МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)

Лабораторная работа № 1
дисциплина «Операционные системы»
по теме «Функции Win32 API для получения системной информации»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Михелев В.М.

Белгород 2019

Лабораторная работа № 1

«Функции Win32 API для получения системной информации»

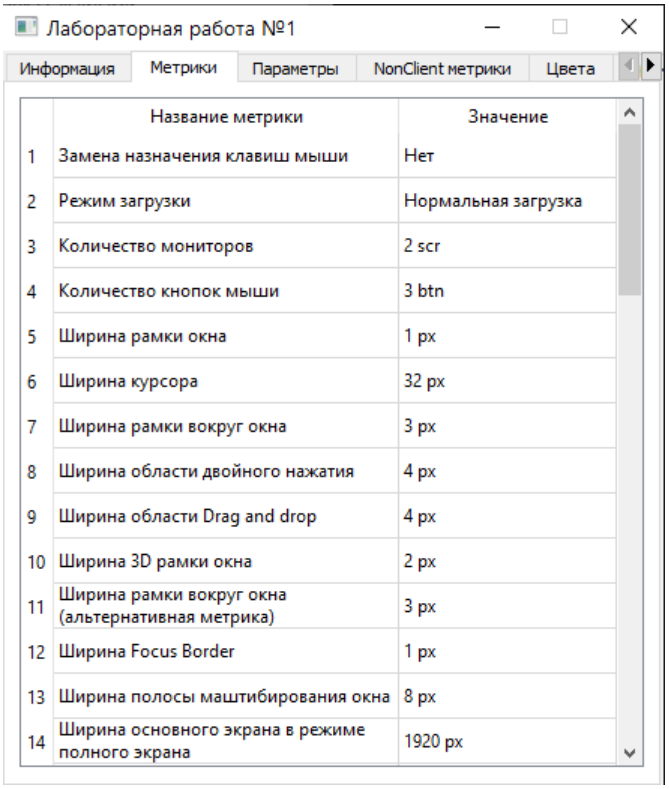
Цель работы: Получение практических навыков по программированию в Win32 API с использованием аппаратных и системных функций.

Вариант 9

Разработать программное обеспечение приложения, обеспечивающего получение следующей системной информации:

- Имя компьютера, имя пользователя;
- Пути к системным каталогам Windows;
- Версия операционной системы;
- Системные метрики (не менее 50 метрик);
- Системные параметры:
  - **SPI_GETFONTSMOOTHING**
  - **SPI_GETMOUSEHOVERTIME**
  - **SPI_GETTOGGLEKEYS**
  - **SPI_SETDOUBLECLICKTIME**
  - **SPI_SETKEYBOARDDELAY**
  - **SPI_SETNONCLIENTMETRICS**
- Системные цвета (*определить цвет для следующих символьных констант и изменить его на любой другой*):
  - **COLOR_3DSHADOW**
  - **COLOR_INFOBK**
  - **COLOR_MENU**
- Функции для работы со временем:
  - **GetSystemTime**
  - **GetTimeZoneInformation**
  - **SetLocalTime**
- Дополнительные API-функции:
  - **GetACP, GetKeyboardLayout**
  - **GetSystemPowerStatus**
  - **SetComputerName**

Ход работы



	Название метрики	Значение
1	Замена назначения клавиш мыши	Нет
2	Режим загрузки	Нормальная загрузка
3	Количество мониторов	2 scr
4	Количество кнопок мыши	3 btn
5	Ширина рамки окна	1 px
6	Ширина курсора	32 px
7	Ширина рамки вокруг окна	3 px
8	Ширина области двойного нажатия	4 px
9	Ширина области Drag and drop	4 px
10	Ширина 3D рамки окна	2 px
11	Ширина рамки вокруг окна (альтернативная метрика)	3 px
12	Ширина Focus Border	1 px
13	Ширина полосы маштибирования окна	8 px
14	Ширина основного экрана в режиме полного экрана	1920 px

Рис. 1: Окно программы

Приложение

Содержимое файла clock.cpp

```
#include "clock.hpp"

void SystemTime::s_set(){
    current_time.wYear = vYear->value();
    current_time.wMonth = vMonth->currentIndex()-1;
    current_time.wDay = vDay->value();
    current_time.wHour = vHour->value();
    current_time.wMinute = vMinute->value();
    current_time.wDayOfWeek = vDayOfWeek->currentIndex();
    SetLocalTime(&current_time);
};

void SystemTime::s_reset(){
    current_time.wYear = default_time.wYear;
    current_time.wMonth = default_time.wMonth;
    current_time.wDay = default_time.wDay;
    current_time.wHour = default_time.wHour;
    current_time.wMinute = default_time.wMinute;
    current_time.wDayOfWeek = default_time.wDayOfWeek;
    vYear->setValue(current_time.wYear);
    vMonth->setCurrentIndex(current_time.wMonth-1);
    vDay->setValue(current_time.wDay);
    vHour->setValue(current_time.wHour);
    vMinute->setValue(current_time.wMinute);
    vDayOfWeek->setCurrentIndex(current_time.wDayOfWeek);
    SetLocalTime(&current_time);
};

SystemTime::SystemTime(bool editLock,SYSTEMTIME time){

    default_time.wYear = time.wYear;
    default_time.wMonth = time.wMonth;
    default_time.wDay = time.wDay;
    default_time.wHour = time.wHour;
    default_time.wMinute = time.wMinute;
    default_time.wDayOfWeek = time.wDayOfWeek;

    current_time.wYear = default_time.wYear;
    current_time.wMonth = default_time.wMonth;
    current_time.wDay = default_time.wDay;
    current_time.wHour = default_time.wHour;
    current_time.wMinute = default_time.wMinute;
    current_time.wDayOfWeek = default_time.wDayOfWeek;

    this->editLock = editLock;

    layout = new QVBoxLayout;

    lDate = new QHBoxLayout;
    lTime = new QHBoxLayout;
    lDayOfWeek = new QHBoxLayout;
    lButtons = new QHBoxLayout;

    nDate = new QLabel;
    nTime = new QLabel;
    nDayOfWeek = new QLabel;
```

```
vDayOfWeek = new QComboBox;
vMonth = new QComboBox;

vYear = new QSpinBox;
vDay = new QSpinBox;
vHour = new QSpinBox;
vMinute = new QSpinBox;

reset = new QPushButton("Сброс");
set = new QPushButton("Установить время");

nDate->setText("Дата: ");
vYear->setRange(1601,30827);
vYear->setValue(current_time.wYear);
vMonth->addItem(QString("Январь"),QVariant(1));
vMonth->addItem(QString("Февраль"),QVariant(2));
vMonth->addItem(QString("Март"),QVariant(3));
vMonth->addItem(QString("Апрель"),QVariant(4));
vMonth->addItem(QString("Май"),QVariant(5));
vMonth->addItem(QString("Июнь"),QVariant(6));
vMonth->addItem(QString("Июль"),QVariant(7));
vMonth->addItem(QString("Август"),QVariant(8));
vMonth->addItem(QString("Сентябрь"),QVariant(9));
vMonth->addItem(QString("Октябрь"),QVariant(10));
vMonth->addItem(QString("Ноябрь"),QVariant(11));
vMonth->addItem(QString("Декабрь"),QVariant(12));
vMonth->setCurrentIndex(current_time.wMonth-1);
vDay->setRange(1,31);
vDay->setValue(current_time.wDay);

nTime->setText("Время: ");
vHour->setRange(0,23);
vHour->setValue(current_time.wHour);
vMinute->setRange(0,59);
vMinute->setValue(current_time.wMinute);

nDayOfWeek->setText("День недели: ");
vDayOfWeek->addItem(QString("Воскресенье"),QVariant(0));
vDayOfWeek->addItem(QString("Понедельник"),QVariant(1));
vDayOfWeek->addItem(QString("Вторник"),QVariant(2));
vDayOfWeek->addItem(QString("Среда"),QVariant(3));
vDayOfWeek->addItem(QString("Четверг"),QVariant(4));
vDayOfWeek->addItem(QString("Пятница"),QVariant(5));
vDayOfWeek->addItem(QString("Суббота"),QVariant(6));
vDayOfWeek->setCurrentIndex(current_time.wDayOfWeek);

lDate->addWidget(vYear);
lDate->addWidget(vMonth);
lDate->addWidget(vDay);

lTime->addWidget(vHour);
lTime->addWidget(vMinute);

lDayOfWeek->addWidget(nDayOfWeek);
lDayOfWeek->addWidget(vDayOfWeek);

lButtons->addWidget(set);
lButtons->addWidget(reset);
```

```
    layout->addWidget(nDate);
    layout->addLayout(lDate);
    layout->addWidget(nTime);
    layout->addLayout(lTime);
    layout->addLayout(lDayOfWeek);
    if(editLock){
        vMonth->setEnabled(false);
        vDayOfWeek->setEnabled(false);
        vYear->setEnabled(false);
        vDay->setEnabled(false);
        vHour->setEnabled(false);
        vMinute->setEnabled(false);
    }
    else{
        layout->addLayout(lButtons);
    }
    this->setLayout(layout);

    connect(set,SIGNAL(pressed()),this,SLOT(s_set()));
    connect(reset,SIGNAL(pressed()),this,SLOT(s_reset()));
};

TimezoneInfo::TimezoneInfo(){
    TIME_ZONE_INFORMATION timeZone;
    DWORD CurrentBias = GetTimeZoneInformation(&timeZone);
    layout = new QVBoxLayout;

    lCurrentBias = new QHBoxLayout;
    lStandartBias = new QHBoxLayout;
    lDaylightBias = new QHBoxLayout;

    nCurrentBias = new QLabel("Текущее смещение (относительно UTC):");
    nStandartBias = new QLabel("Величина стандартного смещения:");
    nDaylightBias = new QLabel("Величина смещения после перевода часов:");
    vCurrentBias = new QLabel;
    vStandartBias = new QLabel;
    vDaylightBias = new QLabel;

    switch(CurrentBias){
        case TIME_ZONE_ID_UNKNOWN:
            vCurrentBias->setText("Перевод времени не используется");
            break;
        case TIME_ZONE_ID_STANDARD:
            vCurrentBias->setText("Часы без перевода");
            break;
        case TIME_ZONE_ID_DAYLIGHT:
            vCurrentBias->setText("Часы с переводом");
            break;
    }
    vStandartBias->setText(QString::number(timeZone.StandardBias)+" мин.");
    vDaylightBias->setText(QString::number(timeZone.DaylightBias)+" мин.");

    lCurrentBias->addWidget(nCurrentBias);
    lCurrentBias->addWidget(vCurrentBias);
    lStandartBias->addWidget(nStandartBias);
    lStandartBias->addWidget(vStandartBias);
    lDaylightBias->addWidget(nDaylightBias);
    lDaylightBias->addWidget(vDaylightBias);
    layout->addLayout(lCurrentBias);
    layout->addLayout(lStandartBias);
```

```
    layout->addLayout(lDaylightBias);
    this->setLayout(layout);
};

SystemClock::SystemClock(){
    layout = new QVBoxLayout;

    GetSystemTime(&system_time);
    GetLocalTime(&local_time);

    systemTime = new SystemTime(true,system_time);
    localTime = new SystemTime(false,local_time);
    timeZone = new TimezoneInfo();

    layout->addWidget(new QLabel("Системное время: "));
    layout->addWidget(systemTime);
    layout->addWidget(new QLabel("Часовой пояс: "));
    layout->addWidget(timeZone);
    layout->addWidget(new QLabel("Локальное время: "));
    layout->addWidget(localTime);

    this->setLayout(layout);
};
```

