklogin();

    void on\_pushButton\_close\_clicked();

private:

    Ui::LoginOperatorOFD \*ui;

    size\_t counter;

};

#endif // LOGINOPERATOROFD\_H

#include "historydb.h"

HistoryDB::HistoryDB(QObject \*parent) : QObject(parent)

{

}

/\* Методы для подключения к базе данных

 \* \*/

void HistoryDB::connectToDataBase()

{

    if(!QFile("C:/Users/Sergei/Documents/KKM/" DATABASE\_NAME).exists()){

        this->restoreDataBase();

    } else {

        this->openDataBase();

    }

}

/\* Методы восстановления базы данных

 \* \*/

bool HistoryDB::restoreDataBase()

{

    if(this->openDataBase()){

        if(!this->createTable()){

            return false;

        } else {

            return true;

        }

    } else {

        qDebug() << "Не удалось восстановить базу данных";

        return false;

    }

    return false;

}

/\* Метод для открытия базы данных

 \* \*/

bool HistoryDB::openDataBase()

{

    db = QSqlDatabase::addDatabase("QSQLITE");

    db.setHostName(DATABASE\_HOSTNAME);

    db.setDatabaseName("C:/Users/Sergei/Documents/KKM/" DATABASE\_NAME);

    if(db.open()){

        return true;

    } else {

        return false;

    }

}

/\* Методы закрытия базы данных

 \* \*/

void HistoryDB::closeDataBase()

{

    db.close();

}

/\* Метод для создания таблицы в базе данных

 \* \*/

bool HistoryDB::createTable()

{

    QSqlQuery query;

    QString temp = "CREATE TABLE " TABLE\_HISTORY " ("

            TABLE\_CREATE          " VARCHAR(255)    NOT NULL,"

            TABLE\_CHECK\_NUMBER    " VARCHAR(255)    NOT NULL,"

            TABLE\_OPERATION       " VARCHAR(255)    NOT NULL,"

            TABLE\_POSITIONS       " VARCHAR(255)    NOT NULL,"

            TABLE\_CASH\_AMOUNT     " VARCHAR(255)    NOT NULL,"

            TABLE\_CASHLESS\_AMOUNT " VARCHAR(255)    NOT NULL,"

            TABLE\_RECEIVED        " VARCHAR(255)    NOT NULL,"

            TABLE\_CHANGE          " VARCHAR(255)    NOT NULL,"

            TABLE\_FISC\_DOC        " VARCHAR(255)    NOT NULL,"

            TABLE\_FISC\_SIGN       " VARCHAR(255)    NOT NULL"

        " )";

    if (!query.exec(temp)) {

        qDebug() <<temp;

        qDebug() << "DataBase: error of create " << TABLE\_HISTORY;

        qDebug() << query.lastError().text();

        return false;

    } else {

        return true;

    }

    return false;

}

/\* Метод для вставки записи в базу данных

 \* \*/

bool HistoryDB::inserIntoTable(const QVariantList &data)

{

    QSqlQuery query;

    query.prepare("INSERT INTO " TABLE\_HISTORY " ( " TABLE\_CREATE ", "

                                             TABLE\_CHECK\_NUMBER ", "

                                             TABLE\_OPERATION ", "

                                             TABLE\_POSITIONS ", "

                                             TABLE\_CASH\_AMOUNT ", "

                                             TABLE\_CASHLESS\_AMOUNT ", "

                                             TABLE\_RECEIVED ", "

                                             TABLE\_CHANGE ", "

                                             TABLE\_FISC\_DOC ", "

                                             TABLE\_FISC\_SIGN " ) "

                  "VALUES (:TimeCreate, :CheckNumber, :Operation, :Positions, "

                  ":CashAmount, :CashlessAmount, :Received, :Change, :FiscDoc, :FiscSign )");

    query.bindValue(":TimeCreate",     data[0].toString());

    query.bindValue(":CheckNumber",    data[1].toString());

    query.bindValue(":Operation",      data[2].toString());

    query.bindValue(":Positions",      data[3].toString());

    query.bindValue(":CashAmount",     data[4].toString());

    query.bindValue(":CashlessAmount", data[5].toString());

    query.bindValue(":Received",       data[6].toString());

    query.bindValue(":Change",         data[7].toString());

    query.bindValue(":FiscDoc",        data[8].toString());

    query.bindValue(":FiscSign",       data[9].toString());

    if(!query.exec()){

        qDebug() << "error insert into " << TABLE\_HISTORY;

        qDebug() << query.lastError().text();

        return false;

    } else {

        return true;

    }

    return false;

}

#ifndef HISTORYDB\_H

#define HISTORYDB\_H

#include <QObject>

#include <QSql>

#include <QSqlQuery>

#include <QSqlError>

#include <QSqlDatabase>

#include <QFile>

#include <QDate>

#include <QDebug>

/\* Директивы имен таблицы, полей таблицы и базы данных \*/

#define DATABASE\_HOSTNAME   "HistoryDB"

#define DATABASE\_NAME       "history.db"

#define TABLE\_HISTORY           "HistoryDB"

#define TABLE\_CREATE            "TimeCreate"

#define TABLE\_CHECK\_NUMBER      "CheckNumber"

#define TABLE\_OPERATION         "Operation"

#define TABLE\_POSITIONS         "Positions"

#define TABLE\_CASH\_AMOUNT       "CashAmount"

#define TABLE\_CASHLESS\_AMOUNT   "CashlessAmount"

#define TABLE\_RECEIVED          "Received"

#define TABLE\_CHANGE            "Change"

#define TABLE\_FISC\_DOC          "FiscDoc"

#define TABLE\_FISC\_SIGN         "FiscSign"

class HistoryDB : public QObject

{

    Q\_OBJECT

public:

    explicit HistoryDB(QObject \*parent = 0);

    ~HistoryDB() = default;

    /\* Методы для непосредственной работы с классом

     \* Подключение к базе данных и вставка записей в таблицу

     \* \*/

    void connectToDataBase();

    bool inserIntoTable(const QVariantList &data);

private:

    // Сам объект базы данных, с которым будет производиться работа

    QSqlDatabase db;

private:

    /\* Внутренние методы для работы с базой данных

     \* \*/

    bool openDataBase();

    bool restoreDataBase();

    void closeDataBase();

    bool createTable();

};

#endif // HISTORYDB\_H

#include "commandwindow.h"

#include "ui\_commandwindow.h"

CommandWindow::CommandWindow(QWidget \*parent) :

    QWidget(parent),

    ui(new Ui::CommandWindow) {

    ui->setupUi(this);

    initCommands();

    connect(ui->pushButton\_request, SIGNAL(clicked()), this, SLOT(sendRequest()));

}

CommandWindow::~CommandWindow() {

    delete ui;

}

void CommandWindow::initCommands() {

    ui->cbCommandFN->addItems({"30h - Запрос статуса ФН",

                               "31h - Запрос номера ФН",

                               "32h - Запрос срока действия ФН",

                               "33h - Запрос версии ФН",

                               "10h - Запрос параметров текущей смены"});

}

void CommandWindow::on\_pushButtonClose\_clicked() {

    this->hide();

}

void CommandWindow::sendRequest() {

    emit this->request(COMMANDS\_FN(ui->cbCommandFN->currentIndex()));

}

void CommandWindow::printInfo(const QString &info) {

    ui->textBrowser->clear();

    ui->textBrowser->setText(info);

}

void CommandWindow::on\_pushButton\_clear\_clicked() {

    ui->textBrowser->clear();

}

#ifndef COMMANDWINDOW\_H

#define COMMANDWINDOW\_H

#include <QWidget>

#include <ComReader/Hardware.h>

#include <iostream>

enum class COMMANDS\_FN {

    \_30h,

    \_31h,

    \_32h,

    \_33h,

    \_10h

};

namespace Ui {

class CommandWindow;

}

class CommandWindow : public QWidget

{

    Q\_OBJECT

public:

    explicit CommandWindow(QWidget \*parent = nullptr);

    ~CommandWindow();

signals:

    void request(COMMANDS\_FN command);

private slots:

    void sendRequest();

    void printInfo(const QString &info);

    void on\_pushButtonClose\_clicked();

    void on\_pushButton\_clear\_clicked();

private:

    Ui::CommandWindow \*ui;

    void initCommands();

};

