Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Системное программирование (СП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**ПРОГРАММНОЕ СРЕДСТВО**

**«Монитор обращений к USB»**

БГУИР КП 1-40 01 01 007 ПЗ

Студент Карась А.С.

Руководитель Алексеев И.Г.

Минск 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc183800048)

[1 АНАЛИЗ ПРЕДМЕНТНОЙ ОБЛАСТИ 6](#_Toc183800049)

[1.1 Обзор аналогов 6](#_Toc183800050)

[1.2 Постановка задачи 8](#_Toc183800051)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9](#_Toc183800052)

[2.1 Структура программы 9](#_Toc183800053)

[2.2 Проектирование интерфейса программного средства 9](#_Toc183800054)

[2.3 Проектирование функционала программного средства 11](#_Toc183800055)

[3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 16](#_Toc183800056)

[3.1 Создание графических ресурсов для главного окна приложения 16](#_Toc183800057)

[3.2 Сортировка строк в сводной таблице 17](#_Toc183800058)

[3.3 Отслеживание изменений в реестре Windows и подключений USB-устройств 19](#_Toc183800059)

[4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 21](#_Toc183800060)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 22](#_Toc183800061)

[5.1 Интерфейс программного средства 22](#_Toc183800062)

[5.2 Управление программным средством 23](#_Toc183800063)

[ЗАКЛЮЧЕНИЕ 24](#_Toc183800064)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc183800065)

[ПРИЛОЖЕНИЕ А 26](#_Toc183800066)

**ВВЕДЕНИЕ**

USB (Universal Serial Bus, универсальная последовательная шина) — это стандарт для подключения периферийных устройств к компьютерам, который стал фактически всеобщим и является одним из самых популярных интерфейсов в современном мире.

В конце 80-х годов XX века компьютеры использовали разнообразные интерфейсы для подключения периферии: Serial (RS-232), Parallel, PS/2, SCSI и другие. Это приводило к неудобствам для пользователей, так как каждое устройство требовало отдельного порта и специфического кабеля.

В 1994 году компания Intel в сотрудничестве с компаниями Microsoft, Compaq, IBM и NEC начала разработку нового стандарта для подключения периферии. Цель состояла в том, чтобы создать более простой и универсальный интерфейс, который мог бы использоваться для подключения широкого спектра устройств. В результате появились стандарты USB 1.0, USB 2.0, USB 3.0, USB 3.1, USB 3.2, USB C и USB 4.

Вместе с появлением технологии появилась и надобность в ее диагностике и мониторинге. Первые программы для мониторинга USB были простыми утилитами, которые выводили информацию о подключенных устройствах – их название, тип, серийный номер. Они служили скорее для любопытства, чем для практического использования. Однако с развитием компьютерных технологий, в частности с распространением мобильных устройств и ростом популярности интернета, роль USB-мониторинга стала более значимой.

С появлением вредоносных программ, распространяемых через USB-носители, возникла необходимость в программах, способных блокировать подключение потенциально опасных устройств. Некоторые программы мониторинга отслеживают передачу данных через USB, позволяя отследить попытки несанкционированного копирования информации.

Также некоторые программы мониторинга позволяют просматривать данные, передаваемые через USB, что очень полезно для тестирования и отладки программного обеспечения.

Вместе с тем USB-мониторы могут отслеживать количество потребляемой энергии подключенными устройствами. Это важно для мобильных устройств с ограниченным ресурсом батареи. Для оптимизации работы системы, программы мониторинга позволяют приоритизировать доступ к USB-портам для определенных устройств.

В настоящее время USB-мониторинг является важной частью программного обеспечения для многих устройств, обеспечивая безопасность, управление и отладку USB-подключений. По мере развития технологии USB и появления новых устройств, программные средства мониторинга будут продолжать совершенствоваться, предлагая еще более широкий спектр функций и возможностей.

1. АНАЛИЗ ПРЕДМЕНТНОЙ ОБЛАСТИ
   1. Обзор аналогов

**1.1.1** Программное средство USBDeview

USBDeview - очень популярная утилита, позволяющая просматривать все подключенные и ранее подключенные USB-устройства. Показывает их свойства, включая ID устройства, дату подключения, тип, производителя и т.д. Позволяет отключать устройства и удалять их из реестра.

Приложение доступно для скачивания на Microsoft Windows 11, 10, 8.1, 8, 7, XP. Внешний вид программы представлен на рисунке 1.1.

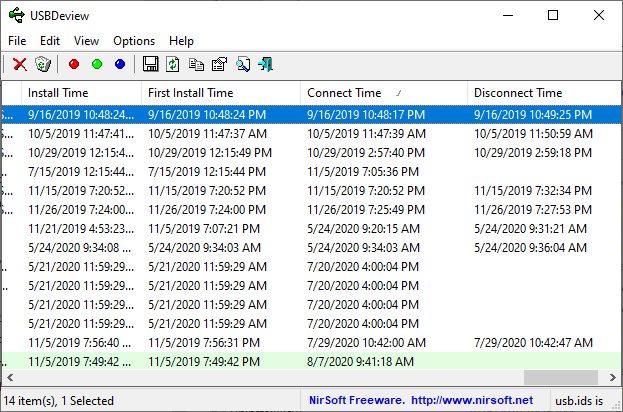


Рисунок 1.1 – Программное средство USBDeview

**1.1.2** Программное средство USB Monitor Pro

USB Monitor Pro позволяет отслеживать все USB-события в режиме реального времени, записывать их в лог-файлы, блокировать подключение устройств, ограничивать скорость передачи данных и многое другое.

Приложение доступно для скачивания на Microsoft Windows 11, 10, 8.1, 8, 7, XP. Внешний вид программы представлен на рисунке 1.2.

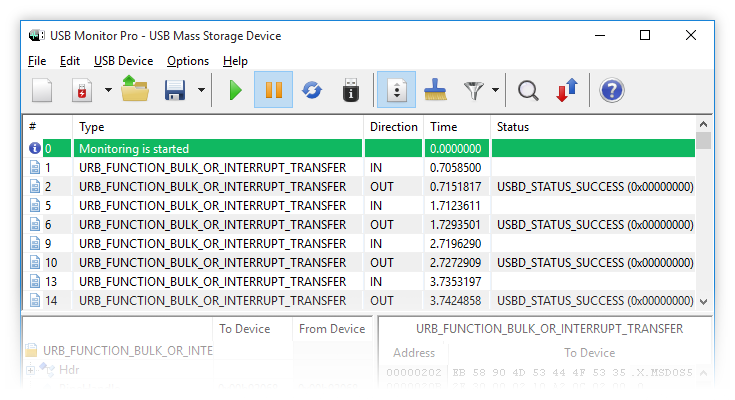


Рисунок 1.2 – Программное средство USB Monitor Pro

**1.1.3** Программное средство USB Device Tree Viewer

USB Device Tree Viewer визуализирует дерево USB-устройств, показывая связи между контроллером, хабом и подключенными устройствами. Приложение взаимодействует со встроенным в Windows “USBView” – инструментом для разработчиков – и показывает детальную информацию о портах и подключенных устройвах в удобном виде.

Приложение доступно для скачивания на Microsoft Windows 11, 10, 8.1, 8, 7. Внешний вид программы представлен на рисунке 1.3.

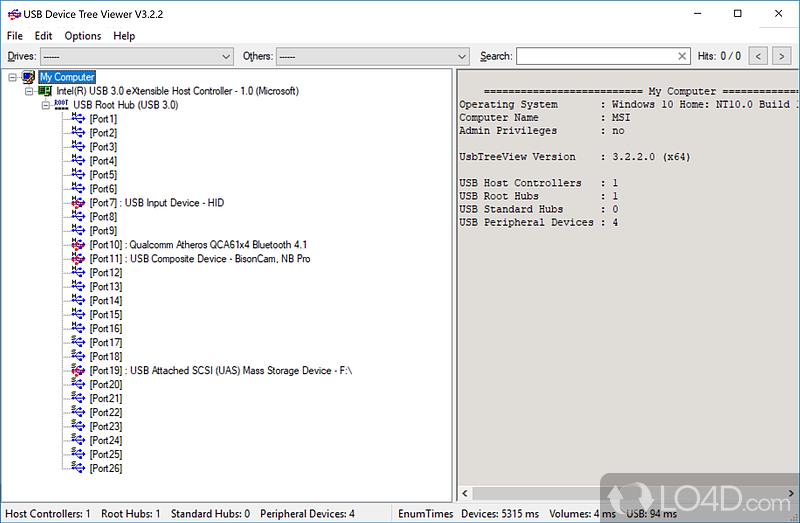


Рисунок 1.3 – Программное средство USB Device Tree Viewer

* 1. Постановка задачи

На основе проведенного анализа аналогов программного средства принято решение разработать приложение, обладающее следующими функциями:

* сбор информации о всех когда-либо подключенных USB-устройствах;
* сбор информации о всех устройствах компьютера, работающих по протоколу USB;
* отображение информации о USB-устройствах в сводной таблице;
* отслеживание подключения и отключения USB-устройств;
* отслеживание изменения реестра Windows для USB-устройств.

Для разработки программного средства будет использоваться язык программирования С++ и среда разработки Visual Studio 2022.

1. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА
   1. Структура программы

При разработке приложения будет использовано 5 заголовочных файлов и 5 модулей:

* General.h – заголовочный файл, содержащий подключение всех необходимых сторонних библиотек и зависимотей проекта, а также объявления структур и перечислимых типов.
* App.h – заголовочный файл, содержащий объявление пользовательского класса MyApp. Класс MyApp реализует основную логику и визуальный интерфейс программного средства.
* MyCircleButton.h – заголовочный файл, содержащий объявление пользовательского класса MyCircleButton, используемого для создания круглых кнопок визуальный интерфейса программного средства.
* UsbAlgs.h – заголовочный файл, содержащий прототипы пользовательских функций для поиска и структурирования информации о когда-либо подключенных USB-устройствах.
* SortAlgs.h – заголовочный файл, содержащий прототипы функций для сортитровки массива структур USBDeviceInfo.
* Entry.cpp – модуль, содержащий точку входа в программу.
* App.cpp – модуль, содержащий реализацию класса MyApp.
* MyCircleButton.cpp – модуль, содержащий реализацию класса MyCircleButton.
* UsbAlgs.cpp – модуль, содержащий реализацию прототипов функций заголовочного файла UsbAlgs.h.
* SortAlgs.cpp – модуль, содержащий реализацию прототипов функций заголовочного файла SortAlgs.h.
  1. Проектирование интерфейса программного средства

**2.2.1** Главное окно приложения

Главное окно приложения должно содержать верхнюю панель, включающую надпись с названием приложения, кнопки для скрытия, расширения и закрытия окна. Также окно должно содержать сводную таблицу с информацией о USB-устройствах и нижнюю панель состояния, отображающую количество найденных USB-устройств. Макет главного окна программного средства представлен на рисунке 2.1.

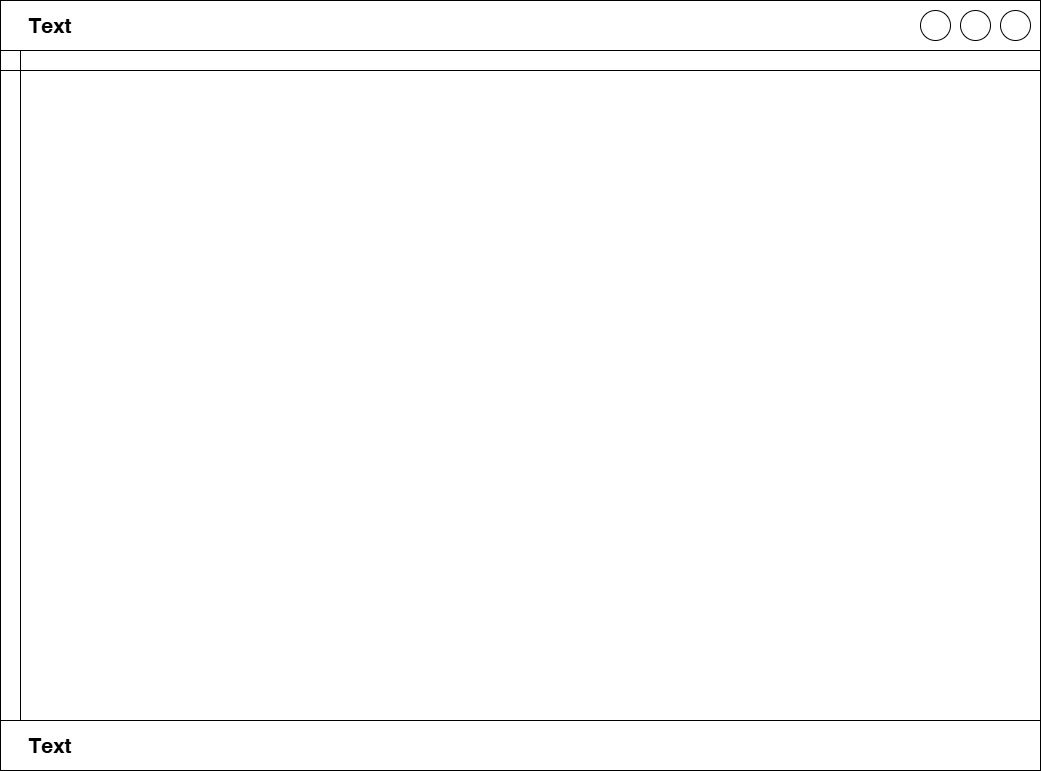


Рисунок 2.1 – Главное окно приложения

**2.2.2** Верхняя панель главного окна

Верхняя панель должна содержать название приложения рядом с левым верхним углом и набор из трех кнопок системного меню рядом с правым верхним углом. Крайняя левая кнопка должна расширять окно до максимального размера и быть окрашенной в зеленый цвет. Центральная кнопка должна скрывать окно и быть окрашенной в желтый цвет. Крайняя правая кнопка должна закрывать приложение и быть окрашенной в красный цвет. Макет верхней панели главного окна представлен на рисунке 2.2.