Содержимое файла clock.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>

class SystemTime: public QWidget{
    Q_OBJECT
    private:
        bool editLock;
        SYSTEMTIME current_time,default_time;
        QVBoxLayout *layout;
        QHBoxLayout *lDate,*lTime,*lDayOfWeek,*lButtons;
        QLabel *nDate,*nTime,*nDayOfWeek;
        QComboBox *vDayOfWeek,*vMonth;
        QSpinBox *vYear,*vDay,*vHour,*vMinute;
        QPushButton *reset,*set;
    public slots:
        void s_set();
        void s_reset();
    public:
        SystemTime(bool editLock,SYSTEMTIME time);
};

class TimezoneInfo: public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QHBoxLayout *lCurrentBias,*lStandartBias,*lDaylightBias;
        QLabel *nCurrentBias,*nStandartBias,*nDaylightBias,
               *vCurrentBias,*vStandartBias,*vDaylightBias;
    public:
        TimezoneInfo();
};

class SystemClock: public QWidget{
```

```
    Q_OBJECT
    private:
        SYSTEMTIME system_time;
        SYSTEMTIME local_time;
        SYSTEMTIME default_local_time;
        QVBoxLayout *layout;
        SystemTime *systemTime,*localTime;
        TimezoneInfo *timeZone;
    public:
        SystemClock();
};
```

Содержимое файла colors.cpp

```
#include "colors.hpp"

void SystemColors::set_color1(){
    QColor color = QColorDialog::getColor();
    QT_SetSysColor(COLOR_3DSHADOW,color);
    c_3DShadow.setColor(QPalette::Background,color);
    v_3DShadow->setPalette(c_3DShadow);
    this->update();
};
void SystemColors::set_color2(){
    QColor color = QColorDialog::getColor();
    QT_SetSysColor(COLOR_INFOBK,color);
    c_infobk.setColor(QPalette::Background,color);
    v_infobk->setPalette(c_infobk);
    this->update();
};
void SystemColors::set_color3(){
    QColor color = QColorDialog::getColor();
    QT_SetSysColor(COLOR_MENU,color);
    c_menu.setColor(QPalette::Background,color);
    v_menu->setPalette(c_menu);
    this->update();
};
void SystemColors::s_reset(){

    QT_SetSysColor(COLOR_3DSHADOW,default_3DShadow.color(QPalette::Background));
    QT_SetSysColor(COLOR_INFOBK,default_infobk.color(QPalette::Background));
    QT_SetSysColor(COLOR_MENU,default_menu.color(QPalette::Background));

    c_3DShadow = default_3DShadow;
    v_3DShadow->setPalette(c_3DShadow);

    c_infobk = default_infobk;
    v_infobk->setPalette(c_infobk);

    c_menu = default_menu;
    v_menu->setPalette(c_menu);
    this->update();
};

QColor QT_GetSysColor(int index){
    QColor color;
    DWORD WIN_color;
    WIN_color = GetSysColor(index);
    int WIN_colorRed = GetRValue(WIN_color);
    int WIN_colorBlue = GetBValue(WIN_color);
```

```
    int WIN_colorGreen = GetGValue(WIN_color);
    color.setRgb(WIN_colorRed,WIN_colorBlue,WIN_colorGreen);
    return color;
};

void QT_SetSysColor(int index,QColor color){
    int red,green,blue;
    red = color.red();
    green = color.green();
    blue = color.blue();
    DWORD WIN_color = RGB(red,green,blue);
    SetSysColors(1,&index,&WIN_color);
};

SystemColors::SystemColors(){
    default_3DShadow.setColor(QPalette::Background,QT_GetSysColor(COLOR_3DSHADOW));
    default_infobk.setColor(QPalette::Background,QT_GetSysColor(COLOR_INFOBK));
    default_menu.setColor(QPalette::Background,QT_GetSysColor(COLOR_MENU));
    c_3DShadow = default_3DShadow;
    c_infobk = default_infobk;
    c_menu = default_menu;

    layout = new QVBoxLayout;

    l_3DShadow = new QHBoxLayout;
    l_infobk = new QHBoxLayout;
    l_menu = new QHBoxLayout;

    reset = new QPushButton("Сброс");
    b_3DShadow = new QPushButton("Выбор цвета");
    b_infobk = new QPushButton("Выбор цвета");
    b_menu = new QPushButton("Выбор цвета");

    n_3DShadow = new QLabel;
    n_infobk = new QLabel;
    n_menu = new QLabel;
    v_3DShadow = new QLabel;
    v_infobk = new QLabel;
    v_menu = new QLabel;

    v_3DShadow->setAutoFillBackground(true);
    v_3DShadow->setPalette(c_3DShadow);

    v_infobk->setAutoFillBackground(true);
    v_infobk->setPalette(c_infobk);

    v_menu->setAutoFillBackground(true);
    v_menu->setPalette(c_menu);

    n_3DShadow->setText("COLOR 3DSHADOW");
    n_infobk->setText("COLOR INFOBK");
    n_menu->setText("COLOR MENU");

    v_3DShadow->setText("3DSHADOW");
    v_3DShadow->setAlignment(Qt::AlignVCenter);
    v_infobk->setText("INFOBK");
    v_infobk->setAlignment(Qt::AlignVCenter);
    v_menu->setText("MENU");
    v_menu->setAlignment(Qt::AlignVCenter);
```

