Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №5

на тему

РЕЕСТР И ЖУРНАЛФЫ (WINDOWS). ДОСТУП К РЕЕСТРУ WINDOWS. РАБОТА С ЖУРНАЛАМИ WINDOWS. ДРУГИЕ ВСПОМОГАТЕЛЬНЫЕ СРЕДСТВА УПРАВЛЕНИЯ.

Студент: гр.153502 Матвеев Н.С.

Руководитель: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Формулировка задачи 3

2 Теоретические сведения 4

3 Описание функций программы 5

Список использованных источников 6

Приложение А (обязательное) Исходный код программы 7

**1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка утилиты для создания и управления реестровыми записями *Windows*, включая создание, изменение и удаление ключей и значений.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Реестр *Windows* (англ. *Windows* *Registry*), или системный реестр — иерархически построенная база данных параметров и настроек в большинстве операционных систем *Microsoft* *Windows*.

Реестр содержит информацию и настройки для аппаратного обеспечения, программного обеспечения, профилей пользователей, предустановки. большинство изменений в панели управления, ассоциации файлов, системные политики, список установленного ПО фиксируются в реестре.

Реестр *Windows* был введён для упорядочения информации, хранившейся до этого во множестве INI-файлов [1].

Для работы с реестром Windows были использованы следующие функции: *RegOpenKeyEx* ­– для открытия ключа в реестре, *RegQueryValueEx* – ­ для получения значения из реестра, *RegCloseKey­* ­­­­– для закрытия открытого ключа реестра. *RegCreateKeyEx* – для создания нового ключа реестра или открытия существующего, *RegDeleteKeyW* – для удаления ключа реестра, *RegSetValueExA* – для установки значения в реестр [2].

Значение реестра может хранить данные в одном из нескольких форматов, например, строковое (REG\_SZ ­– строка с нулевым символом в конце) или целочисленное значение (REG\_DWORD – 32-разрядное число, REG\_QWORD – 64-разрядное число). При хранении данных в значении реестра, например, путем вызова функции *RegSetValueEx*, можно указать тип хранимых данных, указав один из типов в таблице ниже. При получении значения реестра такие функции, как *RegQueryValueEx*, используют эти типы для указания типа полученных данных.

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

При запуске приложения в элемент *TreeView* загружаются ключи и значения реестра. В целях оптимизации ключи загружаются динамически в процессе использования программы.

Для управления добавлены кнопки удаления (Remove), изменения (Update) и добавления (Add) значения. В зависимости от типа выделенного элемента кнопки динамически выключаются и активируются.

При нажатии на кнопку добавления или изменения открывается окно DialogBox, запрашивающее пользовательский ввод Окна для добавления и изменения представлены на рисунках 1 и 2 соответственно.

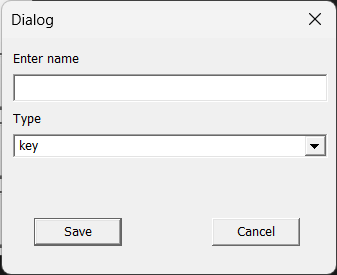


Рисунок 1 – Добавление элемента

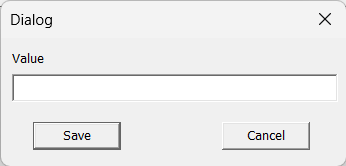


Рисунок 2 – Изменение элемента

Графический интерфейс программы представлен на рисунке 1.

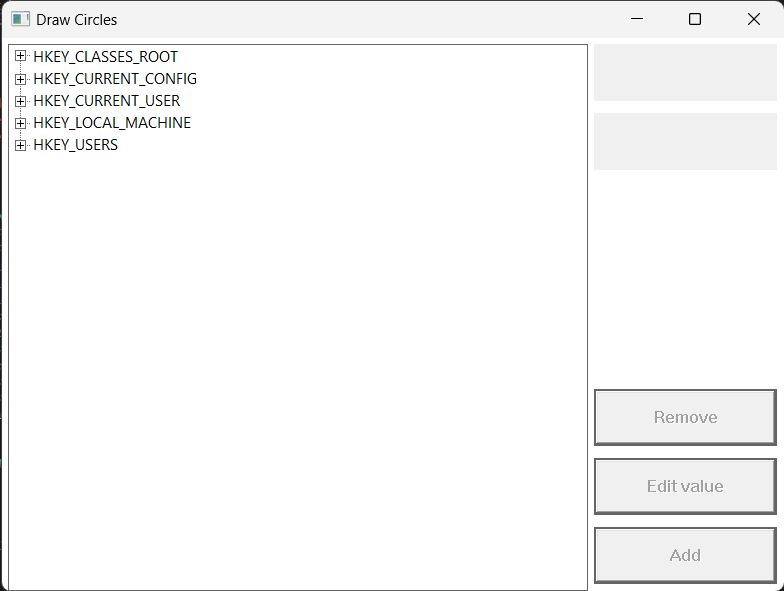


Рисунок 1 – Графический интерфейс программы

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Реестр Windows [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Реестр\_Windows

