**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ЛАБОРАТОРНАЯ РАБОТА №1**

**Дисциплина: Операционные системы**

**Тема: Функции Win32 API для получения системной информации**

Выполнил: ст. группы ВТ-31

Подкопаев Антон Валерьевич

Проверил: ст. пр. ПОВТиАС

Михелев Владимир Михайлович

**Белгород 2019**

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

В настоящее время Microsoft предлагает две 32-разрядные операционные си­стемы – Windows 9x (Windows 95, 98, ME) и Windows NT (Windows NT, 2000, XP). Интерфейс прикладного программирова­ния (Application Programming Interface Win32 - Win32 API) - это программный интерфейс, который используется для управления этими операционными сис­темами. Win32 API состоит из набора функций и подпрограмм, предоставляющих программный доступ к возможностям операцион­ной системы.

Программные интерфейсы приложений представляют собой набо­ры функций, которые обеспечивают сервисы данного приложения. Win32 API содержит более 3000 функции для реализации всех видов сервисов операционной системы.

API-функции Windows входят в состав динамически подключаемых библио­тек, поэтому необходимо знать определение DLL и понимать основ­ные принципы их работы.

Динамически подключаемая библиотека (Dynamic Link Library - DLL) явля­ется исполняемым файлом, который содержит несколько экспортируемых функции (exportable functions), то есть функций, к которым могут обращаться другие ис­полняемые приложения (ЕХЕ или DLL). Файлы DLL намного проще файлов ЕХЕ, например, в них нет кода, который управлял бы графическим интерфейсом или обрабатывал сообщения Windows.

Для размещения API-функций Windows использует несколько DLL. В дей­ствительности большая часть функций Win32 API содержится в трех DLL:

KERNEL32.DLL - содержит около 700 функций, которые предназначены для управления памятью, процессами и потоками;

USER32.DLL - предоставляет порядка 600 функций для управления пользо­вательским интерфейсом, например, созданием окон и передачей сообщений;

GDI.DLL - экспортирует около 400 функций для рисования графических образов, отображения текста и работы со шрифтами.

Кроме этих библиотек Windows также содержит несколько других DLL более узкой специализации. Здесь приводятся некоторые из них:

COMDLG32.DLL - открывает доступ почти к 20 функциям управления стан­дартными диалоговыми окнами Windows;

LZ32.DLL - хранит примерно 12 функций архивирования и разархивирования файлов;

ADVAPI32.DLL - экспортирует около 400 функций, связанных с защитой объектов и работой с реестром;

WINMM.DLL - содержит около 200 функций, относящихся к мультимедиа.

Основные Win32 API-функции получения систем­ной информации перечислены ниже:

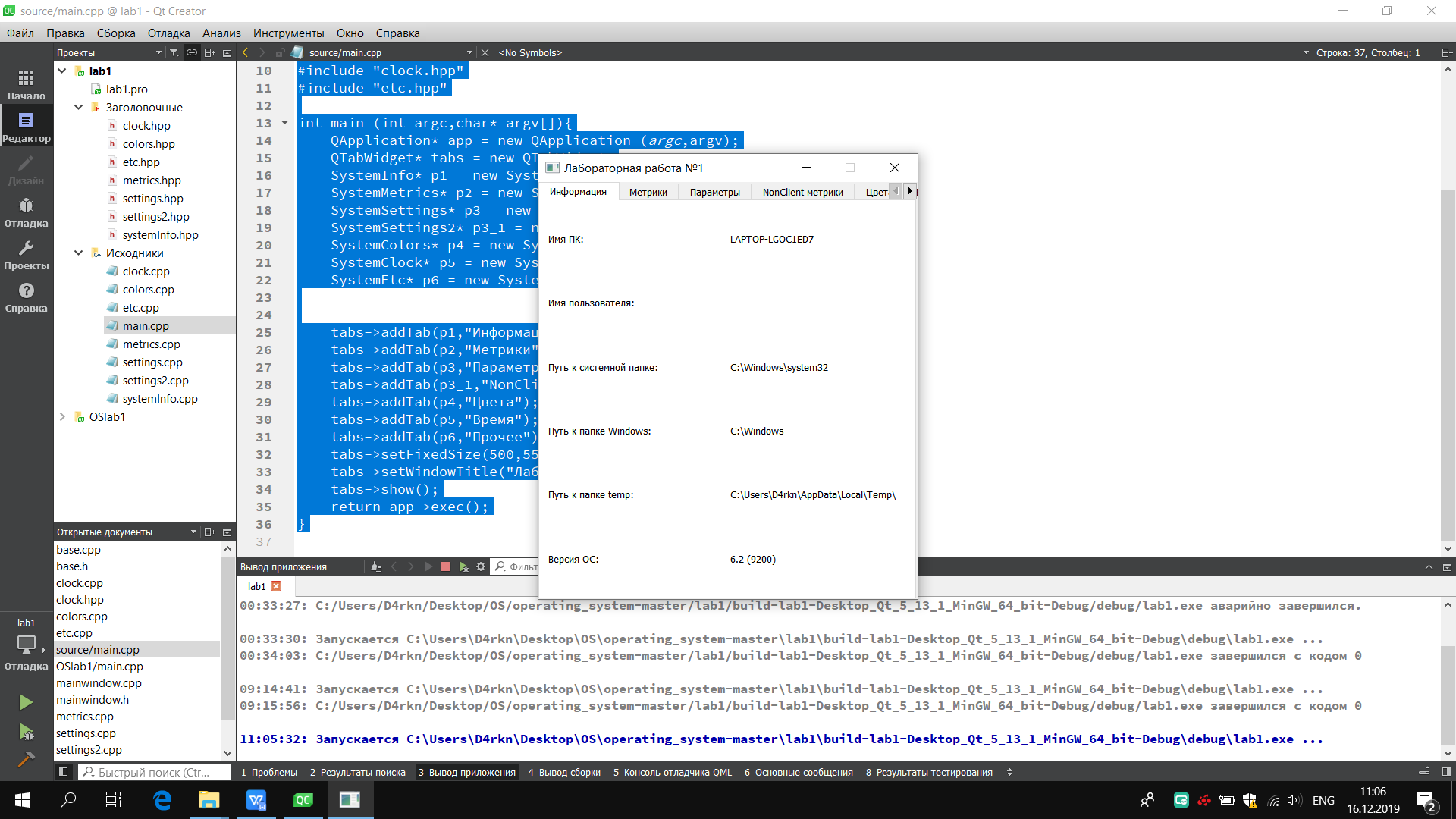
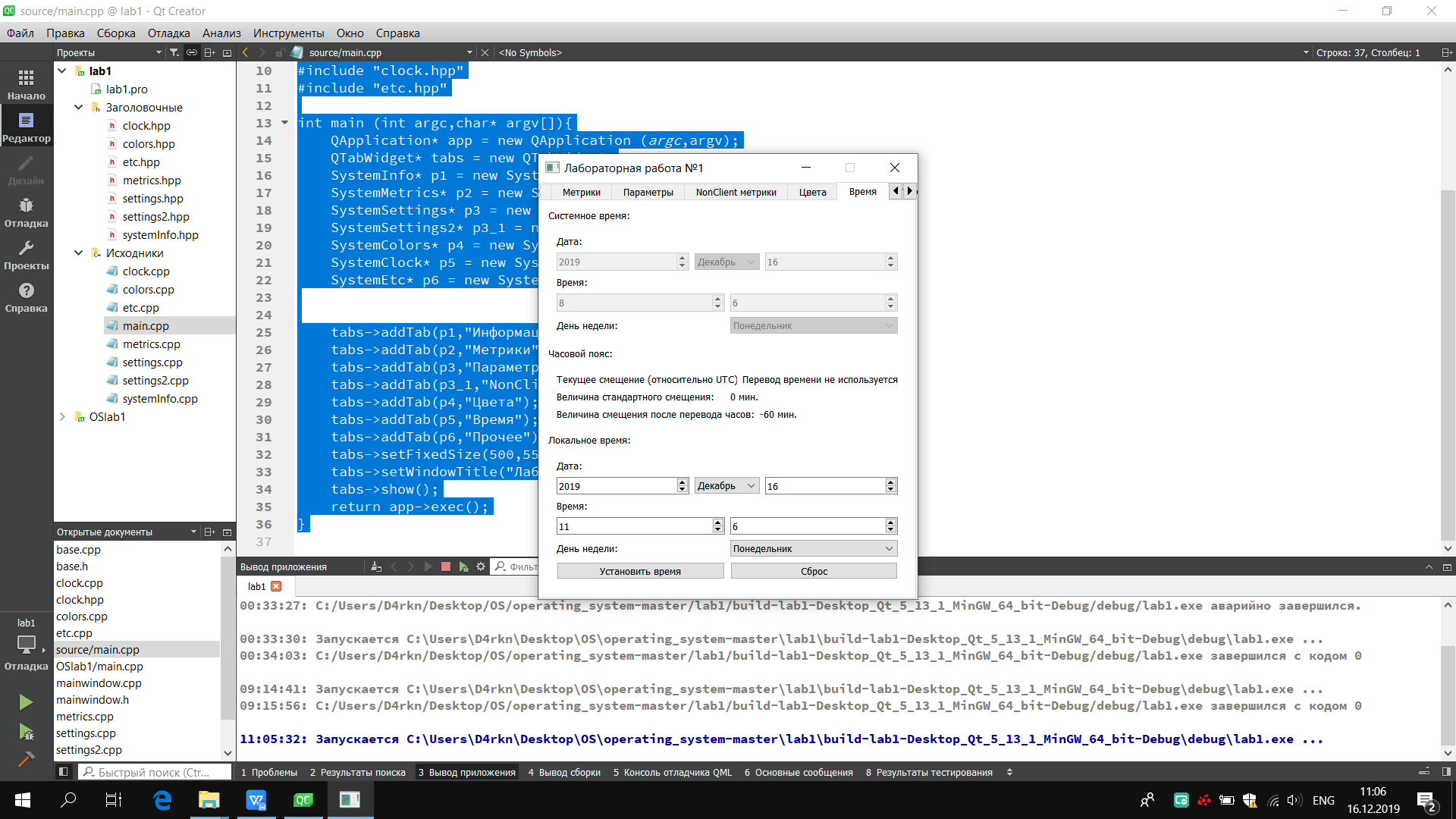
GetComputerName GetSystemMetrics GetWindowsDirectoty