#endif // COMMANDWINDOW\_H

#include "com\_utils.h"

#include <QDebug>

com\_utils::com\_utils(int baud\_rate) {

    init(baud\_rate);

}

void com\_utils::init(int baud\_rate) {

    LPCTSTR sPortName = L"COM2";

    hSerial = ::CreateFile(sPortName, GENERIC\_READ | GENERIC\_WRITE,

                           0, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

    if (hSerial == INVALID\_HANDLE\_VALUE) {

        if (GetLastError() == ERROR\_FILE\_NOT\_FOUND) {

            std::cout << "serial port does not exist.\n";

        }

        std::cout << "some other error occurred.\n";

    }

    DCB dcbSerialParams = {0};

    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

    if (!GetCommState(hSerial, &dcbSerialParams)) {

        std::cout << "getting state error\n";

    }

    dcbSerialParams.BaudRate = baud\_rate;

    dcbSerialParams.ByteSize = 8;

    dcbSerialParams.StopBits = ONESTOPBIT;

    dcbSerialParams.Parity = NOPARITY;

    if(!SetCommState(hSerial, &dcbSerialParams)) {

        std::cout << "error setting serial port state\n";

    }

}

bool com\_utils::send\_data(const std::vector<byte> &data) {

    DWORD dwSize = data.size();

    DWORD dwBytesWritten;

    char buffer[1031];

    std::copy(data.begin(), data.end(), buffer);

    BOOL iRet = WriteFile(hSerial, buffer, dwSize, &dwBytesWritten, NULL);

    return iRet;

}

byte com\_utils::ReadByteCOM() {

    DWORD iSize;

    char sReceivedChar;

    while (true)

    {

       ReadFile(hSerial, &sReceivedChar, 1, &iSize, 0);  // получаем 1 байт

       if (iSize > 0)   // если что-то принято, выводим

           qDebug() << sReceivedChar;

    }

    return sReceivedChar;

}

std::vector<byte> com\_utils::read\_data() {

    std::vector<byte> data;

    byte res = 0;

    res = ReadByteCOM();

    return data;

}

#ifndef COM\_UTILS\_H

#define COM\_UTILS\_H

#include <stdio.h>

#include <tchar.h>

#include <Windows.h>

#include <iostream>

#include <vector>

#include <string>

class com\_utils {

public:

    com\_utils(int baud\_rate);

    bool send\_data(const std::vector<byte> &data);

    std::vector<byte> read\_data();

private:

    HANDLE hSerial;

    void init(int baud\_rate);

    byte ReadByteCOM();

};

#endif // COM\_UTILS\_H

1.2 Текст Связной части с аппаратурой

// Leonid Moguchev (c) 2020

#include "ComChannel.h"

ComChannel::ComChannel(const std::wstring& port\_name, uint32\_t baud\_rate)

    : \_hSerial(nullptr), \_connected(false),

    \_port\_name(port\_name), \_baud\_rate(baud\_rate)

{}

ComChannel::~ComChannel() {

    if (!close()) {

        std::cerr << "potential memory leaks!\n";

    }

}

bool ComChannel::open() {

    \_hSerial = ::CreateFile(

        LPCTSTR(\_port\_name.c\_str()),

        GENERIC\_READ | GENERIC\_WRITE,

        0,

        0,

        OPEN\_EXISTING,

        FILE\_ATTRIBUTE\_NORMAL,

        0

    );

    if (\_hSerial == INVALID\_HANDLE\_VALUE) {

        if (GetLastError() == ERROR\_FILE\_NOT\_FOUND) {

            std::cerr << "serial port does not exist.\n";

            return false;

        }

        std::cerr << "some other error occurred.\n";

        return false;

    }

    DCB dcbSerialParams = { 0 };

    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);

    if (!GetCommState(\_hSerial, &dcbSerialParams)) {

        std::cerr << "getting state error\n";

        return false;

    }

    dcbSerialParams.BaudRate = \_baud\_rate;

    dcbSerialParams.ByteSize = 8;

    dcbSerialParams.StopBits = ONESTOPBIT;

    dcbSerialParams.Parity = NOPARITY;

    if (!SetCommState(\_hSerial, &dcbSerialParams)) {

        std::cerr << "error setting serial port state\n";

        return false;

    }

    \_connected = true;

    return true;

}

bool ComChannel::close() {

    // проверяем статус подключение

    if (\_connected) {

        // отключаемся

        \_connected = false;

        // закрываем дескриптор порта

        return CloseHandle(\_hSerial);

    }

    return true;

}

bool ComChannel::write\_bytes(const std::vector<uint8\_t>& bytes) {

    DWORD dwSize = bytes.size();

    DWORD dwBytesWritten;

    return WriteFile(\_hSerial, bytes.data(), dwSize, &dwBytesWritten, NULL);

}

// return {START\_BYTE L\_LEN H\_LEN CMD\_CODE DATA\_0 ... DATA\_N L\_CRC H\_CRC}

std::vector<uint8\_t> ComChannel::read\_bytes\_until(const std::function<bool(uint8\_t)>& condition) {

    auto bytes = std::vector<uint8\_t>();

    DWORD iSize;

    uint8\_t byte;

    do {

        auto ok = ReadFile(\_hSerial, &byte, 1, &iSize, 0);  // читаем по одному байту

        if (ok) {

            bytes.push\_back(byte);

        }

        else {

            std::cerr << "error while reading\n";

            break;

        }

    } while (condition(byte));

    return bytes;

}

// Leonid Moguchev (c) 2020

#pragma once

#include <stdio.h>

#include <tchar.h>

#include <Windows.h>

#include <iostream>

#include <vector>

#include <string>

#include <functional>

class ComChannel {

public:

    ComChannel(const std::wstring& port\_name, uint32\_t baud\_rate);

    ~ComChannel();

    bool open();

    bool close();

    bool write\_bytes(const std::vector<byte>& bytes);

    std::vector<byte> read\_bytes\_until(const std::function<bool(byte)>& condition);

private:

    // дескриптор COM-порта

    HANDLE \_hSerial;

    bool \_connected;

    std::wstring \_port\_name;

    uint32\_t \_baud\_rate;

};

// Leonid Moguchev (c) 2020

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <stdbool.h>

#include <string.h>

#include <time.h>

static unsigned char reverse\_table[16] = {

  0x0, 0x8, 0x4, 0xC, 0x2, 0xA, 0x6, 0xE,

  0x1, 0x9, 0x5, 0xD, 0x3, 0xB, 0x7, 0xF

};

inline uint8\_t reverse\_bits(uint8\_t byte) {

    // Reverse the top and bottom nibble then swap them.

    return (reverse\_table[byte & 0b1111] << 4) | reverse\_table[byte >> 4];

}

inline uint16\_t reverse\_word(uint16\_t word) {

    return ((reverse\_bits(word & 0xFF) << 8) | reverse\_bits(word >> 8));

}

inline uint16\_t crc16\_common(uint8\_t\* data, uint8\_t len, uint16\_t poly, uint16\_t init,

    uint16\_t doXor, bool refIn, bool refOut) {

    uint8\_t y;

    uint16\_t crc;

    crc = init;

    while (len--) {

        if (refIn)

            crc = ((uint16\_t)reverse\_bits(\*data++) << 8) ^ crc;

        else

            crc = ((uint16\_t)\*data++ << 8) ^ crc;

        for (y = 0; y < 8; y++)  {

            if (crc & 0x8000)

                crc = (crc << 1) ^ poly;

            else

                crc = crc << 1;

        }

    }

    if (refOut)

        crc = reverse\_word(crc);

    return (crc ^ doXor);

}

inline uint16\_t crc16\_ccitt(uint8\_t\* data, uint8\_t len) {

    return crc16\_common(data, len, 0x1021, 0xFFFF, 0x0000, false, false);

}

// Leonid Moguchev (c) 2020

#include "FSParser.h"

#include "utils.h"

bool FSParser::\_reading\_command = false;

uint16\_t FSParser::\_msg\_len = 0;

size\_t FSParser::\_sum\_bytes = 0;

size\_t FSParser::\_read\_bytes = 0;

bool FSParser::process\_byte(uint8\_t b) {

    if (b == MSG\_START && !\_reading\_command) {

        //std::cout << "MSG\_START" << std::endl;

        \_reading\_command = true;

        \_msg\_len = 0x0000;