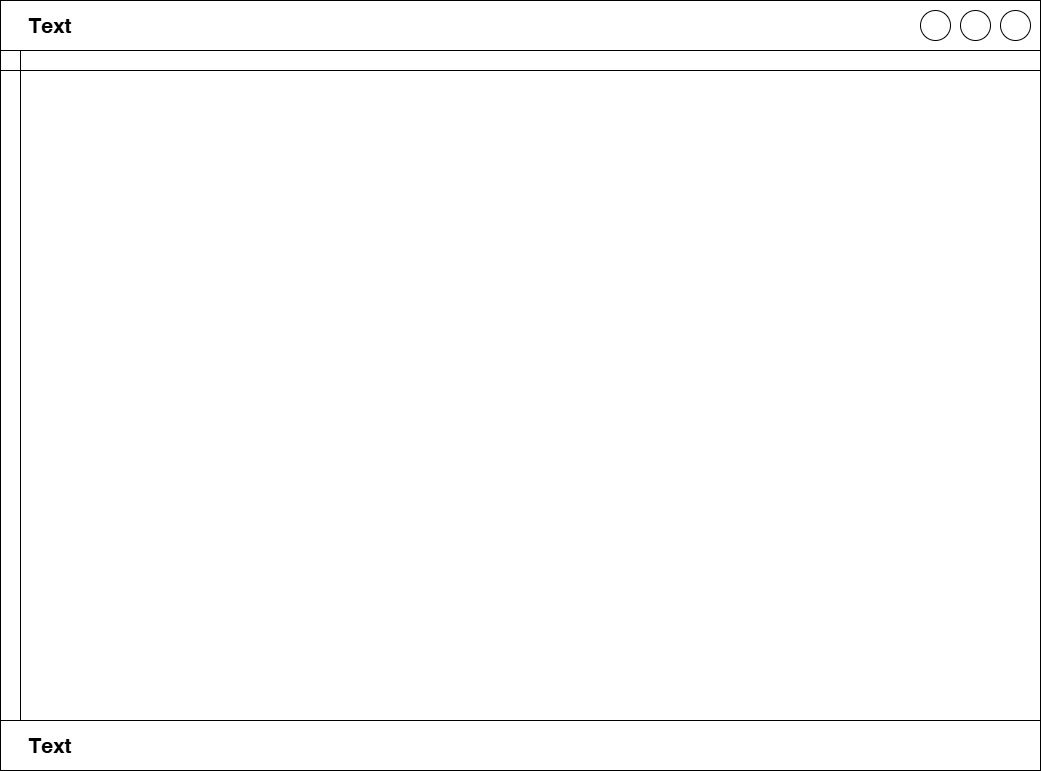


Рисунок 2.2 – Верхняя панель главного окна

**2.2.3** Сводная таблица главного окна

Сводная таблица должна содержать информацию о всех когда-либо подключенных USB-устройствах и предоставлять 17 характеристик для каждого устройства. В случае, если характеристика неизвестна, стоит оставлять поле пустым или вписывать в соответствующую клетку слово «Unknown». Макет сводной таблицы представлен на рисунке 2.3.

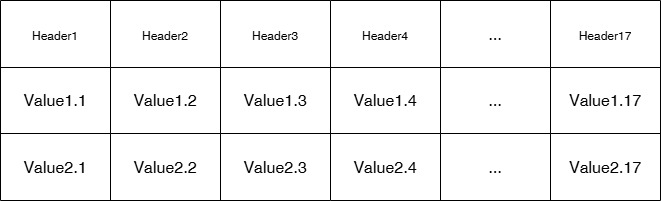


Рисунок 2.3 – Сводная таблица главного окна

**2.2.4** Панель состояния главного окна

Нижняя панель должна содержать надпись с информацией о количестве найденных устройств. Макет панели состояния главного окна представлен на рисунке 2.4.

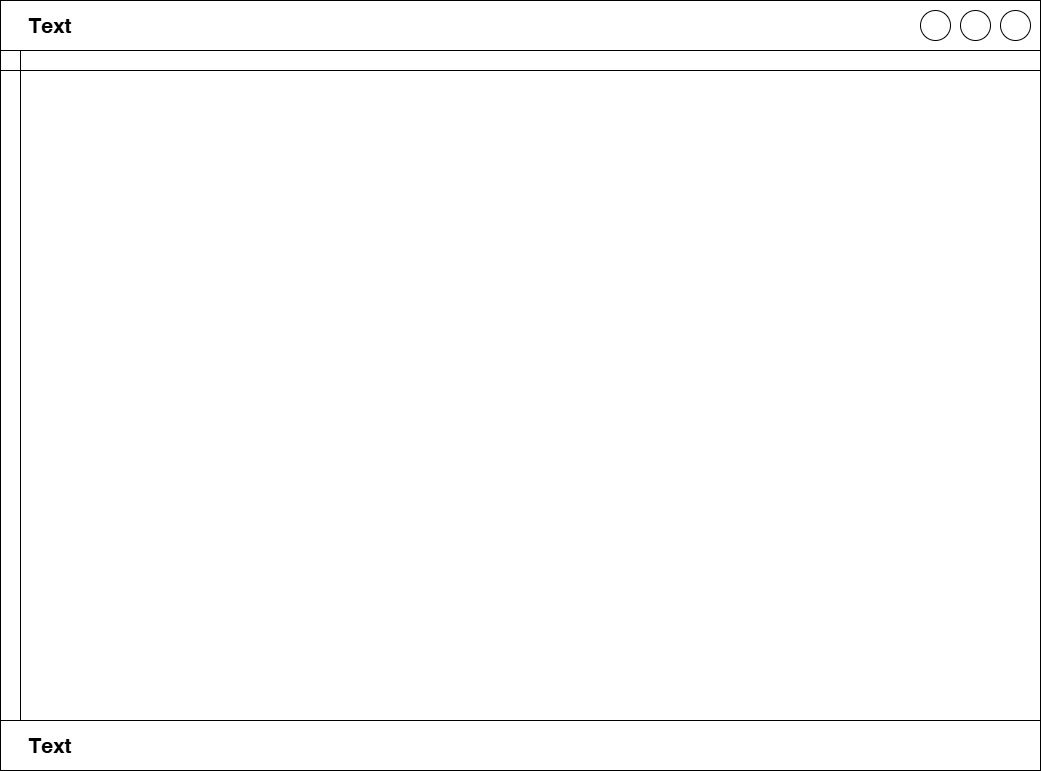


Рисунок 2.4 – Панель состояния главного окна

* 1. Проектирование функционала программного средства

При создании приложения очень важно сразу определить задачу и цели, а также хорошо составить рабочие алгоритмы. В приложении должен присутствовать следующий функционал:

* сбор информации о USB-устройствах с использованием реестра Windows;
* преобразование значений полей реестра Windows в удобный для обработки информации формат;
* получение текущего состояния устройства, включая следующее:

подключено ли устройство, запрещено ли устройство, безопасно ли извлекать устройство.

**2.3.1** Сбор информации о USB-устройствах из раздела реестра Windows.

Перед сбором информации необходимо будет получить доступ к ключу HKEY\_LOCAL\_MACHINE реестра Windows. Далее будет необходимо указать один из разделов, содержащих информацию о USB-устройствах и начать поочередно извлекать ключ каждого из них. Блок-схема алгоритма данной функции приведена на рисунке 2.5.

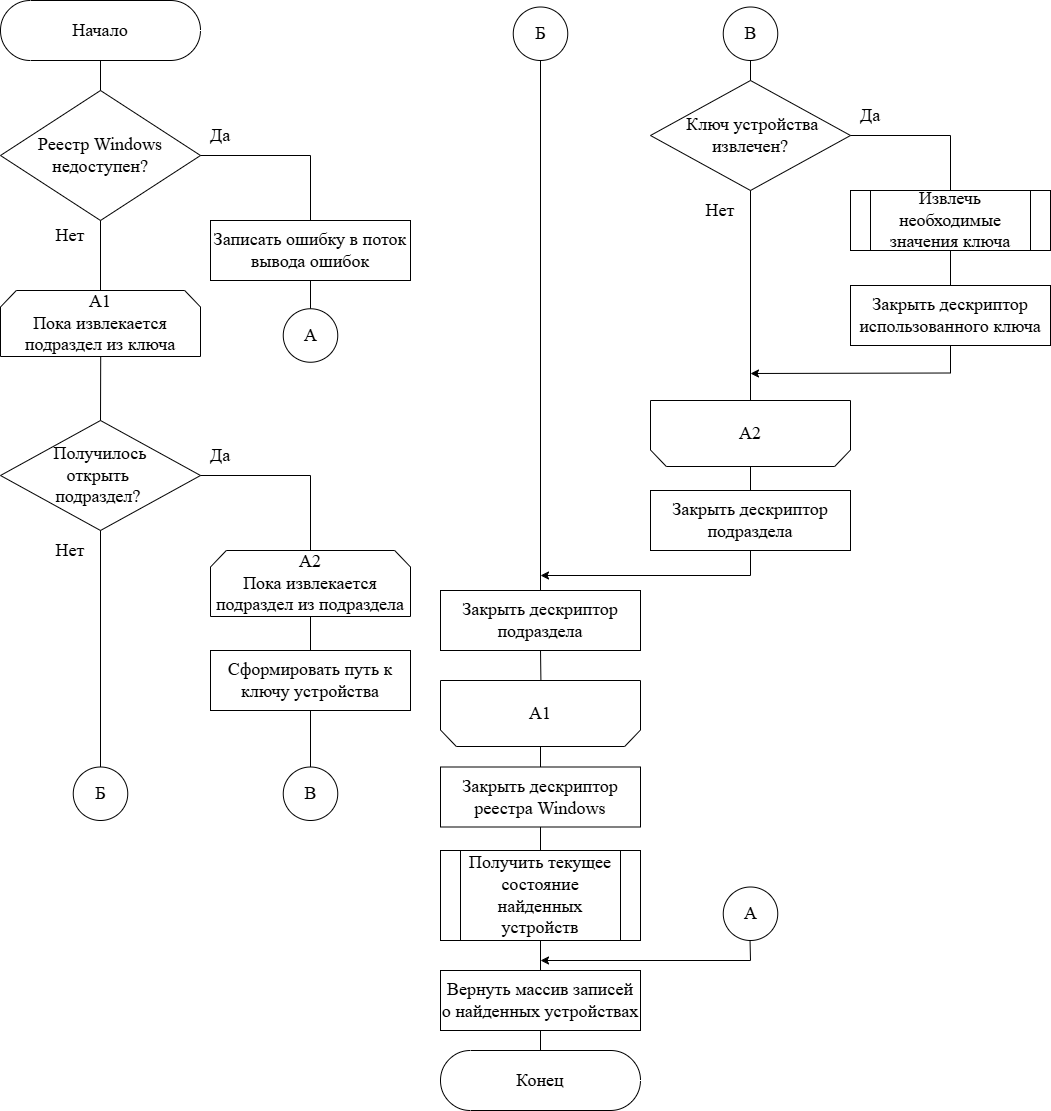
****

Рисунок 2.5 – Блок-схема функции сбора информации о USB-устройствах

**2.3.2** Преобразование строковых значений полей реестра Windows в удобный для обработки информации формат.

Для корректного извлечения строковых значений ключа устройства должен быть предварительно созданный буфер, куда, в случае успешного извлечения, будет помещено значение. В случае ошибки извлечения функция должна возвращать значение «Unknown». Блок-схема алгоритма функции преобразование строковых значений ключа приведена на рисунке 2.6.



