МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра САПР

**Курсовая работа**

**по дисциплине «Операционные системы и оболочки»**

**Тема: «Изучение интерфейса прикладного программирования ОС Windows. Работа с окнами»**

Направление подготовки – 02.03.03 Математическое обеспечение и администрирование информационных систем

Профиль подготовки – Администрирование информационных систем

Выполнили студенты: Ишенбеков Н.М.

Женишбек кызы Э.

Группа: 21ВА1

Руководитель:

к.т.н., доцент Гудков А.А.

2024

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Пояснительная записка содержит 29 страницы, 7 рисунков, 3 источника и 2 приложения.  ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ПРОГРАММНЫЙ ПРОДУКТ РАБОТА С ОКНАМИ, ОПЕРАЦИОННАЯ СИСТЕМА  Объектом разработки является программный продукт для работы с окнами.  Задачами, которые требовалось решить для достижения цели, являются: изучение предметной области, возможностей программных средств, проектирование и разработка программы.  Результатом работы является разработанный программный продукт для работы с окнами. | | | | | | | | | | |
|  |  |  |  |  | КП.020303.04 22 81 01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист. | № докум. | Подпись | Дата |
| Разраб. | | Ишенбеков Н.М.  Женишбек к Э. |  |  | *Изучение интерфейса прикладного программирования ОС Windows. Работа с окнами*  *Пояснительная записка* | Лит. | | | Лист | Листов |
| Провер. | | Гудков А.А. |  |  |  |  |  | 1 | 1 |
| Реценз. | |  |  |  | ФВТ группа 21ВА1 | | | | |
| Н. Контр. | |  |  |  |
| Утверд. | |  |  |  |

Оглавление

[1. Теоретические сведения 4](#_Toc167540708)

[2. Описание программы 6](#_Toc167540709)

[2.1 Общие сведения 6](#_Toc167540710)

[2.2 Структура программы 7](#_Toc167540711)

[2.3 Описание разработанных функций 8](#_Toc167540712)

[2.4 Описание используемых функций WinApi 9](#_Toc167540713)

[2.5 Схемы основных алгоритмов 15](#_Toc167540714)

[Заключение 17](#_Toc167540715)

[Список используемой литературы 18](#_Toc167540716)

[Приложение А 19](#_Toc167540717)

[Приложение Б 28](#_Toc167540718)

# Теоретические сведения

Окна являются основным элементом графического интерфейса пользователя (GUI) в операционной системе Windows. Они позволяют пользователям взаимодействовать с программами посредством визуальных элементов, таких как кнопки, меню, поля ввода и другие компоненты. В Windows окна управляются с помощью системы оконного менеджмента, которая отвечает за создание, отображение, обновление и удаление окон.

Вот некоторые ключевые понятия, которые могут быть полезны при изучении работы с окнами в Windows:

Окно — это прямоугольная область экрана, используемая для отображения информации и взаимодействия с пользователем. Окна могут быть различных типов: основные окна, диалоговые окна, дочерние окна и всплывающие окна.

Основные компоненты окна:

Заголовок окна — верхняя часть окна, содержащая название и кнопки управления (свернуть, развернуть, закрыть).

Клиентская область — внутренняя часть окна, где отображается основной контент приложения.

Рамка окна — границы окна, которые могут быть использованы для изменения его размеров.

Меню окна — обычно расположено в верхней части окна и предоставляет доступ к различным функциям и настройкам приложения.

Создание окна в Windows осуществляется с помощью функции CreateWindow или CreateWindowEx. Эти функции принимают множество параметров, таких как класс окна, заголовок, стиль окна, размеры и положение окна на экране.

Каждая программа в Windows имеет цикл сообщений, который обрабатывает различные события, такие как нажатие клавиш, перемещение мыши, рисование и другие взаимодействия пользователя [1]. Функция GetMessage извлекает сообщения из очереди сообщений, а функция DispatchMessage отправляет их соответствующему оконному процессу.