        \_sum\_bytes = 0;

        \_read\_bytes = 1;

        return true; // продолжить чтение

    }

    if (\_reading\_command) {

        switch (\_read\_bytes) {

        case 1: // младший байт LEN

            \_msg\_len = (uint16\_t)b & 0x00FF;

            break;

        case 2: // старший байт LEN

            \_msg\_len |= (((uint16\_t)b & 0x00FF) << 8);

            //std::cout << utils::string\_format("LEN=%i", \_msg\_len) << std::endl;

            break;

        case 3: // код ответа ANSWER

            \_msg\_len--;

            //std::cout << utils::string\_format("ANSWER=%i", b) << std::endl;

            break;

        default:

            //std::cout << utils::string\_format("PROCESS=%i", \_msg\_len) << std::endl;

            if (\_msg\_len > 0) { // DATA

                \_msg\_len--;

            }

            else { // контрольная сумма

                if (\_sum\_bytes == 0) { // младший байт CRC

                    //std::cout << "L CRC" << std::endl;

                    \_sum\_bytes++;

                }

                else if (\_sum\_bytes == 1) { // старший байт CRC

                    //std::cout << "H CRC" << std::endl;

                    \_sum\_bytes++;

                    \_reading\_command = false;

                    return false; // прекратить чтение

                }

                else {

                    std::cerr << "WTF AFTER CONTROL SUM?" << std::endl;

                    return false; // прекратить чтение

                }

            }

            break;

        }

        \_read\_bytes++;

        return true; // продолжить чтение

    }

    std::cerr << "UNKNOWN STATE" << std::endl;

    return false; // прекратить чтение

}

// Leonid Moguchev (c) 2020

#pragma once

#include "utils.h"

#include <string>

#include <iostream>

const uint8\_t MSG\_START = 0x04;

class FSParser {

public:

    FSParser() = default;

    ~FSParser() = default;

    bool static process\_byte(uint8\_t b);

private:

    bool static \_reading\_command;

    uint16\_t static \_msg\_len;

    size\_t static \_sum\_bytes;

    size\_t static \_read\_bytes;

};

// Leonid Moguchev (c) 2020

#include "Hardware.h"

Hardware::Hardware(const std::wstring& port\_name, uint32\_t baud) {

    \_com\_io = std::make\_unique<ComChannel>(port\_name, baud);

    \_connection\_success = \_com\_io->open();

    if (!\_connection\_success) {

        std::cerr << "open com port failed!" << std::endl;

    }

}

std::shared\_ptr<Message> Hardware::parse\_bytes(std::vector<uint8\_t>&& bytes) {

    auto msg = std::make\_shared<Message>();

    if (bytes.size() < 6) {

        msg->Code = INTERNAL\_ERROR;

        return msg;

    }

    msg->Length = utils::union\_bytes(bytes[1], bytes[2]);

    msg->Code = bytes[3];

    msg->Data = std::vector<uint8\_t>(bytes.begin() + 4, bytes.begin() + bytes.size() - 2);

    msg->Crc = utils::union\_bytes(bytes[bytes.size() - 2], bytes[bytes.size() - 1]);

    return msg;

}

bool Hardware::check\_crc(const Message& msg) {

    auto bytes = utils::split\_le(msg.Length);

    bytes.push\_back(msg.Code);

    bytes.insert(bytes.end(), msg.Data.begin(), msg.Data.end());

    auto real\_sum = crc16\_ccitt(bytes.data(), bytes.size());

    return real\_sum == msg.Crc;

}

bool Hardware::get\_connection\_status() {

    bool good\_connect = false;

    if (\_connection\_success) {

        auto res = this->\_\_30\_\_GetFNStatus();

        good\_connect = res->ErrorMsg == "";

    }

    return good\_connect;

};

std::shared\_ptr<StartFiscalisationResponse> Hardware::\_\_02\_\_StartFiscalisation() {

    auto result = std::make\_shared<StartFiscalisationResponse>();

    uint8\_t cmd = 0x02;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<ApproveFiscalisationResponse> Hardware::\_\_03\_\_ApproveFiscalisation(const ApproveFiscalisationRequest& req) {

    auto result = std::make\_shared<ApproveFiscalisationResponse>();

    uint8\_t cmd = 0x03;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code: " << response->Code << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 9) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->FiscDocNumber = utils::union\_bytes(

        response->Data[0], response->Data[1], response->Data[2], response->Data[3]);

    result->FiscSign = utils::union\_bytes(

        response->Data[4], response->Data[5], response->Data[6], response->Data[7]);

    return result;

}

std::shared\_ptr<StartCloseFiscalisationResponse> Hardware::\_\_04\_\_StartCloseFiscalisation() {

    auto result = std::make\_shared<StartCloseFiscalisationResponse>();

    uint8\_t cmd = 0x04;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<CloseFiscalisationResponse> Hardware::\_\_05\_\_CloseFiscalisation(const CloseFiscalisationRequest& req) {

    auto result = std::make\_shared<CloseFiscalisationResponse>();

    uint8\_t cmd = 0x05;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code: " << response->Code << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 9) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->FiscDocNumber = utils::union\_bytes(

        response->Data[0], response->Data[1], response->Data[2], response->Data[3]);

    result->FiscSign = utils::union\_bytes(

        response->Data[4], response->Data[5], response->Data[6], response->Data[7]);

    return result;

}

std::shared\_ptr<CancelDocumentResponse> Hardware::\_\_06\_\_CancelDocuments() {

    auto result = std::make\_shared<CancelDocumentResponse>();

    uint8\_t cmd = 0x06;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<SendDocumentsResponse> Hardware::\_\_07\_\_SendDocuments(const TLVList& list) {

    auto result = std::make\_shared<SendDocumentsResponse>();

    uint8\_t cmd = 0x07;

    auto chunks = list.to\_bytes\_with\_limit(TLV\_LIMIT);

    for (const auto& ch : chunks) {

        auto len = utils::split\_le(uint16\_t(ch.size() + 1));

        std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

        request.insert(request.end(), ch.begin(), ch.end());

        auto ptr = request.data();

        // crc считается без байта MSG\_START

        auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

        request.insert(request.end(), crc.begin(), crc.end());

        if (!\_com\_io->write\_bytes(request)) {

            std::cerr << "write bytes" << std::endl;

            result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

            return result;

        };

        auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

        auto response = parse\_bytes(std::move(bytes));

        if (!check\_crc(\*response)) {

            std::cerr << "bad control sum" << std::endl;

            result->ErrorMsg = CRC\_ERROR;

            return result;

        }

        if (response->Code != STATUS\_OK) {

            std::cerr << "error code" << std::endl;

            result->ErrorMsg = ERROR\_TEXT[response->Code];

            return result;

        }

    }

    return result;

}

std::shared\_ptr<GetShiftStatusResponse> Hardware::\_\_10\_\_GetShiftStatus() {

    auto result = std::make\_shared<GetShiftStatusResponse>();

    uint8\_t cmd = 0x10;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 6) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->ShiftOpen = response->Data[0] == 0x01;

    result->ShiftNum = utils::union\_bytes(response->Data[1], response->Data[2]);

    result->CheckAmmount = utils::union\_bytes(response->Data[3], response->Data[4]);

    return result;

}

std::shared\_ptr<StartOpeningShiftResponse> Hardware::\_\_11\_\_StartOpeningShift(const StartOpeningShiftRequest& req) {

    auto result = std::make\_shared<StartOpeningShiftResponse>();

    uint8\_t cmd = 0x11;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<ApproveOpeningShiftResponse> Hardware::\_\_12\_\_ApproveOpeningShift() {

    auto result = std::make\_shared<ApproveOpeningShiftResponse>();

    uint8\_t cmd = 0x12;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 11) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->ShiftNum = utils::union\_bytes(response->Data[0], response->Data[1]);

    result->FiscDocNumber = utils::union\_bytes(

        response->Data[2], response->Data[3], response->Data[4], response->Data[5]);

    result->FiscSign = utils::union\_bytes(

        response->Data[6], response->Data[7], response->Data[8], response->Data[9]);

    return result;