Рисунок 2.6 – Блок-схема преобразования значений полей реестра Windows

**2.3.3** Преобразование двоичных значений полей реестра Windows в удобный для обработки информации формат.

Для корректного извлечения двоичных значений ключа устройства должен быть предварительно созданный буфер, куда, в случае успешного извлечения, будет помещено значение. Далее должны будут быть выполнены проверки на верный код и тип возвращаемого значения. В случае ошибки функция должна возвращать значение, равное -1.Блок-схема алгоритма функции преобразование двоичных значений ключа приведена на рисунке 2.7.

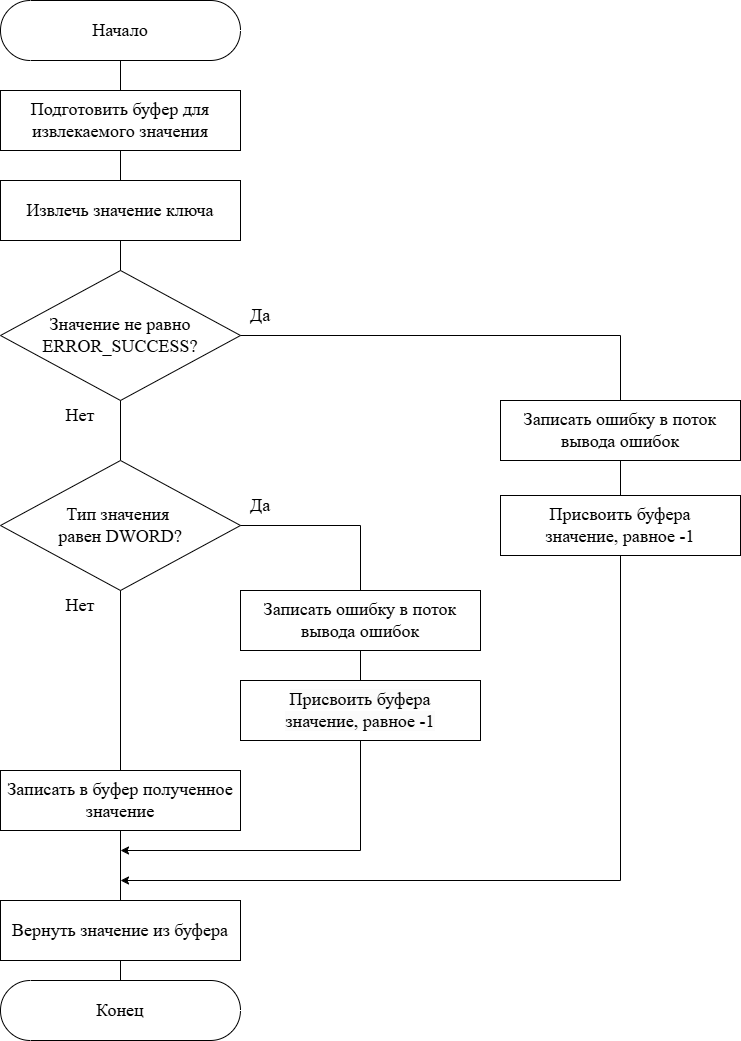


Рисунок 2.7 – Блок-схема преобразования значений полей реестра Windows

**2.3.4** Получение текущего состояния USB-устройства.

Реестр Windows может не содержать информации о текущем состоянии устройства. Для получения этой информации необходимо использовать функции WinAPI, связанные с получением набора сведений об устройстве. Будут необходимы для поиска только USB-устройства, которые присутствуют в системе на данный момент и чей GUID совпадает с любым GUID из ранее полученного массива устройств. Блок-схема алгоритма данной функции приведена на рисунке 2.7.

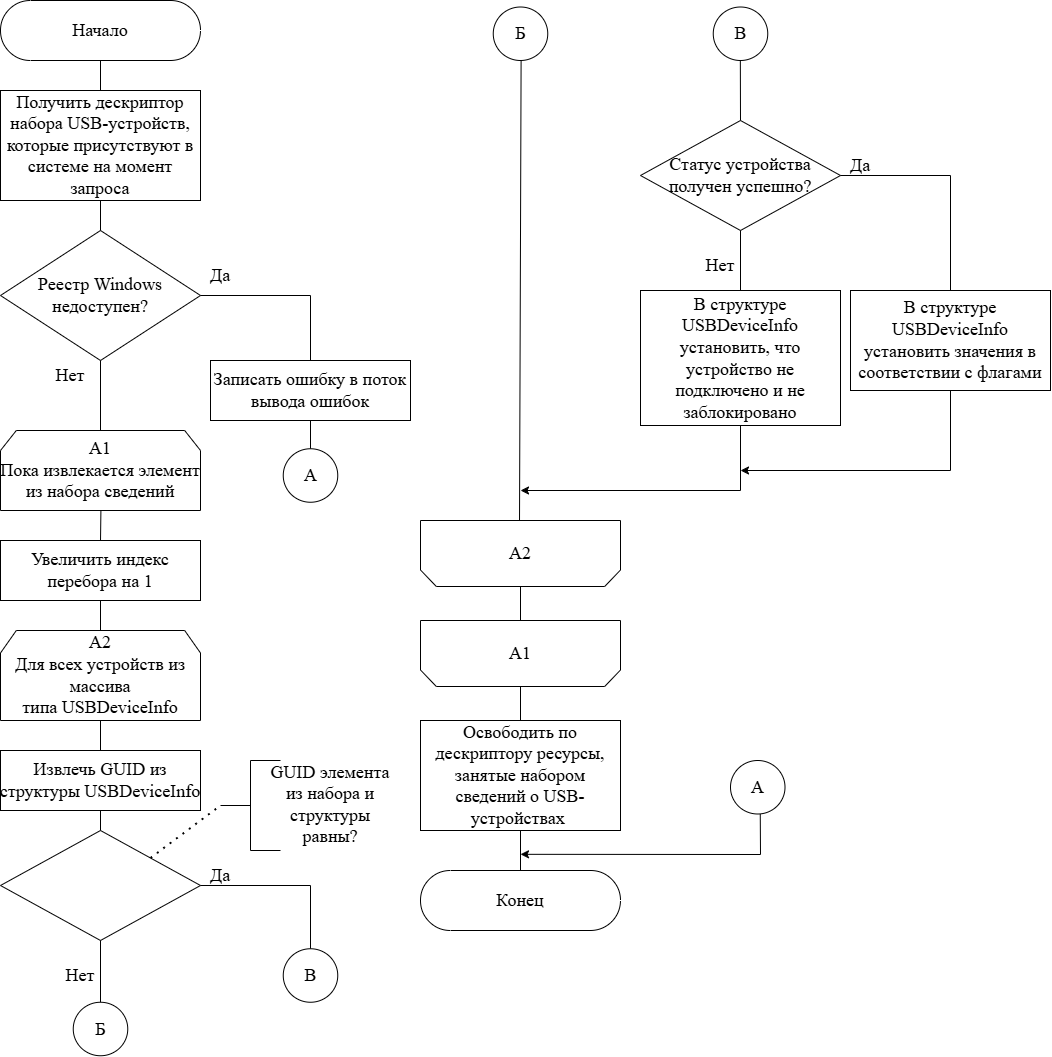


Рисунок 2.8 – Блок-схема функции получения текущего состояния USB-устройства

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
   1. Создание графических ресурсов для главного окна

**3.1.1** Инициализация графических ресурсов приложения

Перед отрисовкой главного окна необходимо инициализировать ресурсы, который в дальнейшем будут необходимы при использовании программного интерфейса Direct2D. Их инициализация определена в обработчике окна для сообщения WM\_CREATE. Код обработчика представлен ниже.

int MyApp::OnCreate()

{

if (FAILED(D2D1CreateFactory(

D2D1\_FACTORY\_TYPE\_SINGLE\_THREADED,

&pFactory)))

{

return -1; // Fail CreateWindowEx.

}

if (FAILED(DWriteCreateFactory(

DWRITE\_FACTORY\_TYPE\_SHARED,

\_\_uuidof(IDWriteFactory),

reinterpret\_cast<IUnknown\*\*>(&pDWriteFactory))))

{

return -1;

}

isMaximized = false;

numOfDevices = 0;

std::thread registryWatcher(WatchRegistryKey, HKEY\_LOCAL\_MACHINE, L"SYSTEM\\Cur rentControlSet\\Enum", handler);

registryWatcher.detach();

return 0;

}

**3.1.2** Использование графических ресурсов при отрисовке окна

При каждом использовании программного интерфейса графической библиотеки Direct2D необходимо заново инициализировать цель отрисовки, т.к. после отрисовки она перестает быть доступной. Функция повторной инициализации также включает в себя и создание кистей для правильной отрисовки фона окна. Код функции представлен ниже.

HRESULT MyApp::CreateGraphicsResources()

{

HRESULT hResult = S\_OK;

if (NULL == pRenderTarget)

{

/\* window \*/

RECT clientRect;

GetClientRect(handler, &clientRect);

D2D1\_SIZE\_U size = D2D1::SizeU(

clientRect.right - clientRect.left,

clientRect.bottom - clientRect.top);

/\* render \*/

hResult = pFactory->CreateHwndRenderTarget(

D2D1::RenderTargetProperties(

D2D1\_RENDER\_TARGET\_TYPE\_DEFAULT,

D2D1::PixelFormat(DXGI\_FORMAT\_UNKNOWN, D2D1\_ALPHA\_MODE\_PREMULT

PLIED)),

D2D1::HwndRenderTargetProperties(handler, size),

&pRenderTarget);

if (SUCCEEDED(hResult))

{

/\* background brush \*/

hResult = pRenderTarget->CreateGradientStopCollection(

gradientStops,

2,

D2D1\_GAMMA\_2\_2,

D2D1\_EXTEND\_MODE\_CLAMP,

&pGradientStops);

if (SUCCEEDED(hResult))

{

hResult = pRenderTarget->CreateLinearGradientBrush(

D2D1::LinearGradientBrushProperties(

D2D1::Point2F(clientRect.left, clientRect.top),

D2D1::Point2F(clientRect.right, clientRect.bottom)),

pGradientStops,

&pBackgroudnGradientBrush);

}

/\* header text brush \*/

pRenderTarget->CreateSolidColorBrush(

D2D1::ColorF(D2D1::ColorF::White),

&pHeaderTextBrush);

}

}

return hResult;

}

* 1. Сортировка строк в сводной таблице

**3.2.1** Алгоритм сортировки массива структур UsbDeviceInfo

Функция реализует сортировку слиянием с использованием нескольких потоков. При сортировке учитывается, по какой характеристике сортируется массив, поэтому среди параметров передается sortType. Код функции сортировки представлен ниже.

void MergeSortMultiThreaded(std::vector<USBDeviceInfo>& files, int maxThreads, DEV\_INFO sortType)

{

int n = files.size();

std::vector<HANDLE> hThreads;

hThreads.reserve(maxThreads);

for (int subSize = 1; subSize < n; subSize \*= 2)

{

for (int left = 0; left < n; left += 2 \* subSize)

{

int mid = (left + subSize - 1) < (n - 1) ? (left + subSize - 1) :

(n - 1);

int right = (left + 2 \* subSize - 1) < (n - 1) ? (left + 2 \* subSize - 1)

: (n - 1);

if (left < right)

{

if (hThreads.size() < maxThreads)

{

ThreadArgs\* args = new ThreadArgs{ &files, left, mid, right, sortType };

HANDLE hThread = CreateThread(NULL, 0, MergeThreadProc, args, 0, NULL);

if (hThread == NULL)

{

delete args;

continue;

}

hThreads.push\_back(hThread);

}

else

{

DWORD endedThreadIndex = WaitForMultipleObjects(1, &hThreads[0], FALSE, INFINITE);

CloseHandle(hThreads[endedThreadIndex - WAIT\_OBJECT\_0]);

hThreads.erase(hThreads.begin() + endedThreadIndex –

WAIT\_OBJECT\_0);

ThreadArgs\* args = new ThreadArgs{ &files, left, mid, right, sortType };

HANDLE hThread = CreateThread(NULL, 0, MergeThreadProc, args, 0, NULL);

hThreads.push\_back(hThread);

}

}

}

if (!hThreads.empty())

{

WaitForMultipleObjects(hThreads.size(), &hThreads[0], TRUE, INFINITE);

for (HANDLE hThread : hThreads) CloseHandle(hThread);

hThreads.clear();

}

}

}

**3.2.2** Реагирование на нажатие заголовка столбца в таблице

При нажатии на заголовок одного из столбцов сводной таблице массив сортируется по характеристике, соответствующей нажатому столбцу. Функция является обработчиком сообщения WM\_NOTIFY, посылаемого сводной таблицей. Код обработчика представлен ниже.