```
    l_3DShadow->addWidget(n_3DShadow);
    l_3DShadow->addWidget(v_3DShadow);
    l_infobk->addWidget(n_infobk);
    l_infobk->addWidget(v_infobk);
    l_menu->addWidget(n_menu);
    l_menu->addWidget(v_menu);

    layout->addLayout(l_3DShadow);
    layout->addWidget(b_3DShadow);
    layout->addLayout(l_infobk);
    layout->addWidget(b_infobk);
    layout->addLayout(l_menu);
    layout->addWidget(b_menu);
    layout->addWidget(reset);
    this->setLayout(layout);
    connect(b_3DShadow,SIGNAL(pressed()),this,SLOT(set_color1()));
    connect(b_infobk,SIGNAL(pressed()),this,SLOT(set_color2()));
    connect(b_menu,SIGNAL(pressed()),this,SLOT(set_color3()));
    connect(reset,SIGNAL(pressed()),this,SLOT(s_reset()));
};
```

Содержимое файла colors.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>

#pragma once

#define COLOR_INFOBK 24
#define COLOR_MENU 4

QColor QT_GetSysColor(int index);
void QT_SetSysColor(int index,QColor color);

class SystemColors : public QWidget{
    Q_OBJECT
    private:
    QVBoxLayout *layout;
    QHBoxLayout *l_3DShadow,*l_infobk,*l_menu;
    QLabel *color1,color2,color3;
    QPalette default_3DShadow,default_infobk,default_menu;
    QPalette c_3DShadow,c_infobk,c_menu;
    QLabel *n_3DShadow,*n_infobk,*n_menu,
           *v_3DShadow,*v_infobk,*v_menu;
    QPushButton *b_3DShadow,*b_infobk,*b_menu,*reset;
    public:
        SystemColors();
    public slots:
        void set_color1();
        void set_color2();
        void set_color3();
        void s_reset();
};
```

Содержимое файла etc.cpp

```
#include "etc.hpp"

void QT_SetComputerName(QString str){
```

```
    BOOL result;
    char char_str[str.length()];
    for(int i = 0; i < str.length(); i++)
        char_str[i] = str.at(i).toLatin1();
    result = SetComputerNameA(char_str);
    if(result == ERROR_INVALID_PARAMETER){
        QMessageBox msgBox;
        msgBox.setText("Превышена длина строки или строка содержит неподдерживаемые символы
        ↪  ");
        msgBox.exec();
    };
}

void SystemEtc::setComputerName(){
    CurrentComputerName = vSetComputerName->text();
    QT_SetComputerName(CurrentComputerName);
    this->computerName->setText(CurrentComputerName);
    QMessageBox msgBox;
    msgBox.setText(CurrentComputerName);
    msgBox.exec();
}

void SystemEtc::s_reset(){
    QT_SetComputerName(DefaultComputerName);
    this->computerName->setText(DefaultComputerName);
    QMessageBox msgBox;
    msgBox.setText(DefaultComputerName);
    msgBox.exec();
}

QString languageId(SHORT languageId){
    QString result;
    switch(languageId){
        case LANG_RUS:
            result = "Русская";
            break;
        case LANG_ENG:
            result = "Английская";
            break;
    };
    return result;
}

SHORT GetLanguageShort(HKL key){
    return (LOWORD(key));
};

QString codePages(int id){
    QString result;
    switch(id){
        case 437:
            result = "OEM United States";
            break;
        case 1201:
            result = "Unicode UTF-16, big endian";
            break;
        case 1200:
            result = "Unicode UTF-16, little endian";
            break;
        case 1251:
```

```
            result = "ANSI Cyrillic; Cyrillic (Windows)";
            break;
        case 866:
            result = "OEM Russian; Cyrillic (DOS)";
            break;
        case 65000:
            result = "Unicode (UTF-7)";
            break;
        case 65001:
            result = "Unicode (UTF-8)";
            break;
    };
    return result;
};

PowerStatus::PowerStatus(){
    SYSTEM_POWER_STATUS power_status;
    GetSystemPowerStatus(&power_status);

    QString vACLineStatus;
    QString vBatteryFlag;
    QString vBatteryLifePercent;
    QString vBatteryLifeTime;
    QString vBatteryFullLifeTime;

    switch(power_status.ACLineStatus){
        case 0:
            vACLineStatus = "От батареи";
            break;
        case 1:
            vACLineStatus = "От сети";
            break;
        case 255:
            vACLineStatus = "Неизвестно";
            break;
    };

    switch(power_status.BatteryFlag){
        case 1:
            vBatteryFlag = "Больше 66%";
            break;
        case 2:
            vBatteryFlag = "Меньше 33%";
            break;
        case 4:
            vBatteryFlag = "Меньше 5%";
            break;
        case 8:
            vBatteryFlag = "Заряжается";
            break;
        case 128:
            vBatteryFlag = "Нет батареи";
            break;
        case 255:
            vBatteryFlag = "Неизвестный статус";
            break;
    };

    if(power_status.BatteryLifePercent != 255){
        vBatteryLifePercent = QString::number(power_status.BatteryLifePercent)+" %";
```

```
}
else{
    vBatteryLifePercent = "Неизвестно";
};

if(power_status.BatteryLifeTime != -1){
    vBatteryLifeTime = QString::number(power_status.BatteryLifeTime)+" секунд";
}
else{
    vBatteryLifeTime = "Неизвестно";
};

if(power_status.BatteryFullLifeTime != -1){
    vBatteryFullLifeTime = QString::number(power_status.BatteryFullLifeTime)+" секунд";
}
else{
    vBatteryFullLifeTime = "Неизвестно";
};

layout = new QVBoxLayout;

lACLine = new QHBoxLayout;
lBatteryFlag = new QHBoxLayout;
lBatteryLifePercent = new QHBoxLayout;
lBatteryLifeTime = new QHBoxLayout;
lBatteryFullLifeTime = new QHBoxLayout;

ACLine = new QLabel;
BatteryFlag = new QLabel;
BatteryLifePercent = new QLabel;
BatteryLifeTime = new QLabel;
BatteryFullLifeTime = new QLabel;

nACLine = new QLabel;
nBatteryFlag = new QLabel;
nBatteryLifePercent = new QLabel;
nBatteryLifeTime = new QLabel;
nBatteryFullLifeTime = new QLabel;

nACLine->setText("Питание от сети: ");
nBatteryFlag->setText("Состояние аккумулятора: ");
nBatteryLifePercent->setText("Процент заряда аккумулятора: ");
nBatteryLifeTime->setText("Осталось времени работы: ");
nBatteryFullLifeTime->setText("Времени работы от полного заряда: ");


ACLine->setText(vACLineStatus);
BatteryFlag->setText(vBatteryFlag);
BatteryLifePercent->setText(vBatteryLifePercent);
BatteryLifeTime->setText(vBatteryLifeTime);
BatteryFullLifeTime->setText(vBatteryFullLifeTime);

lACLine->addWidget(nACLine);
lACLine->addWidget(ACLine);
lBatteryFlag->addWidget(nBatteryFlag);
lBatteryFlag->addWidget(BatteryFlag);
lBatteryLifePercent->addWidget(nBatteryLifePercent);
lBatteryLifePercent->addWidget(BatteryLifePercent);
lBatteryLifeTime->addWidget(nBatteryLifeTime);
lBatteryLifeTime->addWidget(BatteryLifeTime);
```

```
    lBatteryFullLifeTime->addWidget(nBatteryFullLifeTime);
    lBatteryFullLifeTime->addWidget(BatteryFullLifeTime);

    layout->addLayout(lACLine);
    layout->addLayout(lBatteryFlag);
    layout->addLayout(lBatteryLifePercent);
    layout->addLayout(lBatteryLifeTime);
    layout->addLayout(lBatteryFullLifeTime);

    this->setLayout(layout);
};


SystemEtc::SystemEtc(){
    DefaultComputerName = QT_GetComputerName();
    CurrentComputerName = DefaultComputerName;
    PowerStatus* ps = new PowerStatus;

    QVBoxLayout *layout = new QVBoxLayout;
    reset = new QPushButton("Сброс");


    lComputerName = new QHBoxLayout;
    lSetComputerName = new QHBoxLayout;
    lACP = new QHBoxLayout;
    lKeyboardLayout = new QHBoxLayout;

    ACP = new QLabel;
    nACP = new QLabel;
    computerName = new QLabel;
    nComputerName = new QLabel;
    keyboardLayout = new QLabel;
    nKeyboardLayout = new QLabel;

    vSetComputerName = new QLineEdit();
    bSetComputerName = new QPushButton("Установить имя");

    nComputerName->setText("Имя ПК: ");
    nACP->setText("Кодовая страница: ");
    nKeyboardLayout->setText("Раскладка клавиатуры: ");

    computerName->setText(CurrentComputerName);
    ACP->setText(codePages(GetACP()));
    keyboardLayout->setText(languageId(
                            GetLanguageShort(
                            GetKeyboardLayout(0))));

    lComputerName->addWidget(nComputerName);
    lComputerName->addWidget(computerName);

    lACP->addWidget(nACP);
    lACP->addWidget(ACP);

    lSetComputerName->addWidget(vSetComputerName);
    lSetComputerName->addWidget(bSetComputerName);

    lKeyboardLayout->addWidget(nKeyboardLayout);
    lKeyboardLayout->addWidget(keyboardLayout);

    layout->addLayout(lACP);
```

```
    layout->addLayout(lKeyboardLayout);
    layout->addWidget(ps);
    layout->addLayout(lComputerName);
    layout->addLayout(lSetComputerName);
    layout->addWidget(reset);
    this->setLayout(layout);
    QWidget::connect(reset,SIGNAL(pressed()),this,SLOT(s_reset()));
    QWidget::connect(bSetComputerName,SIGNAL(pressed()),this,SLOT(setComputerName()));
};
```