[2] Функции реестра – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry-functions

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

**Файл BaseWindow.h**

*#pragma once*

*template <class DERIVED\_TYPE>*

*class BaseWindow*

*{*

*public:*

*const PCWSTR CLASS\_NAME;*

*static LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*DERIVED\_TYPE \*pThis = NULL;*

*if (uMsg == WM\_NCCREATE)*

*{*

*CREATESTRUCT\* pCreate = (CREATESTRUCT\*)lParam;*

*pThis = (DERIVED\_TYPE\*)pCreate->lpCreateParams;*

*SetWindowLongPtr(hwnd, GWLP\_USERDATA, (LONG\_PTR)pThis);*

*pThis->m\_hwnd = hwnd;*

*}*

*else*

*{*

*pThis = (DERIVED\_TYPE\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);*

*}*

*if (pThis)*

*{*

*return pThis->HandleMessage(uMsg, wParam, lParam);*

*}*

*else*

*{*

*return DefWindowProc(hwnd, uMsg, wParam, lParam);*

*}*

*}*

*BaseWindow(PCWSTR CLASS\_NAME) : CLASS\_NAME(CLASS\_NAME), m\_hwnd(NULL) { }*

*BOOL Create(*

*PCWSTR lpWindowName,*

*DWORD dwStyle,*

*HWND hWndParent = 0,*

*DWORD dwExStyle = 0,*

*int x = CW\_USEDEFAULT,*

*int y = CW\_USEDEFAULT,*

*int nWidth = CW\_USEDEFAULT,*

*int nHeight = CW\_USEDEFAULT,*

*HMENU hMenu = 0*

*)*

*{*

*WNDCLASS wc = {};*

*wc.lpfnWndProc = DERIVED\_TYPE::WindowProc;*

*wc.hInstance = GetModuleHandle(NULL);*

*wc.lpszClassName = CLASS\_NAME;*

*RegisterClass(&wc);*

*m\_hwnd = CreateWindowEx(*

*dwExStyle, CLASS\_NAME, lpWindowName, dwStyle, x, y,*

*nWidth, nHeight, hWndParent, hMenu, GetModuleHandle(NULL), this*

*);*

*return (m\_hwnd ? TRUE : FALSE);*

*}*

*HWND Window() const { return m\_hwnd; }*

*protected:*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;*

*HWND m\_hwnd;*

*};*

**Файл DPIScale.h**

*#pragma once*

*#include <d2d1.h>*

*class DPIScale*

*{*

*static float scaleX;*

*static float scaleY;*

*public:*

*static void Initialize()*

*{*

*FLOAT dpi = GetDpiForSystem();*

*scaleX = dpi / 96.0f;*

*scaleY = dpi / 96.0f;*

*}*

*template <typename T>*

*static float PixelsToDipsX(T x)*

*{*

*return static\_cast<float>(x) / scaleX;*

*}*

*template <typename T>*

*static float PixelsToDipsY(T y)*

*{*

*return static\_cast<float>(y) / scaleY;*

*}*

*template <typename T>*

*static T DipXToPixels(float x)*

*{*

*return static\_cast<T>(x) \* scaleX;*

*}*

*template <typename T>*

*static T DipYToPixels(float y)*

*{*

*return static\_cast<T>(y) \* scaleY;*

*}*

*};*

**Файл DPIScale.cpp**

*#include "DPIScale.h"*

*float DPIScale::scaleX = 1;*

*float DPIScale::scaleY = 1;*

**Файл main.cpp**

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <Winuser.h>*

*#include <list>*

*#include <memory>*

*using namespace std;*

*#pragma comment(lib, "d2d1")*

*#pragma comment(lib, "Ntdll")*

*#pragma comment(lib, "Comctl32")*

*#include "BaseWindow.h"*

*#include "MainWindow.h"*

*#include <objidl.h>*

*#include <gdiplus.h>*

*#include <winternl.h>*

*#include <CommCtrl.h>*