}

std::shared\_ptr<StartCloseShiftResponse> Hardware::\_\_13\_\_StartCloseShift(const StartCloseShiftRequest& req) {

    auto result = std::make\_shared<StartCloseShiftResponse>();

    uint8\_t cmd = 0x13;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<ApproveCloseShiftResponse> Hardware::\_\_14\_\_ApproveCloseShift() {

    auto result = std::make\_shared<ApproveCloseShiftResponse>();

    uint8\_t cmd = 0x14;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 11) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->ShiftNum = utils::union\_bytes(response->Data[0], response->Data[1]);

    result->FiscDocNumber = utils::union\_bytes(

        response->Data[2], response->Data[3], response->Data[4], response->Data[5]);

    result->FiscSign = utils::union\_bytes(

        response->Data[6], response->Data[7], response->Data[8], response->Data[9]);

    return result;

}

std::shared\_ptr<StartCheckResponse> Hardware::\_\_15\_\_StartCheck(const StartCheckRequest& req) {

    auto result = std::make\_shared<StartCheckResponse>();

    uint8\_t cmd = 0x15;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<CreateCheckResponse> Hardware::\_\_16\_\_CreateCheck(const CreateCheckRequest& req) {

    auto result = std::make\_shared<CreateCheckResponse>();

    uint8\_t cmd = 0x16;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 11) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->CheckNum = utils::union\_bytes(response->Data[0], response->Data[1]);

    result->FiscDocNumber = utils::union\_bytes(

        response->Data[2], response->Data[3], response->Data[4], response->Data[5]);

    result->FiscSign = utils::union\_bytes(

        response->Data[6], response->Data[7], response->Data[8], response->Data[9]);

    return result;

}

std::shared\_ptr<StartCheckCorrectionResponse> Hardware::\_\_17\_\_StartCheckCorrection(const StartCheckCorrectionRequest& req) {

    auto result = std::make\_shared<StartCheckCorrectionResponse>();

    uint8\_t cmd = 0x17;

    auto data = req.to\_bytes();

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    return result;

}

std::shared\_ptr<GetFNStatusResponse> Hardware::\_\_30\_\_GetFNStatus() {

    auto result = std::make\_shared<GetFNStatusResponse>();

    uint8\_t cmd = 0x30;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 31) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->PhaseOfLife = FN\_LIFE\_PHASES[response->Data[0]];

    result->CurrentDocument = CURRENT\_DOCUMENT\_TYPES[response->Data[1]];

    result->DocumentDataRecived = response->Data[2] == 1;

    result->ShiftIsOpen = response->Data[3] == 1;

    result->Warnings = WARNING\_FLAGS[response->Data[4]];

    result->DateTime = utils::date\_time\_from\_bytes(

        std::vector<uint8\_t>(response->Data.begin() + 5, response->Data.begin() + 10));

    result->Number\_cp866 = std::string(response->Data.begin() + 10, response->Data.begin() + 26);

    result->LastFDNumber = utils::union\_bytes(

        response->Data[26], response->Data[27], response->Data[28], response->Data[29]);

    return result;

}

std::shared\_ptr<GetFNNumberResponse> Hardware::\_\_31\_\_GetFNNumber() {

    auto result = std::make\_shared<GetFNNumberResponse>();

    uint8\_t cmd = 0x31;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 17) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->Number\_cp866 = std::string(response->Data.begin(), response->Data.end());

    return result;

}

std::shared\_ptr<GetFNEndDateResponse> Hardware::\_\_32\_\_GetFNEndDate() {

    auto result = std::make\_shared<GetFNEndDateResponse>();

    uint8\_t cmd = 0x32;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 6) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->Date = utils::date\_from\_bytes(std::vector<uint8\_t>(response->Data.begin(), response->Data.begin() + 3));

    result->LeftRegistrations = (response->Data[3]);

    result->DoneRegistrations = (response->Data[4]);

    return result;

}

std::shared\_ptr<GetFNVersionResponse> Hardware::\_\_33\_\_GetFNVersion() {

    auto result = std::make\_shared<GetFNVersionResponse>();

    uint8\_t cmd = 0x33;

    std::vector<uint8\_t> request = { MSG\_START, 0x01, 0x00, cmd };

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Length != 18) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->VersionSoftWare\_crc866 = std::string(response->Data.begin(), response->Data.begin() + 16);

    switch (response->Data[16]) {

    case 0:

        result->TypeSoftWare = "отладочная версия";

        break;

    case 1:

        result->TypeSoftWare = "серийная версия";

        break;

    }

    return result;

}

std::shared\_ptr<GetFiscDocumentResponse> Hardware::\_\_40\_\_GetFiscDocument(uint32\_t num) {

    auto result = std::make\_shared<GetFiscDocumentResponse>();

    uint8\_t cmd = 0x40;

    auto data = utils::split\_le(num);

    auto len = utils::split\_le(uint16\_t(data.size() + 1));

    std::vector<uint8\_t> request = { MSG\_START, len[0], len[1], cmd };

    request.insert(request.end(), data.begin(), data.end());

    auto ptr = request.data();

    // crc считается без байта MSG\_START

    auto crc = utils::split\_le(crc16\_ccitt(++ptr, request.size() - 1));

    request.insert(request.end(), crc.begin(), crc.end());

    if (!\_com\_io->write\_bytes(request)) {

        std::cerr << "write bytes" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    };

    auto bytes = \_com\_io->read\_bytes\_until(\_parser.process\_byte);

    auto response = parse\_bytes(std::move(bytes));

    if (!check\_crc(\*response)) {

        std::cerr << "bad control sum" << std::endl;

        result->ErrorMsg = CRC\_ERROR;

        return result;

    }

    if (response->Code != STATUS\_OK) {

        std::cerr << "error code" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[response->Code];

        return result;

    }

    if (response->Length - 1 != response->Data.size() || response->Data.size() < 3) {

        std::cerr << "wrong data size" << std::endl;

        result->ErrorMsg = ERROR\_TEXT[INTERNAL\_ERROR];

        return result;

    }

    result->DocumentType = DocumentType(response->Data[0]);

    result->GetOFDReceipt = response->Data[1] == 1;

    auto doc\_data = std::vector<uint8\_t>(response->Data.begin() + 2, response->Data.end());

    switch (result->DocumentType) {

    case DocumentType::CHECK:

        result->Document = new CheckDocument();

        break;

    case DocumentType::CLOSE\_FISCAL\_REGIME:

        result->Document = new CloseFiscDocumnet();

        break;

    case DocumentType::CLOSE\_SHIFT:

        result->Document = new ShiftDocument();

        break;

    case DocumentType::OPEN\_SHIFT:

        result->Document = new ShiftDocument();

        break;

    case DocumentType::REGISTRATION:

        result->Document = new RegistrationDocument();

        break;

    }

    result->Document->init(doc\_data);

    return result;

}

std::vector<uint8\_t> ApproveFiscalisationRequest::to\_bytes() const {

    std::vector<uint8\_t> res = utils::time\_to\_bytes(this->DateTime);

    auto inn = std::vector<uint8\_t>(this->Inn\_cp866.begin(), this->Inn\_cp866.end());

    while (inn.size() < 12) {

        inn.push\_back(0x00);

    }

    res.insert(res.end(), inn.begin(), inn.begin() + 12);

    auto kkt = std::vector<uint8\_t>(this->KKTNumber\_cp866.begin(), this->KKTNumber\_cp866.end());

    while (kkt.size() < 20) {

        kkt.push\_back(0x00);

    }

    res.insert(res.end(), kkt.begin(), kkt.begin() + 20);

    res.push\_back(uint8\_t(this->NalogCode));

    res.push\_back(uint8\_t(this->WorkMode));

    return res;

}

std::vector<uint8\_t> CloseFiscalisationRequest::to\_bytes() const {

    auto res = utils::time\_to\_bytes(this->DateTime);

    auto num = std::vector<uint8\_t>(this->KKTNumber\_cp866.begin(), this->KKTNumber\_cp866.end());

    while (num.size() < 20) {

        num.push\_back(0x00);

    }

    res.insert(res.end(), num.begin(), num.begin() + 20);

    return res;

}

std::vector<uint8\_t> StartOpeningShiftRequest::to\_bytes() const {

    return utils::time\_to\_bytes(this->DateTime);

}

std::vector<uint8\_t> StartCloseShiftRequest::to\_bytes() const {

    return utils::time\_to\_bytes(this->DateTime);

}

std::vector<uint8\_t> StartCheckRequest::to\_bytes() const {

    return utils::time\_to\_bytes(this->DateTime);