Содержимое файла etc.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "systemInfo.hpp"
#pragma once

#define LANG_RUS 0x0419
#define LANG_ENG 0x0409

SHORT GetLanguageShort(HKL key);

QString codePages(int id);

QString languageId(SHORT languageId);

class SystemEtc : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QLineEdit *scomputerName;
        QPushButton *bcomputerName,*reset;
        QLabel *computerName,*ACP,*keyboardLayout;
        QLabel *nComputerName, *nACP,*nKeyboardLayout;
        QHBoxLayout *lComputerName,*lSetComputerName,*lKeyboardLayout,*lACP;
        QPushButton *bSetComputerName;
        QLineEdit *vSetComputerName;
        QString DefaultComputerName;
        QString CurrentComputerName;
    public slots:
        void setComputerName();
        void s_reset();
    public:
        SystemEtc();
};

class PowerStatus:public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QHBoxLayout *lACLine,*lBatteryFlag,*lBatteryLifePercent,
                    *lSystemStatusFlag,*lBatteryLifeTime,*lBatteryFullLifeTime;
        QLabel *nACLine,*nBatteryFlag,*nBatteryLifePercent,
                    *nSystemStatusFlag,*nBatteryLifeTime,*nBatteryFullLifeTime;
        QLabel *ACLine,*BatteryFlag,*BatteryLifePercent,
                    *SystemStatusFlag,*BatteryLifeTime,*BatteryFullLifeTime;
    public:
        PowerStatus();
};
```

Содержимое файла main.cpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>
#include "systemInfo.hpp"
#include "colors.hpp"
#include "metrics.hpp"
#include "settings.hpp"
#include "settings2.hpp"
#include "clock.hpp"
#include "etc.hpp"

int main (int argc,char* argv[]){
    QApplication* app = new QApplication (argc,argv);
    QTabWidget* tabs = new QTabWidget;
    SystemInfo* p1 = new SystemInfo();
    SystemMetrics* p2 = new SystemMetrics();
    SystemSettings* p3 = new SystemSettings();
    SystemSettings2* p3_1 = new SystemSettings2();
    SystemColors* p4 = new SystemColors();
    SystemClock* p5 = new SystemClock();
    SystemEtc* p6 = new SystemEtc();


    tabs->addTab(p1,"Информация");
    tabs->addTab(p2,"Метрики");
    tabs->addTab(p3,"Параметры");
    tabs->addTab(p3_1,"NonClient метрики");
    tabs->addTab(p4,"Цвета");
    tabs->addTab(p5,"Время");
    tabs->addTab(p6,"Прочее");
    tabs->setFixedSize(440,490);
    tabs->setWindowTitle("Лабораторная работа №1");
    tabs->show();
    return app->exec();
}
```