Оконная процедура (window procedure) — это функция, которая обрабатывает все сообщения, отправленные окну. Она определяется пользователем и регистрируется при создании класса окна.

Перед созданием окна необходимо зарегистрировать класс окна, который определяет основные характеристики окна, такие как его стиль и указывает оконную процедуру.

Windows предоставляет множество функций для управления окнами, таких как изменение размеров, перемещение, скрытие и отображение окон. Некоторые из этих функций включают ShowWindow, MoveWindow, SetWindowPos и DestroyWindow.

Ваша курсовой работе должны были включить такие задачи, как создание простого приложения с окнами, обработка пользовательских событий, управление состоянием окон и взаимодействие с элементами GUI.

Работа с окнами является фундаментальной частью разработки приложений в Windows и позволяет создавать удобные и функциональные графические интерфейсы для пользователей.

# Описание программы

## Общие сведения

Программа написана на языке С++, с использованием функций API Windows по работе с окнами.

Используется шаблон классического Windows приложения Visual Studio 2022.

Программа должна реализовывать создание и уничтожение окон; перемещение, изменение размера и порядка окон; перечисление работающих в ОС окон; перечисление всех дочерних окон для конкретного окна; поиск окон; получение информации о свойствах окна и их изменение.

Основными функциями программы являются:

1. Создание и уничтожение окон
2. Перемещение окон
3. Изменение размера и порядка окон
4. Перечисление списка работающих окон
5. Перечисление списка дочерних окон выбранного окна
6. Поиск работающих окон
7. Получение информации о свойствах окна и их изменение

## Структура программы



Рисунок 1 - Схема структуры программы

## Описание разработанных функций

Функция EnumWindowsProc заполняет список запущенными окнами.

Функция EnumChildProc заполняет список дочерних окон для выбранного окна.

Функция RefreshThreadList заполняет список потоков для выбранного процесса.

Функция CreateNewWindow создает новое окно.

Функция SearchWindows ищет запущенные окна в системе отображает его в списке.

Функция PrintWindowStyle позволяет получить стиль выбранного окна.

## Описание используемых функций WinApi

Все функции приведены из официальной документации Microsoft для WinApi [3].

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow); – точка входа в программу Windows

WNDPROC Wndproc;

LRESULT Wndproc(

HWND unnamedParam1,

UINT unnamedParam2,

WPARAM unnamedParam3,

LPARAM unnamedParam4

)

{...} – Функция обратного вызова, которую вы определяете в своем приложении, которая обрабатывает сообщения, отправляемые в окно.

WinUser.h[2]

ATOM RegisterClassEx(

[in] const WNDCLASSEXW \*unnamedParam1

); – Регистрирует класс окна для последующего использования в вызовах функции CreateWindow или CreateWindowEx .

HWND CreateWindowW(

[in, optional] lpClassName,

[in, optional] lpWindowName,

[in] dwStyle,

[in] x,

[in] y,

[in] nWidth,

[in] nHeight,

[in, optional] hWndParent,

[in, optional] hMenu,

[in, optional] hInstance,

[in, optional] lpParam

); – Создает перекрывающееся, всплывающее или дочернее окно. В нем указывается класс окна, заголовок окна, стиль окна и (необязательно) начальное положение и размер окна. Функция также указывает родительский элемент или владельца окна, если таковой имеется, и меню окна.

BOOL ShowWindow(

[in] HWND hWnd,

[in] int nCmdShow

); – Задает состояние отображения указанного окна.

BOOL UpdateWindow(

[in] HWND hWnd

); – Функция UpdateWindow обновляет клиентскую область указанного окна, отправляя WM\_PAINT сообщение в окно, если область обновления окна не пуста. Функция отправляет сообщение WM\_PAINT непосредственно в процедуру окна указанного окна, обходя очередь приложения. Если регион обновления пуст, сообщение не отправляется.