void MyApp::OnNotify(LPARAM lParam)

{

LPNMHDR pnmh = (LPNMHDR)lParam;

if (pnmh->hwndFrom == ListView\_GetHeader(hMainList))

{

switch (pnmh->code)

{

case HDN\_ITEMCLICK:

{

LPNMHEADER phdr = (LPNMHEADER)lParam;

int clickedColumn = phdr->iItem;

MergeSortMultiThreaded(deviceInfos, MAX\_THREADS, (DEV\_INFO)clicked Column);

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

break;

}

}

}

* 1. Отслеживание изменений в реестре Windows и подключений USB-устройств

**3.3.1.** Отслеживание изменений в реестре Windows

При запуске приложения создается отдельный поток, в котором запускается функция отслеживания изменений. Отслеживаются определенные ключи реестра, содержащие информацию о USB-устройствах. При изменении этих ключей или добавлении новых функция будет получать уведомление и переходить в соответствующий обработчик. Код данной функции представлен ниже.

void WatchRegistryKey(HKEY hKey, LPCWSTR subKey, HWND hMainApp)

{

HKEY key;

HANDLE hChange;

if (RegOpenKeyEx(hKey, subKey, 0, KEY\_NOTIFY, &key) != ERROR\_SUCCESS)

{

std::wcerr << L"Failed to open registry key." << std::endl;

return;

}

hChange = CreateEvent(NULL, TRUE, FALSE, NULL);

if (hChange == NULL)

{

std::wcerr << L"Failed to create event." << std::endl;

RegCloseKey(key);

return;

}

while (true)

{

if (RegNotifyChangeKeyValue(key, TRUE, REG\_NOTIFY\_CHANGE\_NAME | REG\_NO TIFY\_CHANGE\_LAST\_SET, hChange, TRUE) != ERROR\_SUCCESS)

{

std::wcerr << L"Failed to set registry notification." << std::endl;

break;

}

DWORD result = WaitForSingleObject(hChange, 1000);

if (result == WAIT\_OBJECT\_0)

{

SendMessage(hMainApp, WM\_DEVICECHANGE, NULL, NULL);

}

else if (result == WAIT\_TIMEOUT){/\*do nothing or something\*/}

else

{

std::wcerr << L"Error in waiting for registry change." << std::endl;

break;

}

}

CloseHandle(hChange);

RegCloseKey(key);

}

**3.3.2** Реагирование на подключение и отключение устройств

Функции являются обработчиками сообщений, поступающих главному окну приложения. Событию подключения и событию отключения устройств соответствует код макроса WM\_DEVICECHANGE. Код функции обработчика представлен ниже.

void MyApp::OnDeviceChange()

{

deviceInfos = GetAllUsbDevices();

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения были выявлены некоторые ошибки и недочеты в работе программного средства.

Одним из недочетов было то, что при обновлении информации о USB-устройствах сводная таблица приложения не очищалась от информации, полученной при прошлом обновлении. Решением стало использование макроса ListView\_DeleteAllItems, который очищает все позиции в указанном компоненте ListView. Код исправленных функций представлен ниже.

void MyApp::OnDeviceChange()

{

deviceInfos = GetAllUsbDevices();

ListView\_DeleteAllItems(hMainList);

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

void MyApp::OnNotify(LPARAM lParam)

{

LPNMHDR pnmh = (LPNMHDR)lParam;

if (pnmh->hwndFrom == ListView\_GetHeader(hMainList))

{

switch (pnmh->code)

{

case HDN\_ITEMCLICK:

{

LPNMHEADER phdr = (LPNMHEADER)lParam;

int clickedColumn = phdr->iItem;

MergeSortMultiThreaded(deviceInfos,

MAX\_THREADS, (DEV\_INFO)clickedColumn);

ListView\_DeleteAllItems(hMainList);

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

break;

}

}

}

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Интерфейс программного средства

**5.1.1** Главно окно приложения

Главное окно приложения содержит верхнюю панель с системным меню, сводную таблицу с информацией о USB-устройствах и нижнюю панель. Внешний вид главного окна приложения представлен на рисунке 5.1.

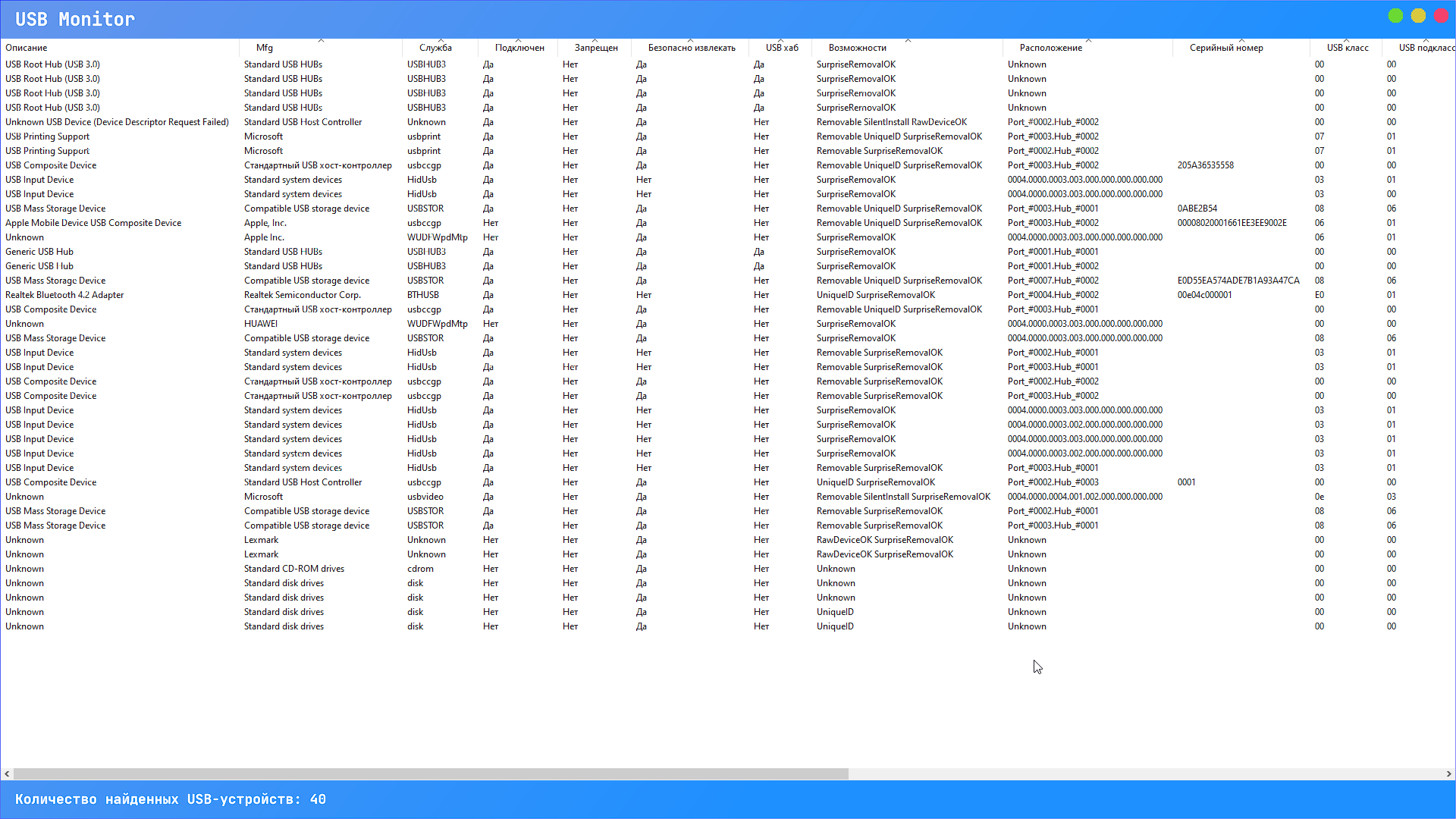


Рисунок 5.1 – Главное окно приложения

**5.1.2** Верхняя панель главного окна

Верхняя панель содержит 3 кнопки системного меню, отвечающих за взаимодействие с главным окном. Внешний вид верхней панели представлен на рисунке 5.2.



Рисунок 5.2 – Верхняя панель главного окна

**5.1.3** Сводная таблица главного окна

Сводная таблица содержит в себе информацию о всех когда-либо подключенных USB-устройствах к компьютеру, запустившему приложение. Для каждого устройства отображается 17 характеристик. Внешний вид сводной таблицы представлен на рисунке 5.3.

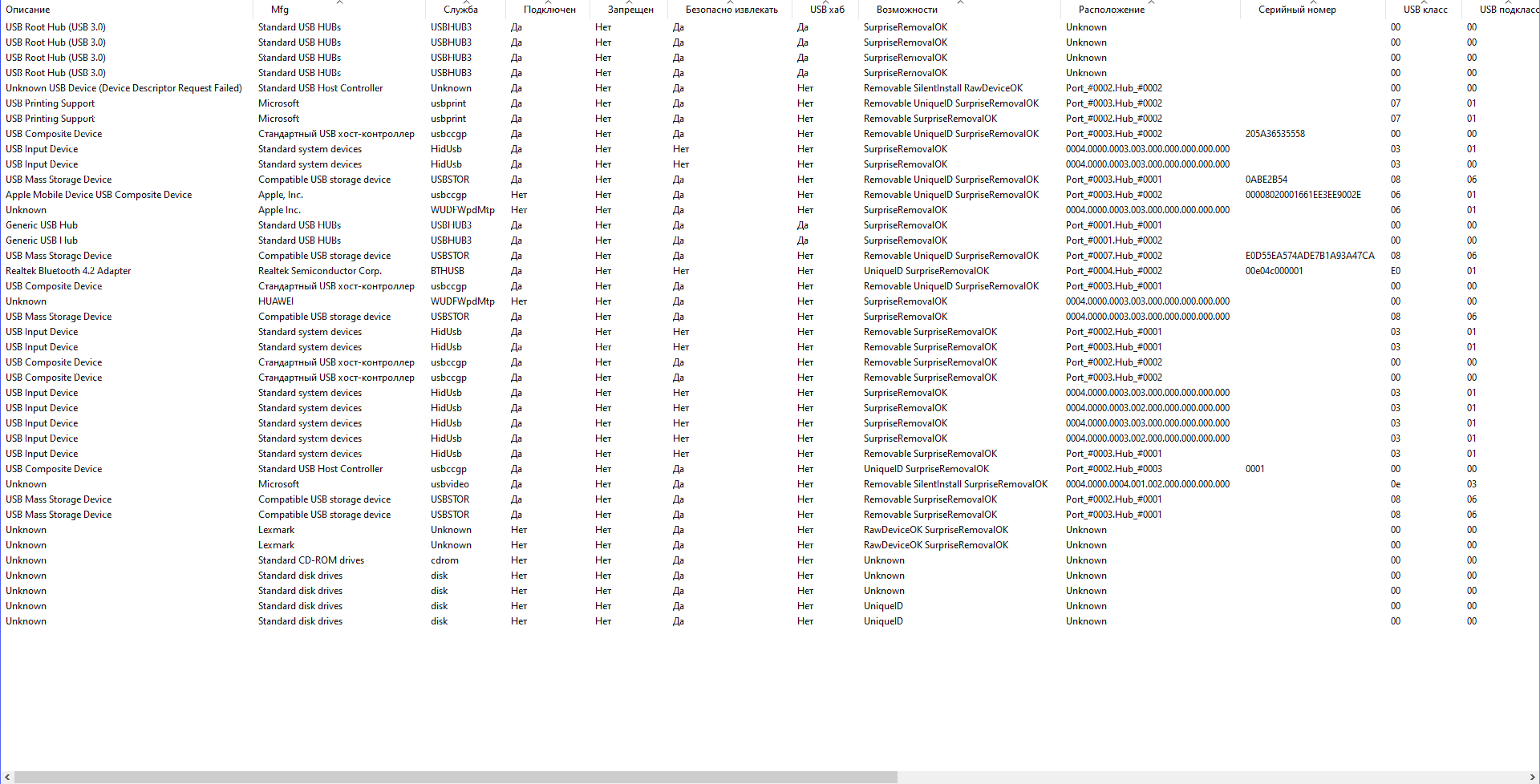


Рисунок 5.3 – Сводная таблица главного окна

**5.1.4** Нижняя панель главного окна

Нижняя панель является панелью состояния, отображающей количество всех найденных USB-устройств. Внешний вид нижней панели представлен на рисунке 5.4.



Рисунок 5.4 – Страница с информацией о приложении

* 1. Управление программным средством

**5.2.1** Использование кнопок верхней панели

Для использования любой из кнопок верхней панели приложения необходимо нажать на соответствующую кнопку левой кнопкой мыши.