Содержимое файла metrics.cpp

```
#include "metrics.hpp"

void QT_GetSystemMetrics(QTableWidget* table,int row,int index,QString header,QString suffix){
    int result = GetSystemMetrics(index);
    QTableWidgetItem* nameItem = new QTableWidgetItem(header);
    QTableWidgetItem* valueItem;
    table->setItem(row,0,nameItem);
    if (result==0){
        valueItem = new QTableWidgetItem(QString("Ошибка чтения"));
        table->setItem(row,1,valueItem);
    }
    else{
        valueItem = new QTableWidgetItem(QString::number(result)+" "+suffix);
        table->setItem(row,1,valueItem);
    }
};

void QT_GetSystemMetricBoot(QTableWidget* table,int row){
    int result = GetSystemMetrics(SM_CLEANBOOT);
    QTableWidgetItem* nameItem = new QTableWidgetItem(QString("Режим загрузки"));
```

```
    QTableWidgetItem* valueItem;
    table->setItem(row,0,nameItem);
    switch (result){
        case 0:
            valueItem = new QTableWidgetItem(QString("Нормальная загрузка"));
            break;
        case 1:
            valueItem = new QTableWidgetItem(QString("Безопасная загрузка"));
            break;
        case 2:
            valueItem = new QTableWidgetItem(QString("Безопасная загрузка с поддержкой сетевых
            ↪  драйверов"));
            break;
        };
    table->setItem(row,1,valueItem);
};

void QT_GetSystemMetricsBool(QTableWidget* table,int row,int index,QString header){
    int result = GetSystemMetrics(index);
    QTableWidgetItem* nameItem = new QTableWidgetItem(header);
    QTableWidgetItem* valueItem;
    table->setItem(row,0,nameItem);
    if (result==0){
        valueItem = new QTableWidgetItem(QString("Нет"));
        table->setItem(row,1,valueItem);
    }
    else{
        valueItem = new QTableWidgetItem(QString("Да"));
        table->setItem(row,1,valueItem);
    }
};
SystemMetrics::SystemMetrics(){
    QVBoxLayout *layout = new QVBoxLayout;
    table = new QTableWidget;
    table->setRowCount(50);
    table->setColumnCount(2);
    table->setColumnWidth(0,230);
    table->setColumnWidth(1,145);
    table->setHorizontalHeaderLabels({"Название метрики","Значение"});
    table->setEditTriggers(QAbstractItemView::NoEditTriggers);
    QT_GetSystemMetricsBool(table,0,SM_SWAPBUTTON,QString("Замена назначения клавиш мыши"));
    QT_GetSystemMetricBoot(table,1);
    QT_GetSystemMetrics(table,2,SM_CMONITORS,QString("Количество мониторов"),QString("scr"));
    QT_GetSystemMetrics(table,3,SM_CMOUSEBUTTONS,QString("Количество кнопок
    ↪  мыши"),QString("btn"));
    QT_GetSystemMetrics(table,4,SM_CXBORDER,QString("Ширина рамки окна"),QString("px"));
    QT_GetSystemMetrics(table,5,SM_CXCURSOR,QString("Ширина курсора"),QString("px"));
    QT_GetSystemMetrics(table,6,SM_CXDLGFRAME,QString("Ширина рамки вокруг
    ↪  окна"),QString("px"));
    QT_GetSystemMetrics(table,7,SM_CXDOUBLECLK,QString("Ширина области двойного
    ↪  нажатия"),QString("px"));
    QT_GetSystemMetrics(table,8,SM_CXDRAG,QString("Ширина области Drag and
    ↪  drop"),QString("px"));
    QT_GetSystemMetrics(table,9,SM_CXEDGE,QString("Ширина 3D рамки окна"),QString("px"));
    QT_GetSystemMetrics(table,10,SM_CXFIXEDFRAME,QString("Ширина рамки вокруг окна
    ↪  (альтернативная метрика)"),QString("px"));
    QT_GetSystemMetrics(table,11,SM_CXFOCUSBORDER,QString("Ширина Focus
    ↪  Border"),QString("px"));
    QT_GetSystemMetrics(table,12,SM_CXFRAME,QString("Ширина полосы маштибирования
    ↪  окна"),QString("px"));
```

```
QT_GetSystemMetrics(table,13,SM_CXFULLSCREEN,QString("Ширина основного экрана в режиме
↪  полного экрана"),QString("px"));
QT_GetSystemMetrics(table,14,SM_CXHSCROLL,QString("Ширина кнопки полосы
↪  прокрутки"),QString("px"));
QT_GetSystemMetrics(table,15,SM_CXHTHUMB,QString("Ширина полосы
↪  прокрутки"),QString("px"));
QT_GetSystemMetrics(table,16,SM_CXICON,QString("Ширина иконки"),QString("px"));
QT_GetSystemMetrics(table,17,SM_CXICONSPACING,QString("Ширина ячейки сетки
↪  иконок"),QString("px"));
QT_GetSystemMetrics(table,18,SM_CXMAXIMIZED,QString("Ширина развернутого
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,19,SM_CXMAXTRACK,QString("Максимальная ширина окна с учетом
↪  рамок"),QString("px"));
QT_GetSystemMetrics(table,20,SM_CXMENUCHECK,QString("Ширина контекстного
↪  меню"),QString("px"));
QT_GetSystemMetrics(table,21,SM_CXMENUSIZE,QString("Ширина кнопок меню"),QString("px"));
QT_GetSystemMetrics(table,22,SM_CXMIN,QString("Минимальная ширина окна"),QString("px"));
QT_GetSystemMetrics(table,23,SM_CXMINIMIZED,QString("Ширина минимизированного
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,24,SM_CXMINSPACING,QString("Ширина сетки минимизированного
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,25,SM_CXMINTRACK,QString("Минимальная ширина для перетаскивания
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,26,SM_CXSCREEN,QString("Ширина основного
↪  экрана"),QString("px"));
QT_GetSystemMetrics(table,27,SM_CXSIZE,QString("Ширина кнопки в заголовке
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,28,SM_CXSIZEFRAME,QString("Ширина полосы маштибирования окна
↪  (альтернативная метрика)"),QString("px"));
QT_GetSystemMetrics(table,29,SM_CXSMICON,QString("Рекомендованная ширина иконки (иконка в
↪  заголовке окна)"),QString("px"));
QT_GetSystemMetrics(table,30,SM_CXSMSIZE,QString("Ширина кнопок управления
↪  окном"),QString("px"));
QT_GetSystemMetrics(table,31,SM_CXVIRTUALSCREEN,QString("Сумма ширины всех
↪  экранов"),QString("px"));
QT_GetSystemMetrics(table,32,SM_CXVSCROLL,QString("Ширина полосы
↪  прокрутки"),QString("px"));
QT_GetSystemMetrics(table,33,SM_CYBORDER,QString("Высота рамки окна"),QString("px"));
QT_GetSystemMetrics(table,34,SM_CYCAPTION,QString("Высота заголовка окна"),QString("px"));
QT_GetSystemMetrics(table,35,SM_CYCURSOR,QString("Высота курсора"),QString("px"));
QT_GetSystemMetrics(table,36,SM_CYDLGFRAME,QString("Высота рамки вокруг
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,37,SM_CYDOUBLECLK,QString("Высота области двойного
↪  нажатия"),QString("px"));
QT_GetSystemMetrics(table,38,SM_CYDRAG,QString("Высота области Drag and
↪  drop"),QString("px"));
QT_GetSystemMetrics(table,39,SM_CYEDGE,QString("Высота 3D рамки окна"),QString("px"));
QT_GetSystemMetrics(table,40,SM_CYFIXEDFRAME,QString("Высота рамки вокруг окна
↪  (альтернативная метрика)"),QString("px"));
QT_GetSystemMetrics(table,41,SM_CYFOCUSBORDER,QString("Высота Focus
↪  Border"),QString("px"));
QT_GetSystemMetrics(table,42,SM_CYFRAME,QString("Высота полосы маштибирования
↪  окна"),QString("px"));
QT_GetSystemMetrics(table,43,SM_CYFULLSCREEN,QString("Высота основного экрана в режиме
↪  полного экрана"),QString("px"));
QT_GetSystemMetrics(table,44,SM_CYMENUCHECK,QString("Высота контекстного
↪  меню"),QString("px"));
QT_GetSystemMetrics(table,45,SM_CYHSCROLL,QString("Высота кнопки полосы
↪  прокрутки"),QString("px"));
QT_GetSystemMetrics(table,46,SM_CYICON,QString("Высота иконки"),QString("px"));
```

```
    QT_GetSystemMetrics(table,47,SM_CYICONSPACING,QString("Высота ячейки сетки
    ↪  иконок"),QString("px"));
    QT_GetSystemMetrics(table,48,SM_CYMAXIMIZED,QString("Высота развернутого
    ↪  окна"),QString("px"));
    QT_GetSystemMetrics(table,49,SM_CYMAXTRACK,QString("Максимальная высота окна с учетом
    ↪  рамок"),QString("px"));
    table->horizontalHeader()->setSectionResizeMode (QHeaderView::Fixed);
    table->verticalHeader()->setSectionResizeMode (QHeaderView::Fixed);
    layout->addWidget(table);
    this->setLayout(layout);
};
```

Содержимое файла metrics.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>

#pragma once

#define SM_SWAPBUTTON 23
#define SM_CLEANBOOT 67
#define SM_CMONITORS 80
#define SM_CMOUSEBUTTONS 43
#define SM_CXBORDER 5
#define SM_CXCURSOR 13
#define SM_CXDLGFRAME 7
#define SM_CXDOUBLECLK 36
#define SM_CXDRAG 68
#define SM_CXEDGE 45
#define SM_CXFOCUSBORDER 83
#define SM_CXFRAME 32
#define SM_CXFULLSCREEN 16
#define SM_CXHSCROLL 21
#define SM_CXHTHUMB 10
#define SM_CXICON 11
#define SM_CXICONSPACING 38
#define SM_CXMAXIMIZED 61
#define SM_CXMAXTRACK 59
#define SM_CXMENUCHECK 71
#define SM_CXMENUSIZE 54
#define SM_CXMIN 28
#define SM_CXMINIMIZED 57
#define SM_CXMINSPACING 47
#define SM_CXMINTRACK 34
#define SM_CXSCREEN 0
#define SM_CXSIZE 30
#define SM_CXSMICON 49
#define SM_CXSMSIZE 52
#define SM_CXVIRTUALSCREEN 78
#define SM_CXVSCROLL 2
#define SM_CYBORDER 6
#define SM_CYCAPTION 4
#define SM_CYCURSOR 14
#define SM_CYDLGFRAME 8
#define SM_CYDOUBLECLK 37
#define SM_CYDRAG 69
#define SM_CYEDGE 46
#define SM_CYFOCUSBORDER 84
#define SM_CYFRAME 33
```

```
#define SM_CYFULLSCREEN 17
#define SM_CYHSCROLL 3
#define SM_CYICON 12
#define SM_CYICONSPACING 39
#define SM_CYMAXIMIZED 62
#define SM_CYMAXTRACK 60
#define SM_CYMENU 15
#define SM_CYMENUCHECK 72


void QT_GetSystemMetrics(QTableWidget* table,int row,int index,QString header,QString suffix);
void QT_GetSystemMetricsBool(QTableWidget* table,int row,int index,QString header);
void QT_GetSystemMetricBoot(QTableWidget* table,int row);

class SystemMetrics : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QTableWidget* table;
    public:
        SystemMetrics();
};
```

Содержимое файла settings.cpp