BOOL GetMessage(

[out] LPMSG lpMsg,

[in, optional] HWND hWnd,

[in] UINT wMsgFilterMin,

[in] UINT wMsgFilterMax

); – Извлекает сообщение из очереди сообщений вызывающего потока. Функция отправляет входящие отправленные сообщения до тех пор, пока отправленное сообщение не будет доступно для получения.

BOOL TranslateMessage(

[in] const MSG \*lpMsg

); – Преобразует сообщения с виртуальным ключом в символьные сообщения. Символьные сообщения помещаются в очередь сообщений вызывающего потока, чтобы их можно было прочитать при следующем вызове функции GetMessage или PeekMessage .

LRESULT DispatchMessage(

[in] const MSG \*lpMsg

); – Отправляет сообщение оконной процедуре. Обычно он используется для отправки сообщения, полученного функцией GetMessage.

LRESULT SendMessage(

[in] HWND hWnd,

[in] UINT Msg,

[in] WPARAM wParam,

[in] LPARAM lParam

); – Отправляет указанное сообщение в окно или окна. Функция SendMessage вызывает оконную процедуру для указанного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообщение.

BOOL SetWindowText(

[in] HWND hWnd,

[in, optional] LPCWSTR lpString

); – Изменяет текст строки заголовка указанного окна (если она имеется). Если указанное окно является элементом управления, текст элемента управления изменяется. Однако SetWindowText не может изменить текст элемента управления в другом приложении.

BOOL SetDlgItemText(

[in] HWND hDlg,

[in] int nIDDlgItem,

[in] LPCSTR lpString

); – Задает заголовок или текст элемента управления в диалоговом окне.

int MessageBox(

[in, optional] HWND hWnd,

[in, optional] LPCTSTR lpText,

[in, optional] LPCTSTR lpCaption,

[in] UINT uType

); – Отображает модальное диалоговое окно, содержащее значок системы, набор кнопок и краткое сообщение для конкретного приложения, например, сведения о состоянии или ошибке. Окно сообщения возвращает целочисленное значение, указывающее, какую кнопку нажал пользователь.

BOOL EnumWindows(

[in] WNDENUMPROC lpEnumFunc,

[in] LPARAM lParam

); – Перечисляет все окна верхнего уровня на экране, передавая дескриптор каждому окну, в свою очередь, в определяемую приложением функцию обратного вызова. EnumWindows продолжается до тех пор, пока не будет перечислено последнее окно верхнего уровня или функция обратного вызова не вернет значение FALSE.

UINT GetDlgItemInt(

[in] HWND hDlg,

[in] int nIDDlgItem,

[out, optional] BOOL \* lpTranslated,

[in] BOOL bSigned

); – Преобразует текст указанного элемента управления в диалоговом окне в целочисленное значение.

SetWindowPos(

\_In\_ HWND hWnd,

\_In\_opt\_ HWND hWndInsertAfter,

\_In\_ int X,

\_In\_ int Y,

\_In\_ int cx,

\_In\_ int cy,

\_In\_ UINT uFlags); – Изменяет размер, положение и порядок Z дочернего, всплывающего окна или окна верхнего уровня. Эти окна упорядочены в соответствии с их внешним видом на экране. Самое верхнее окно получает наивысший ранг и является первым окном в порядке Z.

WNDPROC Wndproc;

LRESULT Wndproc(

HWND unnamedParam1,

UINT unnamedParam2,

WPARAM unnamedParam3,

LPARAM unnamedParam4

) – Функция обратного вызова, определяемая в приложении и обрабатывающая сообщения, отправленные в окно. Тип WNDPROC определяет указатель на эту функцию обратного вызова. Имя WndProc является заполнителем для имени функции, определяемой в приложении.