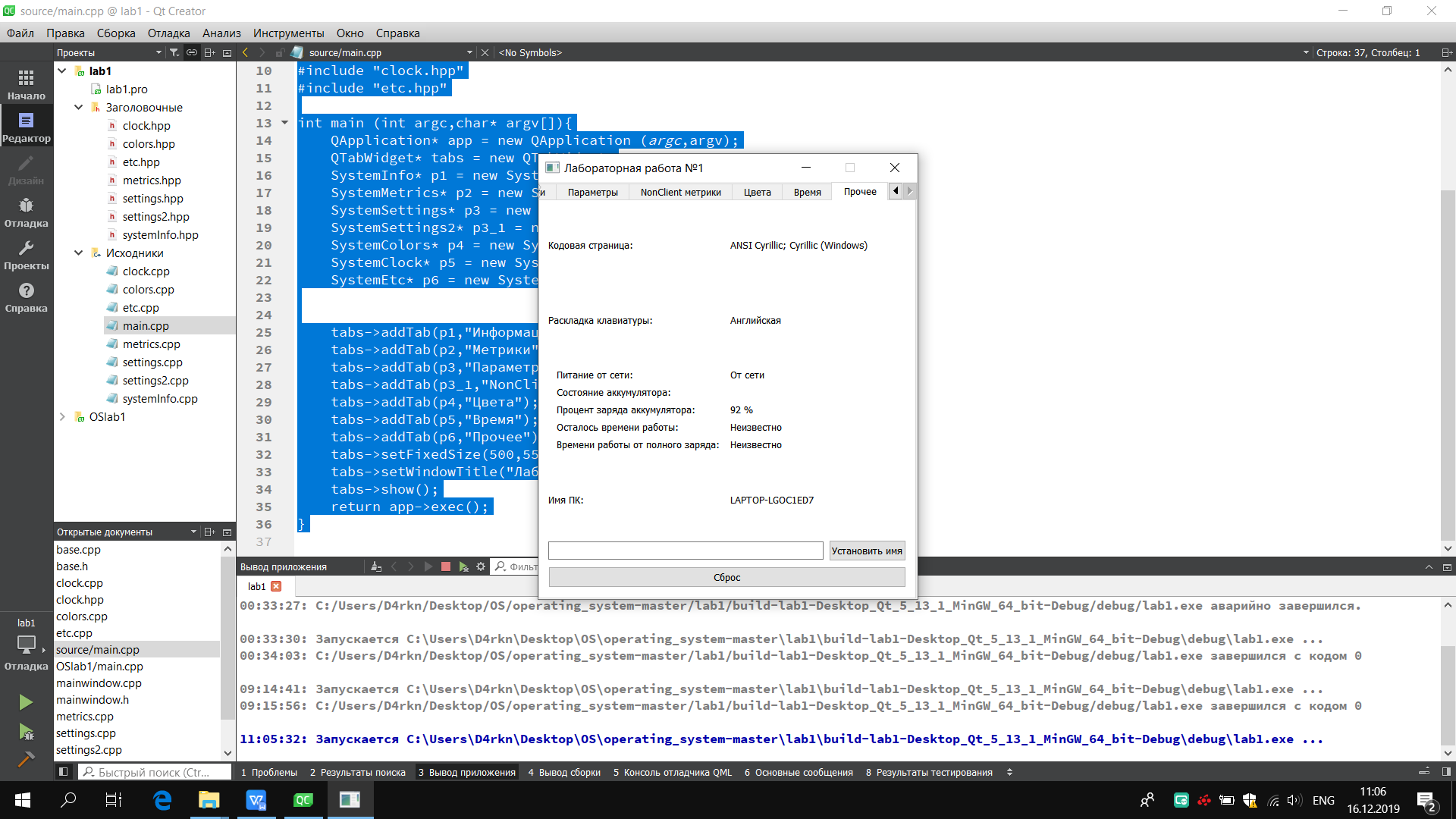
GetKeyboardTyре GetTempPath SetСоmрuterName

GetSysColor GetUserName SetSysColors

CecSystemDirectory GetVersion SystemParametersInfo

GetSystemlnfo GetVersioriEx GetUserName



#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(500,550);

tabs->setWindowTitle("Лабораторная работа №1");

tabs->show();

return app->exec();

}

#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);

};

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\_colorGreen,WIN\_colorBlue);

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()));

};

#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()));

};

#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);

};

#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()));

};

#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()));

};

#include "systemInfo.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 OC: ");

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);

};