*int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR, int nCmdShow)*

*{*

*//Gdiplus::GdiplusStartupInput gdiplusStartupInput;*

*//ULONG\_PTR gdiplusToken;*

*//Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);*

*MainWindow win = MainWindow();*

*//Mode mode = Mode::SelectMode;*

*//Figure figure = Figure::Ellipse;*

*//D2D1\_COLOR\_F color = D2D1::ColorF(D2D1::ColorF::Black);*

*//SceneControl win = SceneControl(&mode, &figure, &color);*

*//GraphicsScene win = GraphicsScene(&mode, &figure, &color);*

*if (!win.Create(L"Draw Circles",*

*WS\_OVERLAPPEDWINDOW,*

*NULL,*

*NULL,*

*CW\_USEDEFAULT,*

*CW\_USEDEFAULT,*

*800,*

*600))*

*{*

*return 0;*

*}*

*//SetWindowsHookEx(WH\_GETMESSAGE, GetMsgProcCustom, NULL, GetCurrentThreadId());*

*//HACCEL hAccel1 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL1));*

*//HACCEL hAccel2 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL2));*

*ShowWindow(win.Window(), nCmdShow);*

*MSG msg;*

*while (GetMessage(&msg, NULL, 0, 0))*

*{*

*//if (!TranslateAccelerator(GetFocus(), hAccel1, &msg) && !TranslateAccelerator(GetFocus(), hAccel2, &msg))*

*//{*

*TranslateMessage(&msg);*

*DispatchMessage(&msg);*

*//}*

*}*

*return 0;*

*}*

**Файл MainWindow.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <compressapi.h>*

*#include <shlwapi.h>*

*#include <string>*

*#include <map>*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*#include "BaseNode.h"*

*class MainWindow : public BaseWindow<MainWindow>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*static const float MARGIN\_X;*

*static const float MARGIN\_Y;*

*static const int MAX\_NAME\_SIZE;*

*static const int MAX\_VALUE\_SIZE;*

*static const int MAX\_TYPE\_SIZE;*

*static const std::map<std::wstring, DWORD> types;*

*static LPCWSTR DEFAULT\_TYPE;*

*static std::wstring currentName;*

*static DWORD currentDwordValue;*

*static std::wstring currentSzValue;*

*static std::wstring currentType;*

*static bool succeded;*

*public:*

*MainWindow(PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME);*

*~MainWindow();*

*void update();*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*void CreateLayout();*

*void SetLayout();*

*static BOOL CALLBACK addDialogProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam);*

*static BOOL CALLBACK changeValueSzDialogProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam);*

*static BOOL CALLBACK changeValueDwordDialogProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam);*

*HWND addButton;*

*HWND removeButton;*

*HWND updateButton;*

*HWND typeLabel;*

*HWND valueLabel;*

*BaseNode\* selected;*

*HWND tree;*

*};*

**Файл MainWindow.cpp**

*#include "MainWindow.h"*

*#include "KeyNode.h"*

*#include "ValueNode.h"*

*#include "resource.h"*

*#include <stdexcept>*

*const PCWSTR MainWindow::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const float MainWindow::MARGIN\_X = 6.0F;*