GetWindowTextW(

\_In\_ HWND hWnd,

\_Out\_writes\_(nMaxCount) LPWSTR lpString,

\_In\_ int nMaxCount); - Копирует текст строки заголовка указанного окна (если она имеется) в буфер. Если указанное окно является элементом управления, копируется текст элемента управления. Однако GetWindowText не может получить текст элемента управления в другом приложении.

SetWindowTextW(

\_In\_ HWND hWnd,

\_In\_opt\_ LPCWSTR lpString); - Изменяет текст строки заголовка указанного окна (если она имеется). Если указанное окно является элементом управления, текст элемента управления изменяется. Однако SetWindowText не может изменить текст элемента управления в другом приложении.

SetWindowLongPtrW(

\_In\_ HWND hWnd,

\_In\_ int nIndex,

\_In\_ LONG\_PTR dwNewLong); - Изменяет атрибут указанного окна. Функция также задает значение с указанным смещением в дополнительной памяти окна.

FindWindowW(

\_In\_opt\_ LPCWSTR lpClassName,

\_In\_opt\_ LPCWSTR lpWindowName); - Извлекает дескриптор для окна верхнего уровня, имя класса и имя окна которого соответствуют указанным строкам. Эта функция не выполняет поиск дочерних окон. Эта функция не выполняет поиск с учетом регистра.

GetWindowRect(

\_In\_ HWND hWnd,

\_Out\_ LPRECT lpRect); – Извлекает размеры ограничивающего прямоугольника указанного окна. Измерения задаются в координатах экрана, которые находятся относительно левого верхнего угла экрана.

SetDlgItemInt(

\_In\_ HWND hDlg,

\_In\_ int nIDDlgItem,

\_In\_ UINT uValue,

\_In\_ BOOL bSigned); - Задает для текста элемента управления в диалоговом окне строковое представление указанного целочисленного значения.

void DialogBoxW(

[in, optional] hInstance,

[in] lpTemplate,

[in, optional] hWndParent,

[in, optional] lpDialogFunc

); - Создает модальное диалоговое окно из ресурса шаблона диалогового окна. DialogBox не возвращает управление, пока указанная функция обратного вызова не завершит модальное диалоговое окно путем вызова функции EndDialog .

DialogBox реализуется как вызов функции DialogBoxParam.

DestroyWindow(

\_In\_ HWND hWnd); - Уничтожает указанное окно. Функция отправляет WM\_DESTROY и WM\_NCDESTROY сообщения в окно, чтобы деактивировать его и снять с него фокус клавиатуры. Функция также уничтожает меню окна, очищает очередь сообщений потока, уничтожает таймеры, удаляет владение буфером обмена и разрывает цепочку просмотра буфера обмена (если окно находится в верхней части цепочки просмотра).

DefWindowProcW(

\_In\_ HWND hWnd,

\_In\_ UINT Msg,

\_In\_ WPARAM wParam,

\_In\_ LPARAM lParam); - Вызывает процедуру окна по умолчанию, чтобы обеспечить обработку по умолчанию для всех оконных сообщений, которые не обрабатываются приложением. Эта функция обеспечивает обработку каждого сообщения. Метод DefWindowProc вызывается с теми же параметрами, что и процедура окна.

EndDialog(

\_In\_ HWND hDlg,

\_In\_ INT\_PTR nResult); - Уничтожает модальное диалоговое окно, в результате чего система завершает обработку этого диалогового окна.

GetWindowTextA(

\_In\_ HWND hWnd,

\_Out\_writes\_(nMaxCount) LPSTR lpString,

\_In\_ int nMaxCount); - Копирует текст строки заголовка указанного окна (если она имеется) в буфер. Если указанное окно является элементом управления, копируется текст элемента управления. Однако GetWindowText не может получить текст элемента управления в другом приложении.

IsWindowVisible(

\_In\_ HWND hWnd); - Определяет состояние видимости указанного окна.