```
#include "settings.hpp"

void SystemSettings::setKeyboardDelay(){
    CurrentKeyboardDelay = SetKeyboardDelay->value();
    SystemParametersInfoA(SPI_SETKEYBOARDDELAY,CurrentKeyboardDelay,0,0);
    KeyboardDelay->setText(QString::number(CurrentKeyboardDelay));
};
void SystemSettings::setDoubleClickTime(){
    CurrentDoubleClickTime = SetDoubleClickDelay->value();
    SystemParametersInfoA(SPI_SETDOUBLECLICKTIME,CurrentDoubleClickTime,0,0);
    DoubleClickTime->setText(QString::number(CurrentDoubleClickTime)+" мс");
};
void SystemSettings::s_reset(){
    CurrentDoubleClickTime = DefaultDoubleClickTime;
    CurrentKeyboardDelay = DefaultKeyboardDelay;
    SystemParametersInfoA(SPI_SETKEYBOARDDELAY,CurrentKeyboardDelay,0,0);
    SystemParametersInfoA(SPI_SETDOUBLECLICKTIME,CurrentDoubleClickTime,0,0);
    SetKeyboardDelay->setValue(DefaultDoubleClickTime);
    SetDoubleClickDelay->setValue(DefaultKeyboardDelay);
    DoubleClickTime->setText(QString::number(CurrentDoubleClickTime)+" мс");
    KeyboardDelay->setText(QString::number(CurrentKeyboardDelay));
};

QString QT_GetFontSmoothing(){
    QString result;
    BOOL flag;
    SystemParametersInfo(SPI_GETFONTSMOOTHING,0,&flag,0);
    if(flag == TRUE){
        result = "Включено";
    }
    else{
        result = "Выключено";
    }
    return result;
};
```

```
QString QT_GetMouseOvertime(){
    QString result;
    UINT value;
    SystemParametersInfo(SPI_GETMOUSEHOVERTIME,0,&value,0);
    result = QString::number(value)+" мс";
    return result;
};

QString QT_GetToggleKeys(){
    QString result;
    TOGGLEKEYS value;
    SystemParametersInfo(SPI_GETTOGGLEKEYS,0,&value,0);
    bool flag = ( ((value.dwFlags) & ( 0x1 << (0x00000002) )) !=0 );
    if(flag){
        result = "Включено";
    }
    else{
        result = "Выключено";
    }
    return result;
};

SystemSettings::SystemSettings(){
    QVBoxLayout *layout = new QVBoxLayout;

    lFontSmoothing = new QHBoxLayout;
    lMouseOvertime = new QHBoxLayout;
    lToggleKeys = new QHBoxLayout;
    lDoubleClickDelay = new QHBoxLayout;
    lKeyboardDelay = new QHBoxLayout;
    lSetDoubleClickTime = new QHBoxLayout;
    lSetKeyboardDelay = new QHBoxLayout;

    nFontSmoothing = new QLabel;
    nMouseOvertime = new QLabel;
    nToggleKeys = new QLabel;
    nDoubleClickTime = new QLabel;
    nKeyboardDelay = new QLabel;

    FontSmoothing = new QLabel;
    MouseOvertime = new QLabel;
    ToggleKeys = new QLabel;
    DoubleClickTime = new QLabel;
    KeyboardDelay = new QLabel;

    ButtonKeyboardDelay = new QPushButton("Установить");
    ButtonDoubleClickDelay = new QPushButton("Установить");
    reset = new QPushButton("Сброс");

    SetDoubleClickDelay = new QSpinBox;
    SetDoubleClickDelay->setSuffix(" мс");
    SetDoubleClickDelay->setSingleStep(10);
    SetDoubleClickDelay->setMinimum(10);
    SetDoubleClickDelay->setMaximum(5000);

    SetKeyboardDelay = new QSpinBox;
    SetKeyboardDelay->setMinimum(0);
    SetKeyboardDelay->setMaximum(3);
```

```
    DefaultDoubleClickTime = GetDoubleClickTime();
    CurrentDoubleClickTime = DefaultDoubleClickTime;

    SystemParametersInfoA(SPI_GETKEYBOARDDELAY,0,&DefaultKeyboardDelay,0);
    CurrentKeyboardDelay = DefaultKeyboardDelay;

    nFontSmoothing->setText("Сглаживание шрифтов: ");
    nMouseOvertime->setText("Mouse Overtime: ");
    nToggleKeys->setText("Звуковое оповещение о включении lock-клавиш: ");
    nDoubleClickTime->setText("Промежуток срабатывания двойного клика: ");
    nKeyboardDelay->setText("Задержка клавиатуры (0-3) (0 - 250мс, 1 ед. - 250мс): ");

    FontSmoothing->setText(QT_GetFontSmoothing());
    MouseOvertime->setText(QT_GetMouseOvertime());
    ToggleKeys->setText(QT_GetToggleKeys());
    DoubleClickTime->setText(QString::number(CurrentDoubleClickTime)+" мс");
    KeyboardDelay->setText(QString::number(CurrentKeyboardDelay));

    lFontSmoothing->addWidget(nFontSmoothing);
    lFontSmoothing->addWidget(FontSmoothing);
    lMouseOvertime->addWidget(nMouseOvertime);
    lMouseOvertime->addWidget(MouseOvertime);
    lToggleKeys->addWidget(nToggleKeys);
    lToggleKeys->addWidget(ToggleKeys);
    lDoubleClickDelay->addWidget(nDoubleClickTime);
    lDoubleClickDelay->addWidget(DoubleClickTime);
    lKeyboardDelay->addWidget(nKeyboardDelay);
    lKeyboardDelay->addWidget(KeyboardDelay);
    lSetDoubleClickTime->addWidget(SetDoubleClickDelay);
    lSetDoubleClickTime->addWidget(ButtonDoubleClickDelay);
    lSetKeyboardDelay->addWidget(SetKeyboardDelay);
    lSetKeyboardDelay->addWidget(ButtonKeyboardDelay);

    layout->addLayout(lFontSmoothing);
    layout->addLayout(lMouseOvertime);
    layout->addLayout(lToggleKeys);
    layout->addLayout(lDoubleClickDelay);
    layout->addLayout(lSetDoubleClickTime);
    layout->addLayout(lKeyboardDelay);
    layout->addLayout(lSetKeyboardDelay);
    layout->addWidget(reset);

    this->setLayout(layout);

    connect(reset,SIGNAL(clicked()),this,SLOT(s_reset()));
    connect(ButtonKeyboardDelay,SIGNAL(pressed()),this,SLOT(setKeyboardDelay()));
    connect(ButtonDoubleClickDelay,SIGNAL(pressed()),this,SLOT(setDoubleClickTime()));
};
```

Содержимое файла settings.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>

#pragma once

#define SPI_GETFONTSMOOTHING 0x004A
#define SPI_GETMOUSEHOVERTIME 0x0066
#define SPI_GETTOGGLEKEYS 0x0034
```

```
#define SPI_SETDOUBLECLICKTIME 0x0020
#define SPI_SETKEYBOARDDELAY 0x0017
#define SPI_GETKEYBOARDDELAY 0x0016


QString QT_GetFontSmoothing();
QString QT_GetMouseOvertime();
QString QT_GetToggleKeys();

class SystemSettings : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QHBoxLayout *lFontSmoothing,*lMouseOvertime,*lToggleKeys,
                    *lDoubleClickDelay,*lKeyboardDelay,
                    *lSetDoubleClickTime,*lSetKeyboardDelay;
        QLabel *nFontSmoothing,*nMouseOvertime,*nToggleKeys,
               *nDoubleClickTime,*nKeyboardDelay;
        QLabel *FontSmoothing,*MouseOvertime,*ToggleKeys,
               *DoubleClickTime,*KeyboardDelay;
        QSpinBox *SetDoubleClickDelay,*SetKeyboardDelay;
        QPushButton *ButtonDoubleClickDelay,*ButtonKeyboardDelay,*reset;
        UINT DefaultDoubleClickTime;
        int DefaultKeyboardDelay;
        UINT CurrentDoubleClickTime;
        int CurrentKeyboardDelay;
    public:
        SystemSettings();
    public slots:
        void setKeyboardDelay();
        void setDoubleClickTime();
        void s_reset();
};
```