}

std::vector<uint8\_t> CreateCheckRequest::to\_bytes() const {

    auto res = utils::time\_to\_bytes(this->DateTime);

    res.push\_back(uint8\_t(this->OperationType));

    auto total = utils::split\_le(this->Total);

    res.insert(res.end(), total.begin(), total.begin() + 5);

    return res;

}

std::vector<uint8\_t> StartCheckCorrectionRequest::to\_bytes() const {

    return utils::time\_to\_bytes(this->DateTime);

}

TLVList CommonData::to\_tlv\_list() const {

    TLVList list;

    if (this->UserName.size() != 0) {

        auto user\_name = std::vector<uint8\_t>(this->UserName.begin(), this->UserName.end());

        list.push(std::move(TLV(0x0418, user\_name)));

    }

    if (this->Cashier.size() != 0) {

        auto cashier = std::vector<uint8\_t>(this->Cashier.begin(), this->Cashier.end());

        list.push(std::move(TLV(0x03FD, cashier)));

    }

    if (this->Address.size() != 0) {

        auto address = std::vector<uint8\_t>(this->Address.begin(), this->Address.end());

        list.push(std::move(TLV(0x03F1, address)));

    }

    if (this->InnOFD.size() != 0) {

        auto inn\_ofd = std::vector<uint8\_t>(this->InnOFD.begin(), this->InnOFD.end());

        list.push(std::move(TLV(0x03F9, inn\_ofd)));

    }

    return list;

}

void CheckDocument::init(const std::vector<uint8\_t>& data) {

    if (data.size() != 19) {

        return;

    }

    DateTime = utils::date\_time\_from\_bytes(std::vector<uint8\_t>(data.begin(), data.begin() + 5));

    Number = utils::union\_bytes(data[5], data[6], data[7], data[8]);

    FiscSign = utils::union\_bytes(data[9], data[10], data[11], data[12]);

    OpType = OperationType(data[13]);

    Sum = utils::union\_bytes(data[14], data[15], data[16], data[17], data[18], 0, 0, 0);

}

void  CloseFiscDocumnet::init(const std::vector<uint8\_t>& data) {

    if (data.size() != 45) {

        return;

    }

    DateTime = utils::date\_time\_from\_bytes(std::vector<uint8\_t>(data.begin(), data.begin() + 5));

    Number = utils::union\_bytes(data[5], data[6], data[7], data[8]);

    FiscSign = utils::union\_bytes(data[9], data[10], data[11], data[12]);

    Inn = std::string(data.begin() + 13, data.begin() + 25);

    KTTNumber = std::string(data.begin() + 25, data.begin() + 45);

}

void ShiftDocument::init(const std::vector<uint8\_t>& data) {

    if (data.size() != 15) {

        return;

    }

    DateTime = utils::date\_time\_from\_bytes(std::vector<uint8\_t>(data.begin(), data.begin() + 5));

    Number = utils::union\_bytes(data[5], data[6], data[7], data[8]);

    FiscSign = utils::union\_bytes(data[9], data[10], data[11], data[12]);

    ShiftNum = utils::union\_bytes(data[13], data[14]);

}

void RegistrationDocument::init(const std::vector<uint8\_t>& data) {

    if (data.size() != 47) {

        return;

    }

    DateTime = utils::date\_time\_from\_bytes(std::vector<uint8\_t>(data.begin(), data.begin() + 5));

    Number = utils::union\_bytes(data[5], data[6], data[7], data[8]);

    FiscSign = utils::union\_bytes(data[9], data[10], data[11], data[12]);

    Inn = std::string(data.begin() + 13, data.begin() + 25);

    KTTNumber = std::string(data.begin() + 25, data.begin() + 45);

    NCode = NalogCode(data[45]);

    WMode = WorkMode(data[46]);

}

// Leonid Moguchev (c) 2020

#pragma once

#include "ComChannel.h"

#include "FSParser.h"

#include "utils.h"

#include "CRC16.h"

#include "TLV.h"

#include <ctime>

#include <string>

#include <iostream>

#include <memory>

#include <map>

enum class NalogCode {

    COMMON = 1, // Общая

    SIMPLE\_INCOME = 2, // Упрощенная Доход

    SIMPLE\_INCOME\_MINUS\_EXPENSE = 4, // Упрощенная Доход минус Расход

    SINGLE\_TAX\_ON\_IMPUTED\_INCOME = 8, // Единый налог на вмененный доход

    UNIFIED\_AGRICULTURAL\_TAX = 16, // Единый сельскохозяйственный налог

    PATENT\_TAXATION\_SYSTEM = 32, // Патентная система налогообложения

};

enum class WorkMode {

    ENCRYPTION = 1, // Шифрование

    OFFLINE = 2, // Автономный режим

    AUTO = 4, // Автоматический режим

    SERVICE\_APPLICATIONS = 8, // Применение в сфере услуг

    CHECK = 16, // Режим БСО (1) иначе Режим чеков (0)

    INTERNET\_COMMERCE = 32, // Применение в Интернет-торговле

};

enum class OperationType {

    PARISH = 1, // Приход

    RETURN\_PARISH = 2, // Возврат прихода

    CONSUMPTION = 3, // Расход

    RETURN\_CONSUMPTION = 4, // Возврат расхода

};

enum class DocumentType {

    REGISTRATION = 1,

    OPEN\_SHIFT = 2,

    CHECK = 3,

    CLOSE\_SHIFT = 5,

    CLOSE\_FISCAL\_REGIME = 6,

};

static std::map<uint8\_t, std::string> ERROR\_TEXT = {

    {0x01, "Неизвестная команда, неверный формат посылки или неизвестные параметры"},

    {0x02, "Неверное состояние ФН"},

    {0x03, "Ошибка ФН"},

    {0x04, "Ошибка КС "},

    {0x05, "Закончен срок эксплуатации ФН"},

    {0x06, "Архив ФН переполнен"},

    {0x07, "Неверные дата и/или время"},

    {0x08, "Нет запрошенных данных"},

    {0x09, "Некорректное значение параметров команды"},

    {0x10, "Превышение размеров TLV данных"},

    {0x11, "Нет транспортного соединения"},

    {0x12, "Исчерпан ресурс КС (криптографического сопроцессора)"},

    {0x14, "Исчерпан ресурс хранения "},

    {0x20, "Сообщение от ОФД не может быть принято"},

};

static std::map<uint8\_t, std::string> FN\_LIFE\_PHASES = {

    {0, "Настройка"},

    {1, "Готовность к фискализации"},

    {3, "Фискальный режим"},

    {7, "Фискальный режим закрыт, идет передача ФД в ОФД"},

    {15, "Чтение данных из Архива ФН"},

};

static std::map<uint8\_t, std::string> CURRENT\_DOCUMENT\_TYPES = {

    {0x00, "нет открытого документа"},

    {0x01, "отчёт о фискализации"},

    {0x02, "отчёт об открытии смены"},

    {0x04, "кассовый чек"},

    {0x08, "отчёт о закрытии смены"},

    {0x10, "отчёт о закрытии фискального режима"},

};

static std::map<uint8\_t, std::string> WARNING\_FLAGS = {

    {0, ""},

    {1, "Срочная замена КС (до окончания срока действия 3 дня)"},

    {2, "Исчерпание ресурса КС (до окончания срока действия 30 дней)"},

    {4, "Переполнение памяти ФН (Архив ФН заполнен на 90 %)"},

    {8, "Превышено время ожидания ответа ОФД"},

};

struct Message {

    uint16\_t Length;

    uint8\_t Code;

    std::vector<uint8\_t> Data;

    uint16\_t Crc;

};

struct IRequest {

    time\_t DateTime;

    virtual std::vector<uint8\_t> to\_bytes() const = 0;

    virtual ~IRequest() = default;

};

struct Response {

    std::string ErrorMsg;

};

struct GetFNStatusResponse : public Response {

    std::string PhaseOfLife;  // Фаза жизни

    std::string CurrentDocument;

    bool DocumentDataRecived;

    bool ShiftIsOpen;         // Состояние смены

    std::string Warnings;     // Флаги предупреждения

    std::string DateTime;

    std::string Number\_cp866; // Номер ФН

    uint32\_t LastFDNumber;    // Номер последнего ФД

};

struct GetFNNumberResponse : public Response {

    std::string Number\_cp866;