SendMessageA(

\_In\_ HWND hWnd,

\_In\_ UINT Msg,

\_Pre\_maybenull\_ \_Post\_valid\_ WPARAM wParam,

\_Pre\_maybenull\_ \_Post\_valid\_ LPARAM lParam); - Отправляет указанное сообщение в окно или окна. Функция SendMessage вызывает процедуру окна для указанного окна и не возвращает ее, пока эта процедура не обработает сообщение.

ShowWindow(

\_In\_ HWND hWnd,

\_In\_ int nCmdShow); - Задает состояние отображения указанного окна.

void GetNextWindow(

[in] hWnd,

[in] wCmd

); - Извлекает дескриптор следующего или предыдущего окна в Z-порядке. Следующее окно находится под указанным окном; предыдущее окно находится выше.

## Схемы основных алгоритмов



Рисунок 2 - Блок-схема заполнения списка запущенных видимых окон



Рисунок 3 - Блок-схема алгоритма вывода стиля выбранного окна

# Заключение

В данной курсовой работе было разработано программное обеспечение, реализующее работу с окнами, создание и уничтожение окон, перемещение, изменение размера и порядка окон, перечисление работающих в ОС окон, перечисление всех дочерних окон для конкретного окна, поиск окон, получение информации о свойствах окна и их изменение.

Результатом работы является функционально законченное приложение, выполняющее функции по работе c окнами.

# Список используемой литературы

1. Сведения о сообщениях и очередях сообщений - Win32 apps | Microsoft Learn - <https://learn.microsoft.com/ru-ru/windows/win32/winmsg/about-messages-and-message-queues> (дата обращения: 25.05.2024)
2. Заголовок Winuser.h - Win32 apps | Microsoft Learn - <https://learn.microsoft.com/ru-ru/windows/win32/api/winuser/> (дата обращения: 25.05.2024)
3. Создание классических приложений для Windows с помощью API Win32 - Win32 apps | Microsoft Learn - <https://learn.microsoft.com/ru-ru/windows/win32/> (дата обращения: 25.05.2024)

# Приложение А

**Файл Resource.h:**

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by WinTab.rc

#define IDS\_APP\_TITLE 103

#define IDR\_MAINFRAME 128

#define IDD\_WINTAB\_DIALOG 102

#define IDD\_ABOUTBOX 103

#define IDM\_ABOUT 104

#define IDM\_EXIT 105

#define IDI\_WINTAB 107

#define IDI\_SMALL 108

#define IDC\_WINTAB 109

#define IDC\_MYICON 2

#ifndef IDC\_STATIC

#define IDC\_STATIC -1

// New functional

#define IDC\_LISTBOX 201

#define IDC\_CHILD\_LISTBOX 202

#define IDC\_CREATE\_WINDOW 203

#define IDC\_CLOSE\_WINDOW 204

#define IDC\_MOVEANDRESIZE\_BUTTON 205

#define IDC\_SET\_STYLE 206

#define IDC\_REFRESH\_BUTTON 207

#define IDC\_SEARCH\_BUTTON 208

#define IDC\_RENAME\_TITLE 209

#define IDC\_TITLE\_EDIT 210

#define IDC\_X\_EDIT 211

#define IDC\_Y\_EDIT 212

#define IDC\_WIDTH\_EDIT 213

#define IDC\_HEIGHT\_EDIT 214

#define IDC\_WINDOWPOS\_EDIT 215

#define IDC\_SEARCH\_EDIT 216

#define IDC\_STYLE\_EDIT 217

#endif

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NO\_MFC 130

#define \_APS\_NEXT\_RESOURCE\_VALUE 129

#define \_APS\_NEXT\_COMMAND\_VALUE 32771

#define \_APS\_NEXT\_CONTROL\_VALUE 1000

#define \_APS\_NEXT\_SYMED\_VALUE 110

#endif

#endif

**Файл WinTab.cpp:**

#include "framework.h"

#include "WinTab.h"