Крайняя левая кнопка отвечает за расширение главного окна и возврат окна к стандартному размеру. Центральная кнопка отвечает за сокрытие главного окна. Крайняя правая кнопка отвечает за закрытие приложения.

**5.2.2** Использование сводной таблицы

Информация в сводной таблице отображается и обновляется без действий со стороны пользователя. Для сортировки строк в таблице по определенному критерию необходимо нажать на заголовок соответствующего столбца. Для прорутки таблицы вертикально либо горизонтально необходимо использовать соответствующую направлению полосу прокрутки.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта было разработано программное средство «Монитор обращений к USB», которое позволит пользователям операционной системы Microsoft Windows 10 Professional Edition отслеживать обращения к USB их компьютера и собирать информацию об обращающихся устройствах. При разработке данного средства были успешно реализованы следующие базовые функции:

* сбор информации о всех когда-либо подключенных USB-устройствах;
* сбор информации о всех устройствах компьютера, работающих по протоколу USB;
* отображение информации о USB-устройствах в сводной таблице;
* отслеживание подключения и отключения USB-устройств;
* отслеживание изменения реестра Windows для USB-устройств.

Для успешного выполнения поставленных задач потребовалось изучить принципы создания WinAPI-приложений, использования функций WinAPI для работы с реестром Windows, взаимодействия с графическими API библиотек Direct2D и GDI+.

Существует множество способов улучшить данное программное средство:

* возможность изменения пользователем информации о USB-устройствах в таблице;
* сбор еще большего количества характеристик каждого устройства;
* аналих собранных данных и вывод производных метрик;
* отслеживание информации, передаваемой по USB-портам;
* экспорт полученной информации в различные форматы.

Программное средство «Монитор обращений к USB» позволяет пользователям легко и быстро отследить и проконтролировать наличие и активность всех когда-либо подключенных USB-устройств.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Объектно-ориентированное программирование на языках Delphi и C++: учебное пособие для студентов [Электронный ресурс] / А. Н. Вальвачев, К. А. Сурков, Д. А. Сурков, Ю. М. Четырько. – БГУИР, 2016 . – 432 с.
2. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Р. Джеффри – Спб.: БХВ-Петербург, 2008. – 128 с.: ил. ISBN 978-5-699-78090-7.
3. Внутреннее устройство Windows / М. Руссинович, Д. Соломон. – Питер : БХВ-Петербург, 2019. – 431 с.: ил. ISBN 978-5-4461-1426-9.
4. Undocumented Windows NT / P. Dabak, S. Phadke, M. Borate – Wiley, 1999. – 227 с: ил. ISBN 978-0-764-54569-6.
5. Windows NT/2000 Native API / G. Nebbett – Sams Publishing, 2000. – 399 с.: ил. ISBN 978-1-578-70199-5.
6. Системное программирование в среде Windows / Х. Джонсон – Издательский дом "Вильямс", 2005. – 95 с.: ил. ISBN 5-8459-0879-5.
7. Programming reference for the Win32 API [Электронный ресурс] / Microsoft Learn – Microsoft Coprotation, 2024.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код**

**General.h**

#ifndef GENERAL\_H

#define GENERAL\_H

/\* App \*/

#include <stdlib.h>

#include <exception>

#include <iostream>

#ifndef \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#endif

#ifndef UNICODE

#define UNICODE

#endif

enum class MENU\_ID {

CLOSE\_BUTTON = 10,

MINIMIZE\_BUTTON,

MAXIMIZE\_BUTTON,

MAIN\_LIST = 20

};

/\* Utility \*/

#include <string>

#include <format>

#include <sstream>

#include <vector>

#include <regex>

template <class T> void SafeRelease(T\*\* ppT)

{

if (\*ppT)

{

(\*ppT)->Release();

\*ppT = NULL;

}

}

/\* UI \*/

#define SCREEN\_WIDTH GetSystemMetrics(SM\_CXSCREEN)

#define SCREEN\_HEIGHT GetSystemMetrics(SM\_CYSCREEN)

#include <windows.h>

#include <commctrl.h>

#include <d2d1.h>

#include <dwrite.h>

#include <gdiplus.h>

/\* Algs \*/

#include <initguid.h>

#include <devguid.h>

#include <usbiodef.h>

#include <devpkey.h>

#include <setupapi.h>

#include <cfgmgr32.h>

#include <thread>

enum DEV\_INFO : int {

DESC,

MFG,

SERVICE,

IS\_CONNECTED,

IS\_DIASBLED,

IS\_SAFE\_TO\_UNPLUG,

IS\_USB\_HUB,

CAPABILITIES,

LOCATION,

SERIAL\_NUM,

USB\_CLASS,

USB\_SUBCLASS,

USB\_PROTOCOL,

ID,

DEV\_GUID,

HARDWARE\_ID,

PARENT\_PREFIX

};

struct USBDeviceInfo {

std::wstring description;

std::wstring manufactoring;

std::wstring serviceName;

bool isConnected;

bool isDisabled;

bool isSafeToUnplug;

bool isUsbHub;

std::wstring capabilities;

std::wstring location;

std::wstring serialNumber;

std::wstring usbClass;

std::wstring usbSubclass;

std::wstring usbProtocol;

std::wstring id;

std::wstring devClassGUID;

std::wstring hardwareID;

std::wstring parentIDPrefix;

};

#endif // GENERAL\_H

**App.h**

#ifndef MY\_APP\_CLASS

#define MY\_APP\_CLASS

#include "MainApp\Headers\Components\MyCircleButton.h"

class MyApp

{

public:

/\* funcs \*/

explicit MyApp();

~MyApp();

int Run();

/\* static funcs \*/

static LRESULT CALLBACK AppProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);

/\* static fields \*/

static constexpr const wchar\_t\* appName = L"File Sorter";

static constexpr const wchar\_t\* className = L"MyAppWindowClass";

private:

/\* funcs \*/

//initialization

void InitWindow();

void CreateControls();

void InitMainList();

//logic

//main window

LRESULT CALLBACK WndProc(UINT message, WPARAM wParam, LPARAM lParam);

HRESULT CreateGraphicsResources();

int OnCreate();

void OnPaint();

void OnSize();

void OnLBtnDown();

void OnCommand(WPARAM menuId);

void OnDestroy();

void OnDeviceChange();

void OnNotify(LPARAM lParam);

//list

void FillMainList();

void SetSubItems(int itemIndex, const wchar\_t\*\* texts, int count);

void InsertItemWithSubItems(int itemIndex, const wchar\_t\*\* texts, int count);

void UpdateColumnsWidths();

//finalization

void DiscradControls();

void DiscardGraphicsResources();

/\* static funcs \*/

/\* fields \*/

HWND handler;

HFONT hHeaderFont;

int numOfDevices;

//d2d1

ID2D1Factory\* pFactory;

IDWriteFactory\* pDWriteFactory;

ID2D1HwndRenderTarget\* pRenderTarget;

ID2D1LinearGradientBrush\* pBackgroudnGradientBrush;

ID2D1LinearGradientBrush\* pWindowBorderBrush;

ID2D1SolidColorBrush\* pHeaderTextBrush;

//gdi

Gdiplus::GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

//objects

MyCircleButton\* closeButton;

MyCircleButton\* minimizeButton;

MyCircleButton\* maximizeButton;

HWND hMainList;

std::vector<USBDeviceInfo> deviceInfos;

//utility

double wndScalingCoef\_X;

double wndScalingCoef\_Y;

bool isMaximized;

/\* static fields \*/

static const int TOP\_MENU\_BTN\_RADIUS = 20;

static const int TOP\_MENU\_BTN\_GAP = 10;

static const int TOP\_MENU\_BTN\_Y = 10;

static const int HEADER\_HEIGHT = 50;

static const int LIST\_MARGIN = 1;

static const int NUM\_OF\_COLS = 17;

static const int MAX\_THREADS = 4;

const wchar\_t\* columnHeaders[NUM\_OF\_COLS] = {

L"Описание", L"Mfg", L"Служба",

L"Подключен", L"Запрещен", L"Безопасно извлекать", L"USB хаб", L"Возможности",

L"Расположение",

L"Серийный номер", L"USB класс", L"USB подкласс", L"USB протокол",

L"ID", L"Классовый GUID", L"Hardware ID", L"Префикс родительского ID",

};

};

#endif // MY\_APP\_CLASS

**UsbAlgs.h**

#ifndef USB\_ALGS

#define USB\_ALGS

/\* common \*/

std::wstring GetRegistryStringValue(HKEY hKey, const std::wstring& valueName);

DWORD GetRegistryDWORDValue(HKEY hKey, const std::wstring& valueName);

std::vector<USBDeviceInfo> ListUSBDevices(const std::wstring& registryPath);

std::vector<USBDeviceInfo> GetAllUsbDevices();

/\* basic info \*/

std::wstring GetDeviceDescription(HKEY hKey);

std::wstring GetDeviceManufactoring(HKEY hKey);

/\* status \*/

//void getDevicesStatus(std::vector<USBDeviceInfo> &devices);

/\* capabilities \*/

std::wstring GetDeviceCapabilities(HKEY hKey);

/\* manufactoring info \*/

std::wstring GetDeviceSerialNumber(HKEY hKey);

std::wstring ExtractUsbField(const std::wstring& compatibleIDs, const std::wregex& pattern);

std::wstring GetDeviceUsbClass(HKEY hKey);

std::wstring GetDeviceUsbSubclass(HKEY hKey);

std::wstring GetDeviceUsbProtocol(HKEY hKey);

#endif // !USB\_ALGS

**SortAlgs.h**

#ifndef MY\_ALGS\_H

#define MY\_ALGS\_H

struct ThreadArgs {

std::vector<USBDeviceInfo>\* files;

int left, mid, right;

DEV\_INFO sortType;

};

bool CompareByStringValue(const std::wstring a, const std::wstring b);

bool CompareByBoolValue(const bool a, const bool b);

void Merge(std::vector<USBDeviceInfo>& arr, int left, int mid, int right, DEV\_INFO sortType);

DWORD WINAPI MergeThreadProc(LPVOID lpParameter);

void MergeSortMultiThreaded(std::vector<USBDeviceInfo>& arr, int maxThreads, DEV\_INFO sortType);

#endif //MY\_ALGS\_H

**MyCircleButton.h**

#ifndef MY\_CIRCLE\_BUTTON\_H

#define MY\_CIRCLE\_BUTTON\_H

class MyCircleButton

{

public:

/\* funcs \*/

MyCircleButton(HWND parent, int ID,

int posX, int posY, int radius, Gdiplus::Color fillColor);

~MyCircleButton();

static bool RegisterCustomButtonClass(HINSTANCE hInstance);

static LRESULT CALLBACK StaticCircleButtonProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK CircleButtonProc(UINT message, WPARAM wParam, LPARAM lParam);

/\* fields \*/

HWND handler;

HWND hParent;

int menuID;

/\* static fields \*/

static constexpr const wchar\_t\* className = L"CustomButtonClass";

private:

/\* funcs \*/

void OnPaint();

void OnLBtnDown();

/\* fields \*/

int radius;

Gdiplus::Color fillColor;

};

#endif // MY\_CIRCLE\_BUTTON\_H

**Entry.cpp**

#include "MainApp\Headers\App\general.h"

#include "MainApp\Headers\App\App.h"

int CALLBACK wWinMain(HINSTANCE, HINSTANCE, PWSTR, int)

{

MyApp app{};

return app.Run();

}

**App.cpp**

#include "MainApp\Headers\App\General.h"

#include "MainApp\Headers\App\App.h"

#include "MainApp\Headers\Algs\SortAlgs.h"

#include "MainApp\Headers\Algs\UsbAlgs.h"

#include "MainApp\Resources\resource.h"

/\* prototypes for non class funcs \*/

void WatchRegistryKey(HKEY hKey, LPCWSTR subKey, HWND hMainApp);

/\* Startup code \*/

MyApp::MyApp()

{

try

{

this->InitWindow();

this->CreateControls();

this->InitMainList();

}

catch (const std::exception& exception)

{

std::string exceptionData = exception.what();

MessageBox(

nullptr,

std::wstring(begin(exceptionData), end(exceptionData)).c\_str(),

L"Error!",

MB\_ICONERROR | MB\_OK

);

ExitProcess(EXIT\_FAILURE);

}

}

int MyApp::Run()

{

ShowWindow(this->handler, SW\_SHOWDEFAULT);

UpdateWindow(this->handler);

MSG message{};

while (GetMessage(&message, nullptr, 0, 0))

{

TranslateMessage(&message);

DispatchMessage(&message);

}

Gdiplus::GdiplusShutdown(gdiplusToken);//free gdi+ object

return static\_cast<int>(message.wParam);