Содержимое файла settings2.cpp

```
#include "settings2.hpp"

void SystemSettings2::s_set(){
    CurrentNonClientMetrics.iBorderWidth = bBorderWidth->value();
    CurrentNonClientMetrics.iScrollWidth = bScrollWidth->value();
    CurrentNonClientMetrics.iScrollHeight = bScrollHeight->value();
    CurrentNonClientMetrics.iCaptionWidth = bCaptionWidth->value();
    CurrentNonClientMetrics.iCaptionHeight = bCaptionHeight->value();
    CurrentNonClientMetrics.iSmCaptionWidth = bSmCaptionWidth->value();
    CurrentNonClientMetrics.iSmCaptionHeight = bSmCaptionHeight->value();
    CurrentNonClientMetrics.iMenuWidth = bMenuWidth->value();
    CurrentNonClientMetrics.iMenuHeight = bMenuHeight->value();
    SystemParametersInfoA(SPI_SETNONCLIENTMETRICS,0,&CurrentNonClientMetrics,0);
};

void SystemSettings2::s_reset(){
    SystemParametersInfoA(SPI_SETNONCLIENTMETRICS,0,&DefaultNonClientMetrics,0);
    CurrentNonClientMetrics.iBorderWidth = DefaultNonClientMetrics.iBorderWidth;
    CurrentNonClientMetrics.iScrollWidth = DefaultNonClientMetrics.iScrollWidth;
    CurrentNonClientMetrics.iScrollHeight = DefaultNonClientMetrics.iScrollHeight;
    CurrentNonClientMetrics.iCaptionWidth = DefaultNonClientMetrics.iCaptionWidth;
    CurrentNonClientMetrics.iCaptionHeight = DefaultNonClientMetrics.iCaptionHeight;
    CurrentNonClientMetrics.iSmCaptionWidth = DefaultNonClientMetrics.iSmCaptionWidth;
    CurrentNonClientMetrics.iSmCaptionHeight = DefaultNonClientMetrics.iSmCaptionHeight;
```

```
    CurrentNonClientMetrics.iMenuWidth = DefaultNonClientMetrics.iMenuWidth;
    CurrentNonClientMetrics.iMenuHeight = DefaultNonClientMetrics.iMenuHeight;
    bBorderWidth->setValue(CurrentNonClientMetrics.iBorderWidth);
    bScrollWidth->setValue(CurrentNonClientMetrics.iScrollWidth);
    bScrollHeight->setValue(CurrentNonClientMetrics.iScrollHeight);
    bCaptionWidth->setValue(CurrentNonClientMetrics.iCaptionWidth);
    bCaptionHeight->setValue(CurrentNonClientMetrics.iCaptionHeight);
    bSmCaptionWidth->setValue(CurrentNonClientMetrics.iSmCaptionWidth);
    bSmCaptionHeight->setValue(CurrentNonClientMetrics.iSmCaptionHeight);
    bMenuWidth->setValue(CurrentNonClientMetrics.iMenuWidth);
    bMenuHeight->setValue(CurrentNonClientMetrics.iMenuHeight);
};

SystemSettings2::SystemSettings2(){
    QVBoxLayout *layout = new QVBoxLayout;
    SystemParametersInfoA(SPI_GETNONCLIENTMETRICS,0,&DefaultNonClientMetrics,0);
    DefaultNonClientMetrics.cbSize=sizeof(NONCLIENTMETRICSA);
    CurrentNonClientMetrics.cbSize=sizeof(NONCLIENTMETRICSA);


    set = new QPushButton("Установить");
    reset = new QPushButton("Сброс");

    CurrentNonClientMetrics.iBorderWidth = DefaultNonClientMetrics.iBorderWidth;
    CurrentNonClientMetrics.iScrollWidth = DefaultNonClientMetrics.iScrollWidth;
    CurrentNonClientMetrics.iScrollHeight = DefaultNonClientMetrics.iScrollHeight;
    CurrentNonClientMetrics.iCaptionWidth = DefaultNonClientMetrics.iCaptionWidth;
    CurrentNonClientMetrics.iCaptionHeight = DefaultNonClientMetrics.iCaptionHeight;
    CurrentNonClientMetrics.iSmCaptionWidth = DefaultNonClientMetrics.iSmCaptionWidth;
    CurrentNonClientMetrics.iSmCaptionHeight = DefaultNonClientMetrics.iSmCaptionHeight;
    CurrentNonClientMetrics.iMenuWidth = DefaultNonClientMetrics.iMenuWidth;
    CurrentNonClientMetrics.iMenuHeight = DefaultNonClientMetrics.iMenuHeight;

    lBorderWidth = new QHBoxLayout;
    lScrollWidth = new QHBoxLayout;
    lScrollHeight = new QHBoxLayout;
    lCaptionWidth = new QHBoxLayout;
    lCaptionHeight = new QHBoxLayout;
    lSmCaptionWidth = new QHBoxLayout;
    lSmCaptionHeight = new QHBoxLayout;
    lMenuWidth = new QHBoxLayout;
    lMenuHeight = new QHBoxLayout;
    lButton = new QHBoxLayout;

    nBorderWidth = new QLabel;      nBorderWidth->setText("BorderWidth");
    nScrollWidth = new QLabel;      nScrollWidth->setText("ScrollWidth");
    nScrollHeight = new QLabel;     nScrollHeight->setText("ScrollHeight");
    nCaptionWidth = new QLabel;     nCaptionWidth->setText("CaptionWidth");
    nCaptionHeight = new QLabel;    nCaptionHeight->setText("CaptionHeight");
    nSmCaptionWidth = new QLabel;   nSmCaptionWidth->setText("SmCaptionWidth");
    nSmCaptionHeight = new QLabel;  nSmCaptionHeight->setText("SmCaptionHeight");
    nMenuWidth = new QLabel;        nMenuWidth->setText("MenuWidth");
    nMenuHeight = new QLabel;       nMenuHeight->setText("MenuHeight");

    bBorderWidth = new QSpinBox;
    bBorderWidth->setRange(1,100);
    bBorderWidth->setValue(CurrentNonClientMetrics.iBorderWidth);
    bScrollWidth = new QSpinBox;
    bScrollWidth->setRange(1,100);
    bScrollWidth->setValue(CurrentNonClientMetrics.iScrollWidth);
```

```
    bScrollHeight = new QSpinBox;
    bScrollHeight->setRange(1,100);
    bScrollHeight->setValue(CurrentNonClientMetrics.iScrollHeight);
    bCaptionWidth = new QSpinBox;
    bCaptionWidth->setRange(1,100);
    bCaptionWidth->setValue(CurrentNonClientMetrics.iCaptionWidth);
    bCaptionHeight = new QSpinBox;
    bCaptionHeight->setRange(1,100);
    bCaptionHeight->setValue(CurrentNonClientMetrics.iCaptionHeight);
    bSmCaptionWidth = new QSpinBox;
    bSmCaptionWidth->setRange(1,100);
    bSmCaptionWidth->setValue(CurrentNonClientMetrics.iSmCaptionWidth);
    bSmCaptionHeight = new QSpinBox;
    bSmCaptionHeight->setRange(1,100);
    bSmCaptionHeight->setValue(CurrentNonClientMetrics.iSmCaptionHeight);
    bMenuWidth = new QSpinBox;
    bMenuWidth->setRange(1,100);
    bMenuWidth->setValue(CurrentNonClientMetrics.iMenuWidth);
    bMenuHeight = new QSpinBox;
    bMenuHeight->setRange(1,100);
    bMenuHeight->setValue(CurrentNonClientMetrics.iMenuHeight);

    lBorderWidth->addWidget(nBorderWidth);          lBorderWidth->addWidget(bBorderWidth);
    lScrollWidth->addWidget(nScrollWidth);          lScrollWidth->addWidget(bScrollWidth);
    lScrollHeight->addWidget(nScrollHeight);        lScrollHeight->addWidget(bScrollHeight);
    lCaptionWidth->addWidget(nCaptionWidth);        lCaptionWidth->addWidget(bCaptionWidth);
    lCaptionHeight->addWidget(nCaptionHeight);      lCaptionHeight->addWidget(bCaptionHeight);
    lSmCaptionWidth->addWidget(nSmCaptionWidth);
    ↪  lSmCaptionWidth->addWidget(bSmCaptionWidth);
    lSmCaptionHeight->addWidget(nSmCaptionHeight);
    ↪  lSmCaptionHeight->addWidget(bSmCaptionHeight);
    lMenuWidth->addWidget(nMenuWidth);              lMenuWidth->addWidget(bMenuWidth);
    lMenuHeight->addWidget(nMenuHeight);            lMenuHeight->addWidget(bMenuHeight);
    lButton->addWidget(set);
    lButton->addWidget(reset);

    layout->addLayout(lBorderWidth);
    layout->addLayout(lScrollWidth);
    layout->addLayout(lScrollHeight);
    layout->addLayout(lCaptionWidth);
    layout->addLayout(lCaptionHeight);
    layout->addLayout(lSmCaptionWidth);
    layout->addLayout(lSmCaptionHeight);
    layout->addLayout(lMenuWidth);
    layout->addLayout(lMenuHeight);
    layout->addLayout(lButton);

    this->setLayout(layout);

    connect(set,SIGNAL(pressed()),this,SLOT(s_set()));
    connect(reset,SIGNAL(pressed()),this,SLOT(s_reset()));
};
```