#define MAX\_LOADSTRING 100

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

HWND hwndList; // Listbox handle

HWND hwndChildList; // Child windows listbox handle

HWND hwndButtonCreate; // Create window botton handle

HWND hwndButtonClose; // Close window botton handle

HWND hwndButtonMoveAndResize; // Move and resize window botton handle

HWND hwndButtonSetStyle; // Set style button handle

HWND hwndButtonRefresh; // Refresh button handle

HWND hwndButtonSearch; // Search windows button handle

HWND hwndButtonRename; // Rename window button handle

HWND hwndXEdit, hwndYEdit, hwndWidthEdit, hwndHeightEdit, hwndTitleEdit, hwndSearchEdit, hwndStyleEdit, hwndWindowPosEdit;

HWND selectedHwnd, hwndListLabel, hwndChildListLabel, hwndTitleLabel, hwndXYLabel, hwndWithHeightLabel, hwndSearchLabel, hwndStyleLabel;

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam);

BOOL CALLBACK EnumChildProc(HWND hwnd, LPARAM lParam);

void CreateNewWindow(); // Function tot create new window

void SearchWindows(); // Function to search windows

void PrintWindowStyle(HWND hwnd);// Function to display selected window

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// Initialize global strings

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_WINTAB, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_WINTAB));

MSG msg;

// Main message loop:

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

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

// FUNCTION: MyRegisterClass()

// PURPOSE: Registers the window class.

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_WINTAB));

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_WINTAB);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

// FUNCTION: InitInstance(HINSTANCE, int)

// PURPOSE: Saves instance handle and creates main window

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW & ~WS\_MAXIMIZEBOX,

CW\_USEDEFAULT, 0, 900, 600, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

// PURPOSE: Processes messages for the main window.

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

{

switch (message)

{

case WM\_CREATE:

// Area for Get active the windows list

hwndListLabel = CreateWindow(L"STATIC", L"Windows:", WS\_CHILD | WS\_VISIBLE,

10, 0, 70, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndList = CreateWindow(L"LISTBOX", L"", WS\_VISIBLE | WS\_CHILD | WS\_VSCROLL | LBS\_STANDARD | LBS\_NOTIFY,

10, 20, 400, 300, hWnd, (HMENU)IDC\_LISTBOX, hInst, NULL);

// Area for Get active child windows list the window

hwndChildListLabel = CreateWindow(L"STATIC", L"Child windows:", WS\_CHILD | WS\_VISIBLE,

10, 320, 100, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndChildList = CreateWindow(L"LISTBOX", L"", WS\_VISIBLE | WS\_CHILD | WS\_VSCROLL | LBS\_STANDARD,

10, 340, 400, 200, hWnd, (HMENU)IDC\_CHILD\_LISTBOX, hInst, NULL);

// Button for refresh the windows list

hwndButtonRefresh = CreateWindow(L"BUTTON", L"Refresh", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

420, 20, 100, 30, hWnd, (HMENU)IDC\_REFRESH\_BUTTON, hInst, NULL);

// Button for create new window

hwndButtonCreate = CreateWindow(L"BUTTON", L"New window", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

545, 20, 100, 30, hWnd, (HMENU)IDC\_CREATE\_WINDOW, hInst, NULL);

// Button for close the window

hwndButtonClose = CreateWindow(L"BUTTON", L"Close", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

670, 20, 100, 30, hWnd, (HMENU)IDC\_CLOSE\_WINDOW, hInst, NULL);

// Area for get and set the windows title

hwndTitleLabel = CreateWindow(L"STATIC", L"Title:", WS\_CHILD | WS\_VISIBLE,

420, 60, 50, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndTitleEdit = CreateWindow(L"EDIT", NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER,

420, 80, 330, 30, hWnd, (HMENU)IDC\_TITLE\_EDIT, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndButtonRename = CreateWindow(L"BUTTON", L"Rename", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

770, 80, 100, 30, hWnd, (HMENU)IDC\_RENAME\_TITLE, hInst, NULL);