}

LRESULT MyApp::AppProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

MyApp\* pThis = NULL;

if (message == WM\_NCCREATE)

{

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

pThis = (MyApp\*)pCreate->lpCreateParams;

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

pThis->handler = hwnd;

}

else

{

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

}

if (pThis)

{

return pThis->WndProc(message, wParam, lParam);

}

else

{

return DefWindowProc(hwnd, message, wParam, lParam);

}

}

/\* Initialization \*/

void MyApp::InitWindow()

{

/\* GDI+ initialization\*/

Gdiplus::GdiplusStartup(&this->gdiplusToken, &this->gdiplusStartupInput, NULL);

/\* init main window \*/

WNDCLASS windowClass = { 0 };

windowClass.cbClsExtra = 0;

windowClass.cbWndExtra = 0;

windowClass.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(WHITE\_BRUSH));

windowClass.hCursor = LoadCursor(nullptr, IDC\_ARROW);

windowClass.hInstance = GetModuleHandle(nullptr);

windowClass.hIcon = LoadIcon(windowClass.hInstance, MAKEINTRESOURCE(IDI\_ICON1));

windowClass.lpfnWndProc = MyApp::AppProc;

windowClass.lpszClassName = MyApp::className;

windowClass.lpszMenuName = nullptr;

windowClass.style = CS\_VREDRAW | CS\_HREDRAW;

/\* register classes \*/

if (!RegisterClass(&windowClass)) // window

throw std::runtime\_error("Error! Can't register main window class");

if (!MyCircleButton::RegisterCustomButtonClass(windowClass.hInstance)) // circle button

throw std::runtime\_error("Error! Can't register circle button class");

/\* creating main window \*/

this->wndScalingCoef\_X = 0.6;

this->wndScalingCoef\_Y = 0.8;

this->handler = CreateWindowEx(

WS\_EX\_LAYERED, //for transparent d2d1 clear

MyApp::className,

MyApp::appName,

WS\_POPUP | WS\_EX\_TOPMOST, //borderless window

(SCREEN\_WIDTH \* (1 - wndScalingCoef\_X)) / 2,

(SCREEN\_HEIGHT \* (1 - wndScalingCoef\_Y)) / 2,

SCREEN\_WIDTH \* wndScalingCoef\_X,

SCREEN\_HEIGHT \* wndScalingCoef\_Y,

nullptr,

nullptr,

nullptr,

this);

SetLayeredWindowAttributes( //for transparency

this->handler,

RGB(0, 0, 0),

0,

LWA\_COLORKEY);

if (!this->handler)

throw std::runtime\_error("Error! Can't create main window");

}

void MyApp::CreateControls()

{

RECT clientRect;

GetClientRect(handler, &clientRect);

/\* top menu buttons \*/

Gdiplus::Color darkRedColor(245, 66, 108);

Gdiplus::Color darkYellowColor(217, 201, 63);

Gdiplus::Color darkGreenColor(107, 217, 52);

closeButton = new MyCircleButton(

this->handler,

(int)MENU\_ID::CLOSE\_BUTTON,

clientRect.right - (TOP\_MENU\_BTN\_RADIUS + TOP\_MENU\_BTN\_GAP),

TOP\_MENU\_BTN\_Y,

TOP\_MENU\_BTN\_RADIUS,

darkRedColor);

minimizeButton = new MyCircleButton(

this->handler,

(int)MENU\_ID::MINIMIZE\_BUTTON,

clientRect.right - (TOP\_MENU\_BTN\_RADIUS + TOP\_MENU\_BTN\_GAP) \* 2,

TOP\_MENU\_BTN\_Y,

TOP\_MENU\_BTN\_RADIUS,

darkYellowColor);

maximizeButton = new MyCircleButton(

this->handler,

(int)MENU\_ID::MAXIMIZE\_BUTTON,

clientRect.right - (TOP\_MENU\_BTN\_RADIUS + TOP\_MENU\_BTN\_GAP) \* 3,

TOP\_MENU\_BTN\_Y,

TOP\_MENU\_BTN\_RADIUS,

darkGreenColor);

/\* main list \*/

hMainList = CreateWindow(

WC\_LISTVIEW,

L"",

WS\_CHILD | WS\_VISIBLE | LVS\_REPORT,

LIST\_MARGIN,

HEADER\_HEIGHT + LIST\_MARGIN,

clientRect.right - clientRect.left - LIST\_MARGIN \* 2,

clientRect.bottom - clientRect.top - (HEADER\_HEIGHT + LIST\_MARGIN) \* 2,

handler,

(HMENU)MENU\_ID::MAIN\_LIST,

nullptr,

nullptr);

}

void MyApp::InitMainList()

{

RECT clientRect;

GetClientRect(handler, &clientRect);

/\* init list columns \*/

int colWidth = (clientRect.right - clientRect.top) / NUM\_OF\_COLS;

LVCOLUMN column;

column.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

for (int iCol = 0; iCol < NUM\_OF\_COLS; iCol++)

{

column.iSubItem = iCol;

column.pszText = (wchar\_t\*)columnHeaders[iCol];

column.cx = colWidth;

SendMessage(hMainList, LVM\_INSERTCOLUMN, (WPARAM)iCol, (LPARAM)&column);

}

/\* add items to the list \*/

this->deviceInfos = GetAllUsbDevices();

ListView\_DeleteAllItems(hMainList);

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

/\* Logic \*/

//windows register

void WatchRegistryKey(HKEY hKey, LPCWSTR subKey, HWND hMainApp)

{

HKEY key;

HANDLE hChange;

if (RegOpenKeyEx(hKey, subKey, 0, KEY\_NOTIFY, &key) != ERROR\_SUCCESS)

{

std::wcerr << L"Failed to open registry key." << std::endl;

return;

}

hChange = CreateEvent(NULL, TRUE, FALSE, NULL);

if (hChange == NULL)

{

std::wcerr << L"Failed to create event." << std::endl;

RegCloseKey(key);

return;

}

while (true)

{

if (RegNotifyChangeKeyValue(key, TRUE, REG\_NOTIFY\_CHANGE\_NAME | REG\_NOTIFY\_CHANGE\_LAST\_SET, hChange, TRUE) != ERROR\_SUCCESS)

{

std::wcerr << L"Failed to set registry notification." << std::endl;

break;

}

DWORD result = WaitForSingleObject(hChange, 1000);

if (result == WAIT\_OBJECT\_0)

{

SendMessage(hMainApp, WM\_DEVICECHANGE, NULL, NULL);

}

else if (result == WAIT\_TIMEOUT)

{

//do nothing or something

}

else

{

std::wcerr << L"Error in waiting for registry change." << std::endl;

break;

}

}

CloseHandle(hChange);

RegCloseKey(key);

}

//main window

LRESULT MyApp::WndProc(UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

return OnCreate();

case WM\_DESTROY:

OnDestroy();

return 0;

case WM\_PAINT:

OnPaint();

return 0;

case WM\_SIZE:

OnSize();

return 0;

case WM\_LBUTTONDOWN:

OnLBtnDown();

return 0;

case WM\_COMMAND:

OnCommand(wParam);

return 0;

case WM\_DEVICECHANGE:

OnDeviceChange();

return 0;

case WM\_NOTIFY:

OnNotify(lParam);

return 0;

default:

return DefWindowProc(handler, message, wParam, lParam);

}

}

HRESULT MyApp::CreateGraphicsResources()

{

HRESULT hResult = S\_OK;

if (NULL == pRenderTarget)

{

/\* window \*/

RECT clientRect;

GetClientRect(handler, &clientRect);

D2D1\_SIZE\_U size = D2D1::SizeU(

clientRect.right - clientRect.left,

clientRect.bottom - clientRect.top);

/\* render \*/

hResult = pFactory->CreateHwndRenderTarget(

D2D1::RenderTargetProperties(

D2D1\_RENDER\_TARGET\_TYPE\_DEFAULT,

D2D1::PixelFormat(DXGI\_FORMAT\_UNKNOWN, D2D1\_ALPHA\_MODE\_PREMULTIPLIED)),

D2D1::HwndRenderTargetProperties(handler, size),

&pRenderTarget);

if (SUCCEEDED(hResult))

{

/\* background brush \*/

ID2D1GradientStopCollection\* pGradientStops = NULL;

D2D1\_GRADIENT\_STOP gradientStops[2];

gradientStops[0].color = D2D1::ColorF(D2D1::ColorF::CornflowerBlue, 1);

gradientStops[0].position = 0.0f;

gradientStops[1].color = D2D1::ColorF(D2D1::ColorF::DodgerBlue, 1);

gradientStops[1].position = 1.0f;

hResult = pRenderTarget->CreateGradientStopCollection(

gradientStops,

2,

D2D1\_GAMMA\_2\_2,

D2D1\_EXTEND\_MODE\_CLAMP,

&pGradientStops);

if (SUCCEEDED(hResult))

{

hResult = pRenderTarget->CreateLinearGradientBrush(

D2D1::LinearGradientBrushProperties(

D2D1::Point2F(clientRect.left, clientRect.top),

D2D1::Point2F(clientRect.right, clientRect.bottom)),

pGradientStops,

&pBackgroudnGradientBrush);

}

/\* border brush \*/

pGradientStops = NULL;

gradientStops[0].color = D2D1::ColorF(D2D1::ColorF::Blue, 1);

gradientStops[0].position = 0.0f;

gradientStops[1].color = D2D1::ColorF(D2D1::ColorF::Violet, 1);

gradientStops[1].position = 1.0f;

hResult = pRenderTarget->CreateGradientStopCollection(

gradientStops,

2,

D2D1\_GAMMA\_2\_2,

D2D1\_EXTEND\_MODE\_CLAMP,

&pGradientStops);

if (SUCCEEDED(hResult))

{

hResult = pRenderTarget->CreateLinearGradientBrush(

D2D1::LinearGradientBrushProperties(

D2D1::Point2F(clientRect.left, clientRect.top),

D2D1::Point2F(clientRect.right, clientRect.bottom)),

pGradientStops,

&pWindowBorderBrush);

}

/\* header text brush \*/

pRenderTarget->CreateSolidColorBrush(

D2D1::ColorF(D2D1::ColorF::White),

&pHeaderTextBrush);

}

}

return hResult;

}

int MyApp::OnCreate()

{

if (FAILED(D2D1CreateFactory(

D2D1\_FACTORY\_TYPE\_SINGLE\_THREADED,

&pFactory)))

{

return -1; // Fail CreateWindowEx.

}

if (FAILED(DWriteCreateFactory(

DWRITE\_FACTORY\_TYPE\_SHARED,

\_\_uuidof(IDWriteFactory),

reinterpret\_cast<IUnknown\*\*>(&pDWriteFactory))))

{

return -1;

}

isMaximized = false;

numOfDevices = 0;

std::thread registryWatcher(WatchRegistryKey, HKEY\_LOCAL\_MACHINE, L"SYSTEM\\CurrentControlSet\\Enum", handler);

registryWatcher.detach();

return 0;

}

void MyApp::OnPaint()

{

HRESULT hResult = CreateGraphicsResources();

if (SUCCEEDED(hResult))

{

/\* prepare for drawing \*/

PAINTSTRUCT paintStruct;

BeginPaint(this->handler, &paintStruct);

HDC hDevContext = GetWindowDC(handler);

pRenderTarget->BeginDraw();

pRenderTarget->Clear();

D2D1\_SIZE\_F clientSizes = pRenderTarget->GetSize();

/\* draw window backgorund and border \*/

if (isMaximized)

{

D2D1\_RECT\_F windowRect = D2D1::RectF(0, 0, clientSizes.width, clientSizes.height);

pRenderTarget->FillRectangle(&windowRect, pBackgroudnGradientBrush);

pRenderTarget->DrawRectangle(&windowRect, pWindowBorderBrush);

}

else

{

D2D1\_ROUNDED\_RECT roundedWindowRect = D2D1::RoundedRect(

D2D1::RectF(0, 0, clientSizes.width, clientSizes.height),

30.f,

30.f

);

pRenderTarget->FillRoundedRectangle(&roundedWindowRect, pBackgroudnGradientBrush);

pRenderTarget->DrawRoundedRectangle(&roundedWindowRect, pWindowBorderBrush, 2.0f, NULL);

}

/\* draw text \*/

//title text

IDWriteTextFormat\* pTextFormat = nullptr;

pDWriteFactory->CreateTextFormat(

L"JetBrains Mono", nullptr,

DWRITE\_FONT\_WEIGHT\_BOLD,

DWRITE\_FONT\_STYLE\_NORMAL,

DWRITE\_FONT\_STRETCH\_MEDIUM,

24.0f, L"en-US", &pTextFormat);

pRenderTarget->DrawText(

L"USB Monitor",

wcslen(L"USB Monitor"),

pTextFormat,

D2D1::RectF(20, 10, clientSizes.width, clientSizes.height),

pHeaderTextBrush);

pTextFormat->Release();