};

struct GetFNEndDateResponse : public Response {

    std::string Date;

    uint8\_t LeftRegistrations;

    uint8\_t DoneRegistrations;

};

struct GetFNVersionResponse : public Response {

    std::string VersionSoftWare\_crc866;

    std::string TypeSoftWare;

};

struct StartFiscalisationResponse : public Response {};

struct SendDocumentsResponse : public Response {};

struct CancelDocumentResponse : public Response {};

struct StartCloseFiscalisationResponse : public Response {};

struct FiscData {

    uint32\_t FiscDocNumber;

    uint32\_t FiscSign;

};

struct ApproveFiscalisationResponse : public Response, public FiscData {};

struct CloseFiscalisationResponse : public Response, public FiscData {};

struct StartOpeningShiftResponse : public Response {};

struct ApproveOpeningShiftResponse : public Response, public FiscData {

    uint16\_t ShiftNum;

};

struct GetShiftStatusResponse : public Response {

    bool ShiftOpen;

    uint16\_t ShiftNum;

    uint16\_t CheckAmmount;

};

struct StartCheckResponse : public Response {};

struct StartCheckCorrectionResponse : public Response {};

struct StartCloseShiftResponse : public Response {};

struct ApproveCloseShiftResponse : public Response, public FiscData {

    uint16\_t ShiftNum;

};

struct CreateCheckResponse : public Response, public FiscData {

    uint16\_t CheckNum;

};

struct IFiscalDocument {

    std::string DateTime;

    uint32\_t Number;

    uint32\_t FiscSign;

    virtual void init(const std::vector<uint8\_t>& data) = 0;

    virtual ~IFiscalDocument() = default;

};

struct GetFiscDocumentResponse : public  Response {

    DocumentType DocumentType;

    bool GetOFDReceipt;

    IFiscalDocument\* Document;

};

struct RegistrationDocument : public IFiscalDocument {

    std::string Inn;

    std::string KTTNumber;

    NalogCode NCode;

    WorkMode WMode;

    virtual void init(const std::vector<uint8\_t>& data);

    virtual ~RegistrationDocument() = default;

};

struct ShiftDocument : public IFiscalDocument {

    uint16\_t ShiftNum;

    virtual void init(const std::vector<uint8\_t>& data);

    virtual ~ShiftDocument() = default;

};

struct CheckDocument : public IFiscalDocument {

    OperationType OpType;

    uint64\_t Sum;

    virtual void init(const std::vector<uint8\_t>& data);

    virtual ~CheckDocument() = default;

};

struct CloseFiscDocumnet : public IFiscalDocument {

    std::string Inn;

    std::string KTTNumber;

    virtual void init(const std::vector<uint8\_t>& data);

    virtual ~CloseFiscDocumnet() = default;

};

struct ApproveFiscalisationRequest : public IRequest {

    std::string Inn\_cp866;

    std::string KKTNumber\_cp866;

    NalogCode NalogCode;

    WorkMode WorkMode;

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct CloseFiscalisationRequest : public IRequest {

    std::string KKTNumber\_cp866;

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct StartOpeningShiftRequest : public IRequest {

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct StartCloseShiftRequest : public IRequest {

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct StartCheckRequest : public IRequest {

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct StartCheckCorrectionRequest : public IRequest {

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct CreateCheckRequest : public IRequest {

    OperationType OperationType;

    uint64\_t Total;

    virtual std::vector<uint8\_t> to\_bytes() const;

};

struct CommonData {

    std::string UserName;

    std::string Cashier;

    std::string Address;

    std::string InnOFD;

    TLVList to\_tlv\_list() const;

};

class Hardware {

public:

    Hardware(const std::wstring& port\_name, uint32\_t baud);

    ~Hardware() = default;

    bool get\_connection\_status();

    std::shared\_ptr<StartFiscalisationResponse> \_\_02\_\_StartFiscalisation();

    std::shared\_ptr<ApproveFiscalisationResponse> \_\_03\_\_ApproveFiscalisation(const ApproveFiscalisationRequest& req);

    std::shared\_ptr<StartCloseFiscalisationResponse> \_\_04\_\_StartCloseFiscalisation();

    std::shared\_ptr<CloseFiscalisationResponse> \_\_05\_\_CloseFiscalisation(const CloseFiscalisationRequest& req);

    std::shared\_ptr<CancelDocumentResponse> \_\_06\_\_CancelDocuments();

    std::shared\_ptr<SendDocumentsResponse> \_\_07\_\_SendDocuments(const TLVList& list);

    std::shared\_ptr<GetShiftStatusResponse> \_\_10\_\_GetShiftStatus();

    std::shared\_ptr<StartOpeningShiftResponse> \_\_11\_\_StartOpeningShift(const StartOpeningShiftRequest& req);

    std::shared\_ptr<ApproveOpeningShiftResponse> \_\_12\_\_ApproveOpeningShift();

    std::shared\_ptr<StartCloseShiftResponse> \_\_13\_\_StartCloseShift(const StartCloseShiftRequest& req);

    std::shared\_ptr<ApproveCloseShiftResponse> \_\_14\_\_ApproveCloseShift();

    std::shared\_ptr<StartCheckResponse> \_\_15\_\_StartCheck(const StartCheckRequest& req);

    std::shared\_ptr<CreateCheckResponse> \_\_16\_\_CreateCheck(const CreateCheckRequest& req);

    std::shared\_ptr<StartCheckCorrectionResponse> \_\_17\_\_StartCheckCorrection(const StartCheckCorrectionRequest& req);

    std::shared\_ptr<GetFNStatusResponse> \_\_30\_\_GetFNStatus();

    std::shared\_ptr<GetFNNumberResponse> \_\_31\_\_GetFNNumber();

    std::shared\_ptr<GetFNEndDateResponse> \_\_32\_\_GetFNEndDate();

    std::shared\_ptr<GetFNVersionResponse> \_\_33\_\_GetFNVersion();

    std::shared\_ptr<GetFiscDocumentResponse> \_\_40\_\_GetFiscDocument(uint32\_t num);

private:

    const uint8\_t MSG\_START = 0x04;

    const uint8\_t STATUS\_OK = 0x00;

    const uint8\_t INTERNAL\_ERROR = 0x03;

    const std::string CRC\_ERROR = "Ошибка контрольной суммы";

    const size\_t TLV\_LIMIT = 1024;

    std::unique\_ptr<ComChannel> \_com\_io;

    bool \_connection\_success;

    FSParser \_parser;

    std::shared\_ptr<Message> parse\_bytes(std::vector<uint8\_t>&& bytes);

    bool check\_crc(const Message& msg);

};

// Leonid Moguchev (c) 2020

#include <iostream>

#include <memory>

#include "utils.h"

#include "Hardware.h"

#include "TLV.h"

int main() {

    //SetConsoleCP(866);// установка кодовой страницы win-cp 866 в поток ввода

    //SetConsoleOutputCP(866); // установка кодовой страницы win-cp 866 в поток вывода

    setlocale(LC\_ALL, "ru\_RU");

    auto service = std::make\_shared<Hardware>(L"COM2", 100000);

    auto connected = service->get\_connection\_status();

    std::cout << "CONNECTION: " << std::boolalpha << connected << std::endl;

    if (!connected) {

        return 1;

    }

/////////////////////////////////////////////////////////////////////////////

// 30

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_30\_\_GetFNStatus();

        std::cout << "COMMAND: 30h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->PhaseOfLife << std::endl;

        std::cout << response->CurrentDocument << std::endl;

        std::cout << std::boolalpha << response->DocumentDataRecived << std::endl;

        std::cout << std::boolalpha << response->ShiftIsOpen << std::endl;

        std::cout << response->Warnings << std::endl;

        std::cout << response->Number\_cp866 << std::endl;

        std::cout << response->DateTime << std::endl;

        std::cout << response->LastFDNumber << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 31

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_31\_\_GetFNNumber();

        std::cout << "COMMAND: 31h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->Number\_cp866 << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 32

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_32\_\_GetFNEndDate();

        std::cout << "COMMAND: 32h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->Date << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 33

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_33\_\_GetFNVersion();

        std::cout << "COMMAND: 33h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->VersionSoftWare\_crc866 << std::endl;

        std::cout << response->TypeSoftWare << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 02 - Начать фискализацию (регистрацию ККТ)

