УТВЕРЖДЕН
МГТУ.111111.001-01 12 01-ЛУ

СПЕЦИАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ «ККМ»

Текст программы

МГТУ.111111.001-01 12 01

Листов 91

Подпись и дата	
Инв. № дубл.	
Взам. инв. №	
Подпись и дата	
Инв. № подл.	

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], ГОСТ 19.103-77 [2], ГОСТ 19.104-78* [3], ГОСТ 19.105-78* [4], ГОСТ 19.106-78* [5], ГОСТ 19.401-78 [6], ГОСТ 19.604-78* [7]).

[1] ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов
[2] ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов
[3] ГОСТ 19.104-78* ЕСПД. Основные надписи
[4] ГОСТ 19.105-78* ЕСПД. Общие требования к программным документам
[5] ГОСТ 19.106-78* ЕСПД. Общие требования к программным документам, выполненным печатным способом
[6] ГОСТ 19.401-78 ЕСПД. Текст программы. Требования к содержанию и оформлению
[7] ГОСТ 19.604-78* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом

СОДЕРЖАНИЕ

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(co
nst 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(re
sponse->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(res
ponse->VersionSoftWare_crc866.c_str()).arg("\n");
        message += QString("%1%2%3").arg("Тип программного обеспечения ФН: ").arg(respon
se->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, QSqlTa
bleModel *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 ре
жим", "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("ООО \"Курсовая\"");
    ui->addr->setText("г. Москва");
    ui->place->setText("Там где сделанный курсач");
    ui->regNumber->setText("ККТ-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)>& co
ndition) {
    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 Clo
seFiscalisationRequest& 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 Start
OpeningShiftRequest& 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 StartClos
eShiftRequest& 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 CreateCheckReques
t& 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>
```

```cpp
#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 Appro
veFiscalisationRequest& req);
    std::shared_ptr<StartCloseFiscalisationResponse> __04__StartCloseFiscalisation();
    std::shared_ptr<CloseFiscalisationResponse> __05__CloseFiscalisation(const CloseFisc
alisationRequest& 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 StartOpenin
gShiftRequest& req);
    std::shared_ptr<ApproveOpeningShiftResponse> __12__ApproveOpeningShift();
    std::shared_ptr<StartCloseShiftResponse> __13__StartCloseShift(const StartCloseShift
Request& 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 Start
CheckCorrectionRequest& 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.Cashier = "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 = "ККТ-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.Cashier = "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.Cashier = "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.Cashier = "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.Cashier = "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.Cashier = "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; // E
xtra 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' i
nside
    }

    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(b
3) << 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(b
3) << 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.	№ документа	Входящий № сопроводит. докум. и дата	Подп.	Дата
Изм.	измененных	замененных	новых	аннулированных					