*const float MainWindow::MARGIN\_Y = 6.0F;*

*const int MainWindow::MAX\_NAME\_SIZE = 256;*

*const int MainWindow::MAX\_TYPE\_SIZE = 16;*

*const int MainWindow::MAX\_VALUE\_SIZE = 16384;*

*LPCWSTR MainWindow::DEFAULT\_TYPE = L"key";*

*std::wstring MainWindow::currentName = std::wstring();*

*DWORD MainWindow::currentDwordValue = NULL;*

*std::wstring MainWindow::currentSzValue = std::wstring();*

*std::wstring MainWindow::currentType = std::wstring();*

*bool MainWindow::succeded = false;*

*const std::map<std::wstring, DWORD> MainWindow::types = {*

*{ L"REG\_SZ", REG\_SZ },*

*{ L"REG\_DWORD", REG\_DWORD }*

*};*

*MainWindow::MainWindow(PCWSTR CLASS\_NAME) :*

*BaseWindow<MainWindow>(CLASS\_NAME), selected(NULL)*

*{*

*}*

*MainWindow::~MainWindow()*

*{*

*}*

*void MainWindow::update()*

*{*

*if (!selected)*

*{*

*SetWindowText(typeLabel, NULL);*

*SetWindowText(valueLabel, NULL);*

*EnableWindow(removeButton, FALSE);*

*EnableWindow(updateButton, FALSE);*

*EnableWindow(addButton, FALSE);*

*return;*

*}*

*ValueNode\* value = dynamic\_cast<ValueNode\*>(selected);*

*if (value)*

*{*

*EnableWindow(removeButton, TRUE);*

*EnableWindow(addButton, FALSE);*

*DWORD type;*

*DWORD valueMaxSize = 16384;*

*BYTE\* val = new BYTE[valueMaxSize];*

*value->get(&type, val, &valueMaxSize);*

*switch (type)*

*{*

*case REG\_DWORD:*

*{*

*SetWindowText(typeLabel, L"Type: REG\_DWORD");*

*std::wstring tmp = std::to\_wstring(\*(DWORD\*)val);*

*SetWindowText(valueLabel, tmp.c\_str());*

*EnableWindow(updateButton, TRUE);*

*break;*

*}*

*case REG\_SZ:*

*SetWindowText(typeLabel, L"Type: REG\_SZ");*

*SetWindowText(valueLabel, (LPWSTR)val);*

*EnableWindow(updateButton, TRUE);*

*break;*

*default:*

*SetWindowText(typeLabel, L"Type: unknown");*

*SetWindowText(valueLabel, NULL);*

*EnableWindow(updateButton, FALSE);*

*break;*

*}*

*delete[] val;*

*}*

*else*

*{*

*SetWindowText(typeLabel, L"Type: key");*

*SetWindowText(valueLabel, NULL);*

*EnableWindow(removeButton, TRUE);*

*EnableWindow(updateButton, FALSE);*

*EnableWindow(addButton, TRUE);*

*}*

*}*

*//static HBRUSH hBrush = CreateSolidBrush(RGB(230, 2, 2));*

*LRESULT MainWindow::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*CreateLayout();*

*return 0;*

*case WM\_PAINT:*

*{*

*PAINTSTRUCT ps;*

*HDC hdc = BeginPaint(m\_hwnd, &ps);*

*FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));*

*EndPaint(m\_hwnd, &ps);*

*return 0;*

*}*

*case WM\_INITDIALOG:*

*InitCommonControls();*

*case WM\_SIZE:*

*SetLayout();*

*return 0;*

*/\*case WM\_SIZING:*

*{*

*RECT\* dragRc = (RECT\*)lParam;*

*RECT rcWindow;*

*GetWindowRect(m\_hwnd, &rcWindow);*

*int WINDOW\_HEIGHT\_PIX = rcWindow.bottom - rcWindow.top;*

*int WINDOW\_WIDTH\_PIX = rcWindow.right - rcWindow.left;*

*switch (wParam)*

*{*

*case WMSZ\_BOTTOM:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_BOTTOMLEFT:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_LEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_TOPLEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOP:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOPRIGHT:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_RIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_BOTTOMRIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*}*

*return TRUE;*

*}\*/*

*case WM\_NOTIFY:*

*{*

*NMHDR\* notification = (NMHDR\*)lParam;*

*switch (((NMHDR\*)lParam)->code)*

*{*

*case TVN\_SELCHANGED:*

*selected = (BaseNode\*)((LPNMTREEVIEW)lParam)->itemNew.lParam;*

*update();*

*return 0;*

*case TVN\_ITEMEXPANDED:*

*{*

*NMTREEVIEW\* item = (NMTREEVIEW\*)(lParam);*

*KeyNode\* key = (KeyNode\*)(item)->itemNew.lParam;*

*if (KeyNode::PREDEFINED\_NAMES.find(key->getName()) == KeyNode::PREDEFINED\_NAMES.end())*

*key->initialize(1);*

*break;*

*}*

*}*

*break;*

*}*

*case WM\_COMMAND:*

*if (HIWORD(wParam) == BN\_CLICKED)*

*{*

*if ((HWND)lParam == addButton)*

*{*

*currentName.clear();*

*currentType = DEFAULT\_TYPE;*

*DialogBoxW(NULL, MAKEINTRESOURCEW(IDD\_ADD\_DIALOG), m\_hwnd, (DLGPROC)addDialogProc);*

*if (succeded)*

*{*

*if (currentType == DEFAULT\_TYPE)*

*{*

*try*

*{*

*KeyNode\* key = new KeyNode(tree, currentName.c\_str(), (KeyNode\*)selected);*

*}*

*catch (std::invalid\_argument)*

*{*

*}*

*}*

*else*

*{*

*try*

*{*

*ValueNode\* value = new ValueNode(tree, currentName.c\_str(), (KeyNode\*)selected);*