//bottom text

std::wstring textToPrint = std::format(L"Количество найденных USB-устройств: {} ", numOfDevices).c\_str();

pDWriteFactory->CreateTextFormat(

L"JetBrains Mono", nullptr,

DWRITE\_FONT\_WEIGHT\_BOLD,

DWRITE\_FONT\_STYLE\_NORMAL,

DWRITE\_FONT\_STRETCH\_MEDIUM,

18.0f, L"en-US", &pTextFormat);

pRenderTarget->DrawText(

textToPrint.c\_str(),

textToPrint.length(),

pTextFormat,

D2D1::RectF(20, clientSizes.height - 18 - 20, clientSizes.width, clientSizes.height),

pHeaderTextBrush);

pTextFormat->Release();

/\* end drawing\*/

hResult = pRenderTarget->EndDraw();

if (FAILED(hResult) || hResult == D2DERR\_RECREATE\_TARGET)

{

DiscardGraphicsResources();

}

EndPaint(this->handler, &paintStruct);

}

}

void MyApp::OnSize()

{

if (pRenderTarget != NULL)

{

/\* preparing for resize \*/

isMaximized = !isMaximized;

RECT clientRect;

GetClientRect(handler, &clientRect);

D2D1\_SIZE\_U size = D2D1::SizeU(clientRect.right, clientRect.bottom);

/\* buttons positions \*/

SetWindowPos(

closeButton->handler,

HWND\_TOPMOST,

clientRect.right - (TOP\_MENU\_BTN\_RADIUS + TOP\_MENU\_BTN\_GAP),

TOP\_MENU\_BTN\_Y,

0,

0,

SWP\_NOSIZE | SWP\_NOZORDER);

SetWindowPos(

minimizeButton->handler,

HWND\_TOPMOST,

clientRect.right - (TOP\_MENU\_BTN\_RADIUS + TOP\_MENU\_BTN\_GAP) \* 2,

TOP\_MENU\_BTN\_Y,

0,

0,

SWP\_NOSIZE | SWP\_NOZORDER);

SetWindowPos(

maximizeButton->handler,

HWND\_TOPMOST,

clientRect.right - (TOP\_MENU\_BTN\_RADIUS + TOP\_MENU\_BTN\_GAP) \* 3,

TOP\_MENU\_BTN\_Y,

0,

0,

SWP\_NOSIZE | SWP\_NOZORDER);

/\* main list \*/

SetWindowPos(

hMainList,

HWND\_TOPMOST,

LIST\_MARGIN,

HEADER\_HEIGHT + LIST\_MARGIN,

clientRect.right - clientRect.left - LIST\_MARGIN \* 2,

clientRect.bottom - clientRect.top - (HEADER\_HEIGHT + LIST\_MARGIN) \* 2,

SWP\_NOZORDER);

pRenderTarget->Resize(size);

InvalidateRect(handler, NULL, TRUE);

}

}

void MyApp::OnLBtnDown()

{

SendMessage(handler, WM\_SYSCOMMAND, SC\_MOVE | HTCAPTION, 0);

}

void MyApp::OnCommand(WPARAM menuId)

{

switch (LOWORD(menuId))

{

case (int)MENU\_ID::CLOSE\_BUTTON:

{

DestroyWindow(handler);

}

break;

case (int)MENU\_ID::MINIMIZE\_BUTTON:

{

ShowWindow(handler, SW\_MINIMIZE);

}

break;

case (int)MENU\_ID::MAXIMIZE\_BUTTON:

{

isMaximized ? ShowWindow(handler, SW\_NORMAL) : ShowWindow(handler, SW\_MAXIMIZE);

}

break;

}

}

void MyApp::OnDestroy()

{

DiscardGraphicsResources();

DiscradControls();

SafeRelease(&pFactory);

SafeRelease(&pDWriteFactory);

PostQuitMessage(0);

}

void MyApp::OnDeviceChange()

{

deviceInfos = GetAllUsbDevices();

ListView\_DeleteAllItems(hMainList);

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

void MyApp::OnNotify(LPARAM lParam)

{

LPNMHDR pnmh = (LPNMHDR)lParam;

if (pnmh->hwndFrom == ListView\_GetHeader(hMainList))

{

switch (pnmh->code)

{

case HDN\_ITEMCLICK:

{

LPNMHEADER phdr = (LPNMHEADER)lParam;

int clickedColumn = phdr->iItem;

MergeSortMultiThreaded(deviceInfos, MAX\_THREADS, (DEV\_INFO)clickedColumn);

ListView\_DeleteAllItems(hMainList);

FillMainList();

UpdateColumnsWidths();

InvalidateRect(handler, NULL, TRUE);

}

break;

}

}

}

//list

void MyApp::FillMainList()

{

/\* init items \*/

numOfDevices = deviceInfos.size();

int deviceIndex = 0;

for (USBDeviceInfo device : deviceInfos)

{

const wchar\_t\*\* descriptions = new const wchar\_t\* [NUM\_OF\_COLS];

//info

descriptions[DEV\_INFO::DESC] = device.description.c\_str();

descriptions[DEV\_INFO::MFG] = device.manufactoring.c\_str();

descriptions[DEV\_INFO::SERVICE] = device.serviceName.c\_str();

//status

descriptions[DEV\_INFO::IS\_CONNECTED] = device.isConnected ? L"Да" : L"Нет";

descriptions[DEV\_INFO::IS\_DIASBLED] = device.isDisabled ? L"Да" : L"Нет";

descriptions[DEV\_INFO::IS\_SAFE\_TO\_UNPLUG] = device.isSafeToUnplug ? L"Да" : L"Нет";

//capabilities

descriptions[DEV\_INFO::IS\_USB\_HUB] = device.isUsbHub ? L"Да" : L"Нет";

descriptions[DEV\_INFO::CAPABILITIES] = device.capabilities.c\_str();

//location

descriptions[DEV\_INFO::LOCATION] = device.location.c\_str();

//addintional info

descriptions[DEV\_INFO::SERIAL\_NUM] = device.serialNumber.c\_str();

descriptions[DEV\_INFO::USB\_CLASS] = device.usbClass.c\_str();

descriptions[DEV\_INFO::USB\_SUBCLASS] = device.usbSubclass.c\_str();

descriptions[DEV\_INFO::USB\_PROTOCOL] = device.usbProtocol.c\_str();

//ids

descriptions[DEV\_INFO::ID] = device.id.c\_str();

descriptions[DEV\_INFO::DEV\_GUID] = device.devClassGUID.c\_str();

descriptions[DEV\_INFO::HARDWARE\_ID] = device.hardwareID.c\_str();

descriptions[DEV\_INFO::PARENT\_PREFIX] = device.parentIDPrefix.c\_str();

InsertItemWithSubItems(deviceIndex, descriptions, NUM\_OF\_COLS);

deviceIndex++;

}

}

void MyApp::SetSubItems(int itemIndex, const wchar\_t\*\* texts, int count) {

LVITEM item;

item.mask = LVIF\_TEXT;

item.iItem = itemIndex;

for (int i = 1; i < count; i++) { // Начинаем с 1, так как 0 уже установлен при вставке элемента

item.iSubItem = i;

item.pszText = (wchar\_t\*)texts[i];

ListView\_SetItem(hMainList, &item);

}

}

void MyApp::InsertItemWithSubItems(int itemIndex, const wchar\_t\*\* texts, int count) {

LVITEM item;

item.mask = LVIF\_TEXT;

item.iItem = itemIndex;

item.iSubItem = 0;

item.pszText = (wchar\_t\*)texts[0];

int newItemIndex = ListView\_InsertItem(hMainList, &item);

if (newItemIndex != -1) {

SetSubItems(newItemIndex, texts, count);

}

else {

MessageBox(

NULL,

L"Failed to insert item into ListView",

L"Error",

MB\_OK | MB\_ICONERROR);

}

}

void MyApp::UpdateColumnsWidths()

{

for (int iCol = 0; iCol < NUM\_OF\_COLS; iCol++)

{

int maxWidth = 0;

int textWidth = 0;

if (DEV\_INFO::IS\_CONNECTED == iCol ||

DEV\_INFO::IS\_DIASBLED == iCol ||

DEV\_INFO::IS\_SAFE\_TO\_UNPLUG == iCol ||

DEV\_INFO::IS\_USB\_HUB == iCol ||

DEV\_INFO::USB\_CLASS == iCol ||

DEV\_INFO::USB\_SUBCLASS == iCol ||

DEV\_INFO::USB\_PROTOCOL == iCol)

{

maxWidth = 30;

}

else

for (USBDeviceInfo device : deviceInfos)

{

switch (iCol)

{

case DEV\_INFO::DESC: textWidth = ListView\_GetStringWidth(hMainList, device.description.c\_str()); break;

case DEV\_INFO::MFG: textWidth = ListView\_GetStringWidth(hMainList, device.manufactoring.c\_str()); break;

case DEV\_INFO::SERVICE: textWidth = ListView\_GetStringWidth(hMainList, device.serviceName.c\_str()); break;

case DEV\_INFO::CAPABILITIES: textWidth = ListView\_GetStringWidth(hMainList, device.capabilities.c\_str()); break;

case DEV\_INFO::LOCATION: textWidth = ListView\_GetStringWidth(hMainList, device.location.c\_str()); break;

case DEV\_INFO::SERIAL\_NUM: textWidth = ListView\_GetStringWidth(hMainList, device.serialNumber.c\_str()); break;

case DEV\_INFO::ID: textWidth = ListView\_GetStringWidth(hMainList, device.id.c\_str()); break;

case DEV\_INFO::DEV\_GUID: textWidth = ListView\_GetStringWidth(hMainList, device.devClassGUID.c\_str()); break;

case DEV\_INFO::HARDWARE\_ID: textWidth = ListView\_GetStringWidth(hMainList, device.hardwareID.c\_str()); break;

case DEV\_INFO::PARENT\_PREFIX: textWidth = ListView\_GetStringWidth(hMainList, device.parentIDPrefix.c\_str()); break;

}

if (textWidth > maxWidth)

maxWidth = textWidth;

}

int columnHeaderLen = ListView\_GetStringWidth(hMainList, columnHeaders[iCol]);

if (columnHeaderLen > maxWidth)

maxWidth = columnHeaderLen + 20;

ListView\_SetColumnWidth(hMainList, iCol, maxWidth + 20);

}

}

/\* Finalization \*/

MyApp::~MyApp()

{

}

void MyApp::DiscradControls()

{

if (closeButton)

delete closeButton;

if (minimizeButton)

delete minimizeButton;

if (maximizeButton)

delete maximizeButton;

}

void MyApp::DiscardGraphicsResources()

{

SafeRelease(&pHeaderTextBrush);

SafeRelease(&pBackgroudnGradientBrush);

SafeRelease(&pWindowBorderBrush);

SafeRelease(&pRenderTarget);

}

**UsbAlgs.cpp**

#include "MainApp/Headers/App/general.h"

#include "MainApp/Headers/Algs/UsbAlgs.h"

/\* basic info \*/

std::wstring GetDeviceDescription(HKEY hKey)

{

std::wstring desc = GetRegistryStringValue(hKey, L"FriendlyName");

if (desc == L"Unknown")

{

desc = GetRegistryStringValue(hKey, L"DeviceDesc");

}

size\_t pos = desc.rfind(L';');

if (pos != std::wstring::npos && pos + 1 < desc.length()) {

return desc.substr(pos + 1);

}

return L"Unknown";

};

std::wstring GetDeviceManufactoring(HKEY hKey)

{

std::wstring manufactoring = GetRegistryStringValue(hKey, L"Mfg");

size\_t startMarker = manufactoring.find(L'%');

size\_t endMarker = manufactoring.find(L'%', startMarker + 1);

if (startMarker != std::wstring::npos && endMarker != std::wstring::npos)

{

size\_t semicolonPos = manufactoring.find(L';', endMarker + 1);

if (semicolonPos != std::wstring::npos)

{

size\_t bracketPos = manufactoring.find(L'(', endMarker + 1);

if (bracketPos != std::wstring::npos)

{

endMarker = manufactoring.find(L')', bracketPos + 1);

return manufactoring.substr(bracketPos + 1, endMarker - (bracketPos + 1));

}

return manufactoring.substr(semicolonPos + 1);

}

}

return manufactoring;

}

/\* status \*/

void getDevicesStatus(std::vector<USBDeviceInfo> &devices)

{

HDEVINFO deviceInfoSet;

SP\_DEVINFO\_DATA deviceInfoData;