Содержимое файла settings2.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>

#pragma once
```

```
#define SPI_SETNONCLIENTMETRICS 0x002A
#define SPI_GETNONCLIENTMETRICS 0x0029

class SystemSettings2 : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        NONCLIENTMETRICSA DefaultNonClientMetrics;
        NONCLIENTMETRICSA CurrentNonClientMetrics;
        QHBoxLayout *lBorderWidth,*lScrollWidth,*lScrollHeight,
                    *lCaptionWidth,*lCaptionHeight,*lSmCaptionWidth,
                    *lSmCaptionHeight,*lMenuWidth,*lMenuHeight,*lButton;
        QLabel *nBorderWidth,*nScrollWidth,*nScrollHeight,
               *nCaptionWidth,*nCaptionHeight,*nSmCaptionWidth,
               *nSmCaptionHeight,*nMenuWidth,*nMenuHeight;
        QSpinBox *bBorderWidth,*bScrollWidth,*bScrollHeight,
                 *bCaptionWidth,*bCaptionHeight,*bSmCaptionWidth,
                 *bSmCaptionHeight,*bMenuWidth,*bMenuHeight;
        QPushButton *set,*reset;
    public slots:
        void s_set();
        void s_reset();
    public:
        SystemSettings2();
};
```

Содержимое файла systemInfo.cpp

```
#include "system_info.hpp"

QString QT_GetUserName(){
    wchar_t str[4096];
    DWORD size = 4096;
    QString result_str;
    if(GetUserNameW(str, &size)){
        result_str.fromWCharArray(str,size);
    }
    else{
        result_str = "GET_ERROR";
    }
    return result_str;
};

QString QT_GetComputerName(){
    char str[4096];
    DWORD size = 4096;
    QString result_str;
    if(GetComputerNameA(str, &size)){
        result_str = str;
    }
    else{
        result_str = "GET_ERROR";
    }
    return result_str;
};

QString QT_GetSystemDirectory(){
    char str[4096];
    UINT size = 4096;
```

```
    QString result_str;
    if(GetSystemDirectoryA(str, size)){
        result_str = str;
    }
    else{
        result_str = "GET_ERROR";
    }
    return result_str;
};

QString QT_GetSystemWindowsDirectory(){
    char str[4096];
    UINT size = 4096;
    QString result_str;
    if(GetSystemWindowsDirectoryA(str, size)){
        result_str = str;
    }
    else{
        result_str = "GET_ERROR";
    }
    return result_str;
};

QString QT_GetTempPath(){
    char str[4096];
    DWORD size = 4096;
    QString result_str;
    if(GetTempPathA(size, str)){
        result_str = str;
    }
    else{
        result_str = "GET_ERROR";
    }
    return result_str;
};

QString QT_GetVersionExA(){
    OSVERSIONINFOA osvi;
    ZeroMemory(&osvi, sizeof(OSVERSIONINFOA));
    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOA);
    GetVersionExA(&osvi);
    QString result_str = QString::number(osvi.dwMajorVersion) +"."
                        +QString::number(osvi.dwMinorVersion) +" ("
                        +QString::number(osvi.dwBuildNumber)  +")";
    return result_str;
};

SystemInfo::SystemInfo(){
    QVBoxLayout *layout = new QVBoxLayout;

    QHBoxLayout *lComputerName = new QHBoxLayout;
    QHBoxLayout *lUserName = new QHBoxLayout;
    QHBoxLayout *lSystemDirectory = new QHBoxLayout;
    QHBoxLayout *lSystemWindowsDirectory = new QHBoxLayout;
    QHBoxLayout *lTempPath = new QHBoxLayout;
    QHBoxLayout *lVersion = new QHBoxLayout;

    QLabel *computerName = new QLabel;
    QLabel *userName = new QLabel;
    QLabel *systemDirectory = new QLabel;
```

```
    QLabel *systemWindowsDirectory = new QLabel;
    QLabel *tempPath = new QLabel;
    QLabel *version = new QLabel;

    QLabel *nComputerName = new QLabel;
    QLabel *nUserName = new QLabel;
    QLabel *nSystemDirectory = new QLabel;
    QLabel *nSystemWindowsDirectory = new QLabel;
    QLabel *nTempPath = new QLabel;
    QLabel *nVersion = new QLabel;

    nComputerName->setText("Имя ПК: ");
    nUserName->setText("Имя пользователя: ");
    nSystemDirectory->setText("Путь к системной папке: ");
    nSystemWindowsDirectory->setText("Путь к папке Windows: ");
    nTempPath->setText("Путь к папке temp: ");
    nVersion->setText("Версия ОС: ");

    computerName->setText(QT_GetComputerName());
    userName->setText(QT_GetUserName());
    systemDirectory->setText(QT_GetSystemDirectory());
    systemWindowsDirectory->setText(QT_GetSystemWindowsDirectory());
    tempPath->setText(QT_GetTempPath());
    version->setText(QT_GetVersionExA());

    lComputerName->addWidget(nComputerName);
    lComputerName->addWidget(computerName);
    lUserName->addWidget(nUserName);
    lUserName->addWidget(userName);
    lSystemDirectory->addWidget(nSystemDirectory);
    lSystemDirectory->addWidget(systemDirectory);
    lSystemWindowsDirectory->addWidget(nSystemWindowsDirectory);
    lSystemWindowsDirectory->addWidget(systemWindowsDirectory);
    lTempPath->addWidget(nTempPath);
    lTempPath->addWidget(tempPath);
    lVersion->addWidget(nVersion);
    lVersion->addWidget(version);

    layout->addLayout(lComputerName);
    layout->addLayout(lUserName);
    layout->addLayout(lSystemDirectory);
    layout->addLayout(lSystemWindowsDirectory);
    layout->addLayout(lTempPath);
    layout->addLayout(lVersion);

    this->setLayout(layout);
};
```

Содержимое файла systemInfo.hpp

```
#include <QtWidgets>
#include <windows.h>
#include <wchar.h>

#pragma once

QString QT_GetUserName();
QString QT_GetComputerName();
QString QT_GetSystemDirectory();
QString QT_GetSystemWindowsDirectory();
```

```
QString QT_GetTempPath();
QString QT_GetVersionExA();

class SystemInfo : public QWidget{
    Q_OBJECT
    private:
        QVBoxLayout *layout;
        QLabel *computerName,*userName,*systemDirectory,
            *systemWindowsDirectory,*tempPath,*version;
        QHBoxLayout *lComputerName,*lUserName,*lSystemDirectory,
            *lSystemWindowsDirectory,*lTempPath,*lVersion;
        QLabel *nComputerName,*nUserName,*nSystemDirectory,
            *nSystemWindowsDirectory,*nTempPath,*nVersion;
    public:
        SystemInfo();
};
```