*value->set(types.at(currentType), NULL, NULL);*

*}*

*catch (std::invalid\_argument)*

*{*

*}*

*}*

*((KeyNode\*)selected)->initialize(1);*

*}*

*update();*

*}*

*else if ((HWND)lParam == removeButton)*

*{*

*selected->remove();*

*delete selected;*

*selected = nullptr;*

*update();*

*}*

*else if ((HWND)lParam == updateButton)*

*{*

*DWORD type;*

*((ValueNode\*)selected)->get(&type, NULL, NULL);*

*if (type == REG\_SZ)*

*{*

*DialogBoxW(NULL, MAKEINTRESOURCEW(IDD\_CHANGE\_SZ\_DIALOG), m\_hwnd, (DLGPROC)changeValueSzDialogProc);*

*if (succeded)*

*{*

*((ValueNode\*)selected)->set(type, (BYTE\*)currentSzValue.c\_str(), currentSzValue.size() + 1);*

*}*

*}*

*else if (type == REG\_DWORD)*

*{*

*DialogBoxW(NULL, MAKEINTRESOURCEW(IDD\_CHANGE\_DWORD\_DIALOG), m\_hwnd, (DLGPROC)changeValueDwordDialogProc);*

*if (succeded)*

*{*

*((ValueNode\*)selected)->set(type, (BYTE\*)&currentDwordValue, sizeof(DWORD));*

*}*

*}*

*update();*

*}*

*return 0;*

*}*

*break;*

*/\*case WM\_CTLCOLORSTATIC:*

*{*

*HDC hdcStatic = (HDC)wParam;*

*SetTextColor(hdcStatic, RGB(0, 0, 0));*

*SetBkColor(hdcStatic, RGB(230, 2, 2));*

*return (INT\_PTR)hBrush;*

*}\*/*

*}*

*return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);*

*}*

*void MainWindow::CreateLayout()*

*{*

*tree = CreateWindow(WC\_TREEVIEW,*

*L"Tree view",*

*WS\_VISIBLE | WS\_CHILD | WS\_BORDER | TVS\_HASLINES | TVS\_HASBUTTONS | TVS\_LINESATROOT | TVS\_SHOWSELALWAYS,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*addButton = CreateWindow(L"BUTTON",*

*L"Add",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*removeButton = CreateWindow(L"BUTTON",*

*L"Remove",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*updateButton = CreateWindow(L"BUTTON",*

*L"Edit value",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*typeLabel = CreateWindow(L"STATIC",*

*NULL,*

*WS\_VISIBLE | WS\_CHILD | SS\_LEFT,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*valueLabel = CreateWindow(L"STATIC",*

*NULL,*

*WS\_VISIBLE | WS\_CHILD | SS\_LEFT,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*for (auto& i : KeyNode::PREDEFINED\_NAMES)*

*{*

*KeyNode\* key = new KeyNode(tree, i.first.c\_str(), NULL);*

*key->initialize(1);*

*}*

*update();*

*}*

*void MainWindow::SetLayout()*

*{*

*RECT rcClient;*

*GetClientRect(m\_hwnd, &rcClient);*

*int MARGIN\_XPix = DPIScale::DipXToPixels<int>(MARGIN\_X);*

*int MARGIN\_YPix = DPIScale::DipYToPixels<int>(MARGIN\_Y);*

*int WINDOW\_HEIGHT\_PIX = rcClient.bottom - rcClient.top;*

*int WINDOW\_WIDTH\_PIX = rcClient.right - rcClient.left;*

*MoveWindow(tree,*

*MARGIN\_XPix,*

*MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 - MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX - MARGIN\_YPix,*

*FALSE);*

*MoveWindow(typeLabel,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(valueLabel,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(addButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 1) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(updateButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 2) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*MoveWindow(removeButton,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 + MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* (8 - 3) / 8 + MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX / 4 - MARGIN\_XPix \* 2,*

*WINDOW\_HEIGHT\_PIX / 8 - MARGIN\_YPix \* 2,*

*FALSE);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*BOOL CALLBACK MainWindow::addDialogProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam)*