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_02\_\_StartFiscalisation();

        std::cout << "COMMAND: 02h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 07 - Передать необходимые документы

/////////////////////////////////////////////////////////////////////////////

    {

        auto doc = CommonData{};

        doc.UserName = "Иванов И.И.";

        doc.Сashier = "Петров П.П.";

        doc.Address = "ул.Тверская, д.1";

        doc.InnOFD = "790123456789";

        auto response = service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        std::cout << "COMMAND: 07h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 03 - Фискализация (регистрация ККТ)

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = ApproveFiscalisationRequest{};

        req.DateTime = time(0);

        req.Inn\_cp866 = "112233445566";

        req.KKTNumber\_cp866 = "KKT-772-233-445-566";

        req.NalogCode = NalogCode::COMMON;

        req.WorkMode = WorkMode::AUTO;

        auto response = service->\_\_03\_\_ApproveFiscalisation(req);

        std::cout << "COMMAND: 03h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->FiscDocNumber << std::endl;

        std::cout << response->FiscSign << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 11 - Начать открытие смены

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = StartOpeningShiftRequest{};

        req.DateTime = time(0);

        auto response = service->\_\_11\_\_StartOpeningShift(req);

        std::cout << "COMMAND: 11h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 07 - Передать необходимые документы

/////////////////////////////////////////////////////////////////////////////

    {

        auto doc = CommonData{};

        doc.UserName = "Иванов И.И.";

        doc.Сashier = "Петров П.П.";

        doc.Address = "ул.Тверская, д.1";

        auto response = service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        std::cout << "COMMAND: 07h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 12 - Открыть смену

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_12\_\_ApproveOpeningShift();

        std::cout << "COMMAND: 12h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->ShiftNum << std::endl;

        std::cout << response->FiscDocNumber << std::endl;

        std::cout << response->FiscSign << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 10 - Запрос параметров текущей смены

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_10\_\_GetShiftStatus();

        std::cout << "COMMAND: 10h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->ShiftNum << std::endl;

        std::cout << std::boolalpha << response->ShiftOpen << std::endl;

        std::cout << response->CheckAmmount << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 15 - Начать формирование чека

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = StartCheckRequest{};

        req.DateTime = time(0);

        auto response = service->\_\_15\_\_StartCheck(req);

        std::cout << "COMMAND: 15h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 07 - Передать необходимые документы

/////////////////////////////////////////////////////////////////////////////

    {

        // ВООБЩЕ ПОХЕР ЧТО ОТПРАВЛЯЕТСЯ, ГЛАВНОЕ ХОТЯБЫ РАЗ ЭТО СДЕЛАТЬ

        auto doc = CommonData{};

        doc.UserName = "Иванов И.И.";

        doc.Сashier = "Петров П.П.";

        doc.Address = "ул.Тверская, д.1";

        auto response = service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        std::cout << "COMMAND: 07h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 16 - Сформировать чек

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = CreateCheckRequest();

        req.DateTime = time(0);

        req.OperationType = OperationType::PARISH; // Приход

        req.Total = 1000 \* 100; // в копейках

        auto response = service->\_\_16\_\_CreateCheck(req);

        std::cout << "COMMAND: 16h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->CheckNum << std::endl;

        std::cout << std::boolalpha << response->FiscDocNumber << std::endl;

        std::cout << response->FiscSign << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 17 - Начать формирование чека коррекции

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = StartCheckCorrectionRequest{};

        req.DateTime = time(0);

        auto response = service->\_\_17\_\_StartCheckCorrection(req);

        std::cout << "COMMAND: 17h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 07 - Передать необходимые документы

/////////////////////////////////////////////////////////////////////////////

    {

        // ВООБЩЕ ПОХЕР ЧТО ОТПРАВЛЯЕТСЯ, ГЛАВНОЕ ХОТЯБЫ РАЗ ЭТО СДЕЛАТЬ

        auto doc = CommonData{};

        doc.UserName = "Иванов И.И.";

        doc.Сashier = "Петров П.П.";

        doc.Address = "ул.Тверская, д.1";

        auto response = service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        std::cout << "COMMAND: 07h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 16 - Сформировать чек (коррекции)

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = CreateCheckRequest();

        req.DateTime = time(0);

        req.OperationType = OperationType::PARISH; // Приход

        req.Total = 1000 \* 100; // в копейках

        auto response = service->\_\_16\_\_CreateCheck(req);

        std::cout << "COMMAND: 16h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->CheckNum << std::endl;

        std::cout << std::boolalpha << response->FiscDocNumber << std::endl;

        std::cout << response->FiscSign << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 13 - Начать закрытие смены

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = StartCloseShiftRequest{};

        req.DateTime = time(0);

        auto response = service->\_\_13\_\_StartCloseShift(req);

        std::cout << "COMMAND: 13h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 07 - Передать необходимые документы

/////////////////////////////////////////////////////////////////////////////

    {

        auto doc = CommonData{};

        doc.UserName = "Иванов И.И.";

        doc.Сashier = "Петров П.П.";

        doc.Address = "ул.Тверская, д.1";

        auto response = service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        std::cout << "COMMAND: 07h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 14 - Закрыть смену

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_14\_\_ApproveCloseShift();

        std::cout << "COMMAND: 14h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->ShiftNum << std::endl;

        std::cout << std::boolalpha << response->FiscDocNumber << std::endl;

        std::cout << response->FiscSign << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 04 - Начать закрытие фискального режима

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_04\_\_StartCloseFiscalisation();

        std::cout << "COMMAND: 04h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 07 - Передать необходимые документы

/////////////////////////////////////////////////////////////////////////////

    {

        auto doc = CommonData{};

        doc.Сashier = "Петров П.П.";

        doc.Address = "ул.Тверская, д.1";

        auto response = service->\_\_07\_\_SendDocuments(doc.to\_tlv\_list());

        std::cout << "COMMAND: 07h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

// 05 - Закрыть фискальный режим ФН

/////////////////////////////////////////////////////////////////////////////

    {

        auto req = CloseFiscalisationRequest{};

        req.DateTime = time(0);

        req.KKTNumber\_cp866 = "KKT-772-233-445-566";

        auto response = service->\_\_05\_\_CloseFiscalisation(req);

        std::cout << "COMMAND: 05h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << response->FiscDocNumber << std::endl;

        std::cout << response->FiscSign << std::endl;

    }

/////////////////////////////////////////////////////////////////////////////

// 40 - Найти фискальный документ по номеру

/////////////////////////////////////////////////////////////////////////////

    {

        auto response = service->\_\_40\_\_GetFiscDocument(3);

        std::cout << "COMMAND: 40h" << std::endl;

        if (response->ErrorMsg != "") {

            std::cout << "ERROR: " << response->ErrorMsg << std::endl;

        }

        std::cout << int(response->DocumentType) << std::endl;

        std::cout << std::boolalpha << response->GetOFDReceipt << std::endl;

        // у всех ФД

        std::cout << response->Document->DateTime << std::endl;

        std::cout << response->Document->Number << std::endl;

        std::cout << response->Document->FiscSign << std::endl;

        // в зависимости от типа документа:

        if (CheckDocument\* doc = dynamic\_cast<CheckDocument\*>(response->Document)) {

            // чек

            std::cout << "CHECK DOCUMENT" << std::endl;

            std::cout << int(doc->OpType) << std::endl;

            std::cout << doc->Sum << std::endl;

        }

        if (RegistrationDocument\* doc = dynamic\_cast<RegistrationDocument\*>(response->Document)) {

            // фискализация

            std::cout << "REGISTRATION DOCUMENT" << std::endl;

            std::cout << doc->Inn << std::endl;

            std::cout << doc->KTTNumber << std::endl;

            std::cout << int(doc->NCode) << std::endl;

            std::cout << int(doc->WMode) << std::endl;

        }

        if (ShiftDocument\* doc = dynamic\_cast<ShiftDocument\*>(response->Document)) {

            // открытие/закрытие смены

            std::cout << "SHIFT DOCUMENT" << std::endl;

            std::cout << doc->ShiftNum << std::endl;

        }

        if (CloseFiscDocumnet\* doc = dynamic\_cast<CloseFiscDocumnet\*>(response->Document)) {