DWORD index = 0;

deviceInfoSet = SetupDiGetClassDevs(NULL, L"USB", NULL, DIGCF\_PRESENT | DIGCF\_ALLCLASSES);

if (deviceInfoSet == INVALID\_HANDLE\_VALUE)

{

std::cerr << "Can't get USB devices list" << std::endl;

}

deviceInfoData.cbSize = sizeof(SP\_DEVINFO\_DATA);

while (SetupDiEnumDeviceInfo(

deviceInfoSet,

index,

&deviceInfoData))

{

index++;

TCHAR deviceName[256];

DWORD status, problemCode;

for (int i = 0; i < devices.size(); i++)

{

GUID devGuid;

HRESULT hResult = CLSIDFromString(devices[i].devClassGUID.c\_str(), &devGuid);

if(SUCCEEDED(hResult) && deviceInfoData.ClassGuid == devGuid)

if (CM\_Get\_DevNode\_Status(

&status,

&problemCode,

deviceInfoData.DevInst,

0) == CR\_SUCCESS)

{

devices[i].isConnected = !(status & DN\_HAS\_PROBLEM);

devices[i].isDisabled = status & DN\_DRIVER\_BLOCKED;

devices[i].isSafeToUnplug = status & DN\_REMOVABLE;

}

else

{

devices[i].isConnected = false;

devices[i].isDisabled = false;

devices[i].isSafeToUnplug = true;

}

}

}

SetupDiDestroyDeviceInfoList(deviceInfoSet);

}

/\* status \*/

std::wstring GetDeviceCapabilities(HKEY hKey)

{

const DWORD MASK\_REMOVABLE = 0x04;

const DWORD MASK\_UNIQUE\_ID = 0x10;

const DWORD MASK\_SILENT\_INSTALL = 0x20;

const DWORD MASK\_RAW\_DEVICE\_OK = 0x40;

const DWORD MASK\_SURPRISE\_REMOVAL\_OK = 0x80;

std::wstring capabilities = L"";

DWORD capabilitiesDwordValue = GetRegistryDWORDValue(hKey, L"Capabilities");

if (capabilitiesDwordValue & MASK\_REMOVABLE)

capabilities += L"Removable ";

if (capabilitiesDwordValue & MASK\_UNIQUE\_ID)

capabilities += L"UniqueID ";

if (capabilitiesDwordValue & MASK\_SILENT\_INSTALL)

capabilities += L"SilentInstall ";

if (capabilitiesDwordValue & MASK\_RAW\_DEVICE\_OK)

capabilities += L"RawDeviceOK ";

if (capabilitiesDwordValue & MASK\_SURPRISE\_REMOVAL\_OK)

capabilities += L"SurpriseRemovalOK ";

if (capabilities.empty())

capabilities = L"Unknown";

return capabilities;

}

/\* manufactoring info \*/

std::wstring GetDeviceSerialNumber(HKEY hKey)

{

std::wstring name = L"Device Parameters";

HKEY devParamsKey;

if (RegOpenKeyExW(

hKey,

name.c\_str(),

0,

KEY\_READ,

&devParamsKey) == ERROR\_SUCCESS)

{

const std::wstring symbolicName = GetRegistryStringValue(devParamsKey, L"SymbolicName");

std::wregex re(LR"(USB#VID\_[0-9A-Fa-f]+&PID\_[0-9A-Fa-f]+#([0-9A-Fa-f]+)#)");

std::wsmatch match;

if (std::regex\_search(symbolicName, match, re) && match.size() > 1)

{

return match[1].str();

}

}

return L"";

}

std::wstring ExtractUsbField(const std::wstring& compatibleIDs, const std::wregex& pattern)

{

std::wsmatch match;

if (std::regex\_search(compatibleIDs, match, pattern) && match.size() > 1)

{

return match.str(1);

}

return L"00"; //if not found

}

std::wstring GetDeviceUsbClass(HKEY hKey)

{

std::wstring compatibleIDs = GetRegistryStringValue(hKey, L"CompatibleIDs");

std::wregex classPattern(LR"(USB\\Class\_([0-9A-Fa-f]{2}))");

return ExtractUsbField(compatibleIDs, classPattern);

}

std::wstring GetDeviceUsbSubclass(HKEY hKey)

{

std::wstring compatibleIDs = GetRegistryStringValue(hKey, L"CompatibleIDs");

std::wregex subclassPattern(LR"(USB\\Class\_[0-9A-Fa-f]{2}&SubClass\_([0-9A-Fa-f]{2}))");

return ExtractUsbField(compatibleIDs, subclassPattern);

}

std::wstring GetDeviceUsbProtocol(HKEY hKey)

{

std::wstring compatibleIDs = GetRegistryStringValue(hKey, L"CompatibleIDs");

std::wregex protocolPattern(LR"(USB\\Class\_[0-9A-Fa-f]{2}&SubClass\_[0-9A-Fa-f]{2}&Prot\_([0-9A-Fa-f]{2}))");

return ExtractUsbField(compatibleIDs, protocolPattern);

}

/\* common \*/

std::wstring GetRegistryStringValue(HKEY hKey, const std::wstring& valueName)

{

DWORD type;

wchar\_t value[256];

DWORD valueSize = sizeof(value);

if (RegQueryValueExW(

hKey,

valueName.c\_str(),

NULL,

&type,

(LPBYTE)value,

&valueSize) == ERROR\_SUCCESS)

{

return value;

}

return L"Unknown";

}

DWORD GetRegistryDWORDValue(HKEY hKey, const std::wstring& valueName)

{

DWORD data;

DWORD dataSize = sizeof(data);

DWORD dataType;

LONG result = RegQueryValueEx(

hKey,

valueName.c\_str(),

nullptr,

&dataType,

reinterpret\_cast<LPBYTE>(&data),

&dataSize);

if (result != ERROR\_SUCCESS)

{

std::wcerr << L"Failed to read registry value: " << valueName << L" (Error: " << result << L")" << std::endl;

return -1;

}

if (dataType != REG\_DWORD) {

std::wcerr << L"Unexpected data type for value: " << valueName << L" (Type: " << dataType << L")" << std::endl;

return -1;

}

return data;

}

std::vector<USBDeviceInfo> ListUSBDevices(const std::wstring& registryPath)

{

std::vector<USBDeviceInfo> devices;

HKEY hKey;

if (RegOpenKeyExW(

HKEY\_LOCAL\_MACHINE,

registryPath.c\_str(),

0,

KEY\_READ,

&hKey) != ERROR\_SUCCESS)

{

std::cerr << "Error: Unable to open registry key." << std::endl;

return devices;

}

DWORD index = 0;

DWORD nameSize = 256;

wchar\_t name[256];

while (RegEnumKeyExW(

hKey,

index,

name,

&nameSize,

NULL, NULL, NULL, NULL) == ERROR\_SUCCESS)

{

std::wstring deviceKeyPath = registryPath + L"\\" + name;

HKEY deviceKey;

if (RegOpenKeyExW(hKey, name, 0, KEY\_READ, &deviceKey) == ERROR\_SUCCESS)

{

DWORD subIndex = 0;

DWORD subNameSize = 256;

wchar\_t subName[256];

while (RegEnumKeyExW(

deviceKey,

subIndex,

subName,

&subNameSize,

NULL, NULL, NULL, NULL) == ERROR\_SUCCESS)

{

std::wstring instanceKeyPath = deviceKeyPath + L"\\" + subName;

HKEY instanceKey;

if (RegOpenKeyExW(

deviceKey,

subName,

0,

KEY\_READ,

&instanceKey) == ERROR\_SUCCESS)

{

USBDeviceInfo deviceInfo;

/\* basic info \*/

deviceInfo.description = GetDeviceDescription(instanceKey);

deviceInfo.manufactoring = GetDeviceManufactoring(instanceKey);

deviceInfo.serviceName = GetRegistryStringValue(instanceKey, L"Service");

/\* capabilities \*/

deviceInfo.isUsbHub = deviceInfo.serviceName == L"USBHUB3";

deviceInfo.capabilities = GetDeviceCapabilities(instanceKey);

/\* preparation for status gathering \*/

deviceInfo.isConnected = false;

deviceInfo.isDisabled = false;

deviceInfo.isSafeToUnplug = true;

/\* location \*/

deviceInfo.location = GetRegistryStringValue(instanceKey, L"LocationInformation");

/\* manufactoring info \*/

deviceInfo.serialNumber = GetDeviceSerialNumber(instanceKey);

deviceInfo.usbClass = GetDeviceUsbClass(instanceKey);

deviceInfo.usbSubclass = GetDeviceUsbSubclass(instanceKey);

deviceInfo.usbProtocol = GetDeviceUsbProtocol(instanceKey);

/\* IDS \*/

deviceInfo.id = std::wstring(name) + L"\\" + subName;

deviceInfo.hardwareID = GetRegistryStringValue(instanceKey, L"HardwareID");

deviceInfo.devClassGUID = GetRegistryStringValue(instanceKey, L"ClassGUID");

deviceInfo.parentIDPrefix = GetRegistryStringValue(instanceKey, L"ParentIdPrefix");

devices.push\_back(deviceInfo);

RegCloseKey(instanceKey);

}

subIndex++;

subNameSize = 256;

}

RegCloseKey(deviceKey);

}

index++;

nameSize = 256;

}

RegCloseKey(hKey);

/\* status \*/

getDevicesStatus(devices);

return devices;

}

std::vector<USBDeviceInfo> GetAllUsbDevices()

{

std::vector<USBDeviceInfo> usbDevices = ListUSBDevices(L"SYSTEM\\CurrentControlSet\\Enum\\USB");

std::vector<USBDeviceInfo> usbPrintDevices = ListUSBDevices(L"SYSTEM\\CurrentControlSet\\Enum\\USBPRINT");

std::vector<USBDeviceInfo> usbStorDevices = ListUSBDevices(L"SYSTEM\\CurrentControlSet\\Enum\\USBSTOR");

usbDevices.insert(usbDevices.end(), usbPrintDevices.begin(), usbPrintDevices.end());

usbDevices.insert(usbDevices.end(), usbStorDevices.begin(), usbStorDevices.end());

return usbDevices;

}

**myCircleButton.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "MainApp\Headers\App\general.h"

#include "MainApp\Headers\Components\myCircleButton.h"

/\* Startup code \*/

bool MyCircleButton::RegisterCustomButtonClass(HINSTANCE hInstance)

{

WNDCLASS windowClass = {};

windowClass.lpfnWndProc = StaticCircleButtonProc;

windowClass.hInstance = hInstance;

windowClass.lpszClassName = MyCircleButton::className;

return RegisterClass(&windowClass);

}

LRESULT CALLBACK MyCircleButton::StaticCircleButtonProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

MyCircleButton\* pThis = NULL;

if (uMsg == WM\_CREATE)

{

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

pThis = (MyCircleButton\*)pCreate->lpCreateParams;

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

pThis->handler = hwnd;

}

else

{

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

}

if (pThis)

{

return pThis->CircleButtonProc(uMsg, wParam, lParam);

}

else

{

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

}

}

/\* Initialization \*/

MyCircleButton::MyCircleButton(HWND parent, int ID, int posX, int posY, int radius, Gdiplus::Color fillColor)

{

this->hParent = parent;

this->menuID = ID;

this->radius = radius;

this->fillColor = fillColor;

this->handler = CreateWindow(

MyCircleButton::className, NULL,

WS\_CHILD | WS\_VISIBLE,

posX, posY, radius, radius,

parent, NULL, NULL, this

);

if (!this->handler)

throw std::runtime\_error("Error! Can't create circle button window");

}

/\* Logic \*/

LRESULT CALLBACK MyCircleButton::CircleButtonProc(UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_PAINT:

OnPaint();

return 0;

case WM\_LBUTTONDOWN:

OnLBtnDown();

return 0;

default:

return DefWindowProc(handler, uMsg, wParam, lParam);

}

}

void MyCircleButton::OnPaint()

{

/\* prepare for drawing \*/

PAINTSTRUCT paintStruct;

HDC hDevContext = BeginPaint(this->handler, &paintStruct);

RECT clientRect;

GetClientRect(handler, &clientRect);

Gdiplus::Graphics graphics(hDevContext);

Gdiplus::SolidBrush brush(fillColor);

/\* draw button \*/

graphics.FillEllipse(&brush,

static\_cast<int>(clientRect.left),

static\_cast<int>(clientRect.top),

static\_cast<int>(clientRect.right - clientRect.left),

static\_cast<int>(clientRect.bottom - clientRect.top));

/\* end drawing\*/

EndPaint(this->handler, &paintStruct);

}

void MyCircleButton::OnLBtnDown()

{

SendMessage(hParent, WM\_COMMAND, MAKEWPARAM(menuID, BN\_CLICKED), (LPARAM)handler);

}

/\* Finalization \*/

MyCircleButton::~MyCircleButton()

{

}