*{*

*switch (message)*

*{*

*case WM\_INITDIALOG:*

*{*

*SetDlgItemTextW(hwndDlg, IDC\_ITEM\_NAME, currentName.c\_str());*

*HWND comboBox = GetDlgItem(hwndDlg, IDC\_TYPE);*

*for (auto& i : types)*

*ComboBox\_AddString(comboBox, i.first.c\_str());*

*ComboBox\_AddString(comboBox, DEFAULT\_TYPE);*

*SetDlgItemTextW(hwndDlg, IDC\_TYPE, currentType.c\_str());*

*}*

*case WM\_COMMAND:*

*switch (LOWORD(wParam))*

*{*

*case IDOK:*

*{*

*LPWSTR name = new WCHAR[MAX\_NAME\_SIZE];*

*if (!GetDlgItemTextW(hwndDlg, IDC\_ITEM\_NAME, name, MAX\_NAME\_SIZE))*

*\*name = 0;*

*currentName = name;*

*delete[] name;*

*LPWSTR type = new WCHAR[MAX\_TYPE\_SIZE];*

*if (!GetDlgItemTextW(hwndDlg, IDC\_TYPE, type, MAX\_TYPE\_SIZE))*

*\*type = 0;*

*currentType = type;*

*delete[] type;*

*succeded = true;*

*EndDialog(hwndDlg, wParam);*

*return TRUE;*

*}*

*case IDCANCEL:*

*{*

*succeded = false;*

*EndDialog(hwndDlg, wParam);*

*return TRUE;*

*}*

*}*

*}*

*return FALSE;*

*}*

*BOOL CALLBACK MainWindow::changeValueSzDialogProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam)*

*{*

*switch (message)*

*{*

*case WM\_INITDIALOG:*

*SetDlgItemTextW(hwndDlg, IDC\_VALUE\_SZ, currentSzValue.c\_str());*

*case WM\_COMMAND:*

*switch (LOWORD(wParam))*

*{*

*case IDOK:*

*{*

*LPWSTR value = new WCHAR[MAX\_VALUE\_SIZE];*

*if (!GetDlgItemTextW(hwndDlg, IDC\_VALUE\_SZ, value, MAX\_VALUE\_SIZE))*

*\*value = 0;*

*currentSzValue = value;*

*delete[] value;*

*succeded = true;*

*EndDialog(hwndDlg, wParam);*

*return TRUE;*

*}*

*case IDCANCEL:*

*{*

*succeded = false;*

*EndDialog(hwndDlg, wParam);*

*return TRUE;*

*}*

*}*

*}*

*return FALSE;*

*}*

*BOOL CALLBACK MainWindow::changeValueDwordDialogProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam)*

*{*

*switch (message)*

*{*

*case WM\_INITDIALOG:*

*SetDlgItemInt(hwndDlg, IDC\_VALUE\_DWORD, currentDwordValue, TRUE);*

*case WM\_COMMAND:*

*switch (LOWORD(wParam))*

*{*

*case IDOK:*

*{*

*BOOL success;*

*currentDwordValue = GetDlgItemInt(hwndDlg, IDC\_VALUE\_DWORD, &success, TRUE);*

*if (!success)*

*currentDwordValue = 0;*

*succeded = true;*

*EndDialog(hwndDlg, wParam);*

*return TRUE;*

*}*

*case IDCANCEL:*

*{*

*succeded = false;*

*EndDialog(hwndDlg, wParam);*

*return TRUE;*

*}*

*}*

*}*

*return FALSE;*

*}*

**Файл BaseNode.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <Winuser.h>*

*#include <list>*

*#include <memory>*

*#include <objidl.h>*

*#include <gdiplus.h>*

*#include <winternl.h>*

*#include <CommCtrl.h>*

*#include <string>*

*class BaseNode*

*{*

*public:*

*BaseNode(HWND tree, LPCWSTR name) : tree(tree), name(name), node() {}*

*virtual ~BaseNode() = default;*

*const TVINSERTSTRUCTW& getNode() const { return node; }*

*const std::wstring& getName() const { return name; }*

*virtual void remove() = 0;*

*protected:*

*HWND tree;*

*std::wstring name;*

*TVINSERTSTRUCTW node;*

*};*

**Файл KeyNode.h**

*#pragma once*

*#include "BaseNode.h"*

*#include <set>*

*#include <map>*

*class ValueNode;*

*class KeyNode : public BaseNode*

*{*

*public:*

*static const std::map<std::wstring, HKEY> PREDEFINED\_NAMES;*

*static const size\_t MAX\_NAME\_SIZE;*

*KeyNode(HWND tree, LPCWSTR name, KeyNode\* parent);*

*virtual ~KeyNode() override;*

*void initialize(int depth);*

*virtual void remove() override;*

*protected:*

*void open(PHKEY key);*

*KeyNode\* parent;*

*std::set<BaseNode\*> children;*

*friend class ValueNode;*

*};*

**Файл KeyNode.cpp**

*#include "KeyNode.h"*

*#include "ValueNode.h"*

*#include <stdexcept>*

*const std::map<std::wstring, HKEY> KeyNode::PREDEFINED\_NAMES = {*

*{ L"HKEY\_CLASSES\_ROOT", HKEY\_CLASSES\_ROOT },*

*{ L"HKEY\_CURRENT\_CONFIG", HKEY\_CURRENT\_CONFIG },*

*{ L"HKEY\_CURRENT\_USER", HKEY\_CURRENT\_USER },*

*{ L"HKEY\_LOCAL\_MACHINE", HKEY\_LOCAL\_MACHINE },*

*{ L"HKEY\_USERS", HKEY\_USERS }*

*};*

*const size\_t KeyNode::MAX\_NAME\_SIZE = 256;*

*KeyNode::KeyNode(HWND tree, LPCWSTR name, KeyNode\* parent) : BaseNode(tree, name), parent(parent), children()*