            // закрытие фиск. режима

            std::cout << "CLOSE FISCAL MODE DOCUMENT" << std::endl;

            std::cout << doc->Inn << std::endl;

            std::cout << doc->KTTNumber << std::endl;

        }

    }

/////////////////////////////////////////////////////////////////////////////

    system("pause");

}

// Leonid Moguchev (c) 2020

#include "TLV.h"

std::vector<uint8\_t> TLV::to\_bytes() const {

    auto result = utils::split\_le(\_tag);

    auto len = utils::split\_le(\_len);

    result.insert(result.end(), len.begin(), len.end());

    result.insert(result.end(), \_data.begin(), \_data.end());

    return result;

}

TLV::TLV(uint16\_t tag, const std::vector<uint8\_t>& data) noexcept :

    \_tag(tag), \_len(data.size()), \_data(data) {}

TLV::TLV(uint16\_t tag, std::vector<uint8\_t>&& data) noexcept :

    \_tag(tag), \_len(data.size()), \_data(std::move(data)) {}

TLV::TLV(TLV&& t) noexcept :

    \_tag(t.\_tag), \_len(t.\_len), \_data(std::move(t.\_data)) {}

TLV::TLV(const TLV& t) noexcept :

    \_tag(t.\_tag), \_len(t.\_len), \_data(t.\_data) {}

TLV& TLV::operator = (const TLV& t) {

    \_tag = t.\_tag;

    \_len = t.\_len;

    \_data = t.\_data;

    return \*this;

}

TLV& TLV::operator = (TLV&& t) noexcept {

    \_tag = t.\_tag;

    \_len = t.\_len;

    \_data = std::move(t.\_data);

    return \*this;

}

uint16\_t TLV::get\_tag() const {

    return \_tag;

}

std::vector<std::vector<uint8\_t>> TLVList::to\_bytes\_with\_limit(size\_t limit) const {

    std::vector<std::vector<uint8\_t>> result;

    std::vector<uint8\_t> chunk;

    for (const auto& obj : \_list) {

        auto bytes = obj.second.to\_bytes();

        if (chunk.size() + bytes.size() > limit) {

            result.push\_back(chunk);

            chunk.clear();

        }

        chunk.insert(chunk.end(), bytes.begin(), bytes.end());

    }

    if (chunk.size() > 0) {

        result.push\_back(chunk);

        chunk.clear();

    }

    return result;

}

void TLVList::push(const TLV& obj) {

    \_list.emplace(std::make\_pair<>(obj.get\_tag(), obj));

}

void TLVList::push(TLV&& obj) {

    \_list.emplace(std::make\_pair<>(obj.get\_tag(), std::move(obj)));

}

// Leonid Moguchev (c) 2020

#pragma once

#include <stdint.h>

#include <vector>

#include <map>

#include "utils.h"

class TLV {

public:

    TLV(uint16\_t tag, const std::vector<uint8\_t>& data) noexcept;

    TLV(uint16\_t tag, std::vector<uint8\_t>&& data) noexcept;

    TLV(TLV&& t) noexcept;

    TLV(const TLV& t) noexcept;

    TLV& operator = (const TLV& t);

    TLV& operator = (TLV&& t) noexcept;

    ~TLV() = default;

    uint16\_t get\_tag() const;

    std::vector<uint8\_t> to\_bytes() const;

private:

    uint16\_t \_tag;

    uint16\_t \_len;

    std::vector<uint8\_t> \_data;

};

class TLVList {

public:

    TLVList() = default;

    ~TLVList() = default;

    void push(const TLV& obj);

    void push(TLV&& obj);

    std::vector<std::vector<uint8\_t>> to\_bytes\_with\_limit(size\_t limit) const;

private:

    std::map<uint16\_t, TLV> \_list;

};

// Leonid Moguchev (c) 2020

#pragma once

#include <string>

#include <vector>

#include <memory>

#include <stdexcept>

#include <time.h>

namespace utils {

    template<typename ...Args>

    std::string string\_format(const std::string& format, Args ...args) {

        size\_t size = uint64\_t(snprintf(nullptr, 0, format.c\_str(), args ...)) + 1; // Extra space for '\0'

        if (size <= 0) {

            throw std::runtime\_error("Error during formatting.");

        }

        std::unique\_ptr<char[]> buf(new char[size]);

        snprintf(buf.get(), size, format.c\_str(), args...);

        return std::string(buf.get(), buf.get() + size - 1); // We don't want the '\0' inside

    }

    template<typename T>

    std::vector<uint8\_t> split\_le(T value) {

        auto res = std::vector<uint8\_t>(sizeof(T));

        uint8\_t\* ptr = (uint8\_t\*)&value;

        for (size\_t i = 0; i < sizeof(T); ++i) {

            res[i] = \*ptr++;

        }

        return res;

    }

    const auto HEX = "0123456789ABCDEF";

    inline std::string byte\_to\_hex(uint8\_t c) {

        return string\_format("0x%c%c", HEX[(c >> 4) & 0xF], HEX[c & 0xF]);

    }

    inline uint16\_t union\_bytes(uint8\_t l, uint8\_t h) {

        return uint16\_t(l) | (uint16\_t(h) << 8);

    }

    inline uint32\_t union\_bytes(uint8\_t b0, uint8\_t b1, uint8\_t b2, uint8\_t b3) {

        return (uint32\_t(b0)) | (uint32\_t(b1) << 8) | (uint32\_t(b2) << 16) | (uint32\_t(b3) << 24);

    }

    inline uint64\_t union\_bytes(uint8\_t b0, uint8\_t b1, uint8\_t b2, uint8\_t b3, uint8\_t b4, uint8\_t b5, uint8\_t b6, uint8\_t b7) {

        return (uint64\_t(b0)) | (uint64\_t(b1) << 8) | (uint64\_t(b2) << 16) | (uint64\_t(b3) << 24) | (uint64\_t(b4) << 32) | (uint64\_t(b5) << 40) | (uint64\_t(b6) << 48) | (uint64\_t(b7) << 56);

    }

    inline std::string date\_from\_bytes(const std::vector<uint8\_t>& bytes) {

        if (bytes.size() != 3) {

            return "";

        }

        auto day = string\_format("%d", bytes[2]);

        if (bytes[2] < 10) {

            day = "0" + day;

        }

        auto month = string\_format("%d", bytes[1]);

        if (bytes[1] < 10) {

            month = "0" + month;

        }

        auto year = string\_format("%d", bytes[0]);

        if (bytes[1] < 10) {

            year = "0" + year;

        }

        return string\_format("%s.%s.20%s", day.c\_str(), month.c\_str(), year.c\_str());

    }

    inline std::string date\_time\_from\_bytes(const std::vector<uint8\_t>& bytes) {

        if (bytes.size() != 5) {

            return "";

        }

        auto min = string\_format("%d", bytes[4]);

        if (bytes[4] < 10) {

            min = "0" + min;

        }

        auto hour = string\_format("%d", bytes[3]);

        if (bytes[3] < 10) {

            hour = "0" + hour;

        }

        auto day = string\_format("%d", bytes[2]);

        if (bytes[2] < 10) {

            day = "0" + day;

        }

        auto month = string\_format("%d", bytes[1]);

        if (bytes[1] < 10) {

            month = "0" + month;

        }

        auto year = string\_format("%d", bytes[0]);

        if (bytes[0] < 10) {

            year = "0" + year;

        }

        return string\_format("%s.%s.20%s %s:%s", day.c\_str(), month.c\_str(), year.c\_str(), hour.c\_str(), min.c\_str());

    }

    inline std::vector<uint8\_t> time\_to\_bytes(time\_t t) {

        std::vector<uint8\_t> data\_time;

        struct tm ltm;

        localtime\_s(&ltm, &t);

        data\_time.push\_back(ltm.tm\_year - 30);

        data\_time.push\_back(ltm.tm\_mon + 1);

        data\_time.push\_back(ltm.tm\_mday);

        data\_time.push\_back(ltm.tm\_hour + 1);

        data\_time.push\_back(ltm.tm\_min + 1);

        return data\_time;

    }

}



Перечень принятых сокращений

**ККТ** — Контрольно-кассовая техника

**ККМ** — Контрольно-кассовая машина

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц) в докум. | № докумен-та | Входящий № сопроводит. докум. и дата | Подп. | Дата |
| Изм. | изменен-ных | заменен-ных | новых | аннули-рованных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

1. ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)