*{*

*if (parent != NULL)*

*{*

*for (auto& ptr : parent->children)*

*{*

*if (ptr->getName() == this->name)*

*{*

*throw std::invalid\_argument("File with such name already exists");*

*}*

*}*

*parent->children.insert(this);*

*node.hParent = parent->getNode().item.hItem;*

*}*

*node.hInsertAfter = TVI\_SORT;*

*node.item.mask = TVIF\_PARAM | TVIF\_TEXT;*

*node.item.lParam = (LPARAM)this;*

*node.item.pszText = (LPWSTR)this->name.c\_str();*

*node.item.cchTextMax = this->name.size() + 1;*

*node.item.hItem = (HTREEITEM)SendMessage(tree, TVM\_INSERTITEM, 0, (LPARAM)&node);*

*HKEY key;*

*open(&key);*

*LPWSTR childName = new WCHAR[MAX\_NAME\_SIZE];*

*DWORD realSize;*

*realSize = MAX\_NAME\_SIZE;*

*if (RegEnumKeyExW(key, NULL, childName, &realSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS)*

*new KeyNode(tree, childName, this);*

*realSize = MAX\_NAME\_SIZE;*

*if (RegEnumValueW(key, NULL, childName, &realSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS)*

*new ValueNode(tree, childName, this);*

*delete[] childName;*

*RegCloseKey(key);*

*}*

*KeyNode::~KeyNode()*

*{*

*while (children.size())*

*delete \*children.begin();*

*if (parent != NULL)*

*parent->children.erase(this);*

*SendMessage(tree, TVM\_DELETEITEM, 0, (LPARAM)node.item.hItem);*

*}*

*void KeyNode::initialize(int depth)*

*{*

*if (depth == 0)*

*{*

*if (children.size())*

*{*

*while (children.size() != 1)*

*delete\* children.begin();*

*}*

*return;*

*}*

*HKEY key;*

*open(&key);*

*LPWSTR childName = new WCHAR[MAX\_NAME\_SIZE];*

*DWORD realSize = MAX\_NAME\_SIZE;*

*LSTATUS some, some1;*

*for (auto& ptr : children)*

*{*

*KeyNode\* childKey = dynamic\_cast<KeyNode\*>(ptr);*

*if (childKey != NULL)*

*childKey->initialize(depth - 1);*

*}*

*for (DWORD i = 0; (some = RegEnumKeyExW(key, i, childName, &realSize, NULL, NULL, NULL, NULL)) == ERROR\_SUCCESS; ++i)*

*{*

*realSize = MAX\_NAME\_SIZE;*

*bool exists = false;*

*for (auto& ptr : children)*

*{*

*if (ptr->getName() == childName)*

*{*

*exists = true;*

*break;*

*}*

*}*

*if (!exists)*

*{*

*KeyNode\* childKey;*

*childKey = new KeyNode(tree, childName, this);*

*childKey->initialize(depth - 1);*

*}*

*}*

*some1 = some;*

*for (DWORD i = 0; (some = RegEnumValueW(key, i, childName, &realSize, NULL, NULL, NULL, NULL)) == ERROR\_SUCCESS; ++i)*

*{*

*realSize = MAX\_NAME\_SIZE;*

*bool exists = false;*

*for (auto& ptr : children)*

*{*

*if (ptr->getName() == childName)*

*{*

*exists = true;*

*break;*

*}*

*}*

*if (!exists)*

*{*

*new ValueNode(tree, childName, this);*

*}*

*}*

*some1 = some;*

*delete[] childName;*

*RegCloseKey(key);*

*}*

*void KeyNode::remove()*

*{*

*if (parent != NULL)*

*{*

*HKEY key;*

*parent->open(&key);*

*RegDeleteTreeW(key, name.c\_str());*

*RegDeleteKeyW(key, name.c\_str());*

*RegCloseKey(key);*

*}*

*}*

*void KeyNode::open(PHKEY key)*

*{*

*auto& it1 = PREDEFINED\_NAMES.find(name);*

*auto& it2 = PREDEFINED\_NAMES.end();*

*if (PREDEFINED\_NAMES.find(name) != PREDEFINED\_NAMES.end())*

*{*

*\*key = PREDEFINED\_NAMES.at(name);*

*}*

*else*

*{*

*HKEY pKey;*

*parent->open(&pKey);*

*LSTATUS some;*

*if ((some = RegCreateKeyExW(pKey, name.c\_str(), NULL, NULL, NULL, KEY\_ALL\_ACCESS, NULL, key, NULL)) == ERROR\_ACCESS\_DENIED)*

*some = RegCreateKeyExW(pKey, name.c\_str(), NULL, NULL, NULL, KEY\_READ, NULL, key, NULL) == ERROR\_ACCESS\_DENIED;*

*LSTATUS some1 = some;*

*RegCloseKey(pKey);*

*}*

*}*

**Файл ValueNode.h**

*#pragma once*

*#include "BaseNode.h"*

*class KeyNode;*

*class ValueNode : public BaseNode*

*{*

*static const DWORD DEFAULT\_TYPE;*

*static const LPCWSTR DEFAULT\_NAME;*

*static const BYTE \*DEFAULT\_VALUE;*

*public:*

*ValueNode(HWND tree, LPCWSTR name, KeyNode\* parent);*

*virtual ~ValueNode() override;*

*void set(DWORD type, const BYTE\* data, DWORD dataSize);*

*void get(DWORD\* type, BYTE\* data, DWORD\* dataSize);*

*virtual void remove() override;*

*protected:*

*KeyNode\* parent;*

*friend class KeyNode;*

*};*

**Файл ValueNode.cpp**

*#include "ValueNode.h"*

*#include "KeyNode.h"*

*#include <stdexcept>*

*const DWORD ValueNode::DEFAULT\_TYPE = REG\_SZ;*

*const LPCWSTR ValueNode::DEFAULT\_NAME = L"DEFAULT";*

*const BYTE \*ValueNode::DEFAULT\_VALUE = (BYTE\*)L"DEFAULT";*

*ValueNode::ValueNode(HWND tree, LPCWSTR name, KeyNode\* parent) : BaseNode(tree, name), parent(parent)*

*{*

*for (auto& ptr : parent->children)*

*{*

*if (ptr->getName() == this->name)*

*{*

*throw std::invalid\_argument("File with such name already exists");*

*}*

*}*

*parent->children.insert(this);*

*node.hParent = parent->getNode().item.hItem;*

*node.hInsertAfter = TVI\_SORT;*

*node.item.mask = TVIF\_PARAM | TVIF\_TEXT;*

*node.item.lParam = (LPARAM)this;*

*if (this->name.size())*

*node.item.pszText = (LPWSTR)this->name.c\_str();*

*else*

*node.item.pszText = (LPWSTR)DEFAULT\_NAME;*

*node.item.cchTextMax = this->name.size() + 1;*

*node.item.hItem = (HTREEITEM)SendMessage(tree, TVM\_INSERTITEM, 0, (LPARAM)&node);*

*HKEY key;*

*parent->open(&key);*

*if (RegGetValueW(key, NULL, this->name.c\_str(), NULL, NULL, NULL, NULL) == ERROR\_FILE\_NOT\_FOUND)*

*RegSetValueExW(key, this->name.c\_str(), NULL, DEFAULT\_TYPE, DEFAULT\_VALUE, sizeof(DEFAULT\_VALUE));*

*RegCloseKey(key);*

*}*

*ValueNode::~ValueNode()*

*{*

*parent->children.erase(this);*

*SendMessage(tree, TVM\_DELETEITEM, 0, (LPARAM)node.item.hItem);*

*}*

*void ValueNode::set(DWORD type, const BYTE\* data, DWORD dataSize)*

*{*

*HKEY key;*

*parent->open(&key);*

*RegSetValueExW(key, this->name.c\_str(), NULL, type, data, dataSize);*

*RegCloseKey(key);*

*}*

*void ValueNode::get(DWORD\* type, BYTE\* data, DWORD\* dataSize)*

*{*

*HKEY key;*

*parent->open(&key);*

*LSTATUS some = RegGetValueW(key, NULL, this->name.c\_str(), RRF\_RT\_ANY, type, data, dataSize);*

*LSTATUS some1 = some;*

*RegCloseKey(key);*

*}*

*void ValueNode::remove()*

*{*

*HKEY key;*

*parent->open(&key);*

*RegDeleteValueW(key, this->name.c\_str());*

*RegCloseKey(key);*

*}*