// Area for enter move and resize

hwndXYLabel = CreateWindow(L"STATIC", L"X:\t Y:", WS\_CHILD | WS\_VISIBLE, 420, 130, 110, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndXEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER, 420, 150, 50, 25, hWnd, (HMENU)IDC\_X\_EDIT, NULL, NULL);

hwndYEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER, 480, 150, 50, 25, hWnd, (HMENU)IDC\_Y\_EDIT, NULL, NULL);

hwndWithHeightLabel = CreateWindow(L"STATIC", L"Width:\t Height: Pos:", WS\_CHILD | WS\_VISIBLE, 580, 130, 170, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndWidthEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER, 580, 150, 50, 25, hWnd, (HMENU)IDC\_WIDTH\_EDIT, NULL, NULL);

hwndHeightEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER, 640, 150, 50, 25, hWnd, (HMENU)IDC\_HEIGHT\_EDIT, NULL, NULL);

hwndWindowPosEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_NUMBER, 700, 150, 50, 25, hWnd, (HMENU)IDC\_WINDOWPOS\_EDIT, NULL, NULL);

hwndButtonMoveAndResize = CreateWindow(L"BUTTON", L"Move/resize", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

770, 145, 100, 30, hWnd, (HMENU)IDC\_MOVEANDRESIZE\_BUTTON, hInst, NULL);

// Area for search window

hwndSearchLabel = CreateWindow(L"STATIC", L"Search window:", WS\_CHILD | WS\_VISIBLE,

420, 190, 110, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndSearchEdit = CreateWindow(L"EDIT", NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER,

420, 210, 330, 30, hWnd, (HMENU)IDC\_SEARCH\_EDIT, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndButtonSearch = CreateWindow(L"BUTTON", L"Search", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

770, 210, 100, 30, hWnd, (HMENU)IDC\_SEARCH\_BUTTON, hInst, NULL);

// Area for get and set the window style

hwndStyleLabel = CreateWindow(L"STATIC", L"Style window:", WS\_CHILD | WS\_VISIBLE,

420, 260, 110, 20, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndStyleEdit = CreateWindow(L"EDIT", NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER,

420, 280, 150, 30, hWnd, (HMENU)IDC\_STYLE\_EDIT, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

hwndButtonSetStyle = CreateWindow(L"BUTTON", L"Set Style", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

590, 280, 100, 30, hWnd, (HMENU)IDC\_SET\_STYLE, hInst, NULL);

SendMessage(hwndList, LB\_RESETCONTENT, 0, 0);

EnumWindows(EnumWindowsProc, reinterpret\_cast<LPARAM>(hwndList));

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDC\_REFRESH\_BUTTON:

selectedHwnd = NULL;

SendMessage(hwndList, LB\_RESETCONTENT, 0, 0);

EnumWindows(EnumWindowsProc, reinterpret\_cast<LPARAM>(hwndList));

SendMessage(hwndChildList, LB\_RESETCONTENT, 0, 0); // Reset child list

break;

case IDC\_CREATE\_WINDOW:

CreateNewWindow();

break;

case IDC\_CLOSE\_WINDOW:

// Close window

if (selectedHwnd)

{

SendMessage(selectedHwnd, WM\_CLOSE, 0, 0);

selectedHwnd = NULL;

SendMessage(hwndList, LB\_RESETCONTENT, 0, 0);

EnumWindows(EnumWindowsProc, reinterpret\_cast<LPARAM>(hwndList));

SendMessage(hwndChildList, LB\_RESETCONTENT, 0, 0); // Reset child list

}

break;

case IDC\_RENAME\_TITLE:

{

if (selectedHwnd)

{

wchar\_t newTitle[256];

GetWindowText(hwndTitleEdit, newTitle, 256);

SetWindowText(selectedHwnd, newTitle);

}

}

break;

case IDC\_MOVEANDRESIZE\_BUTTON:

{

if (selectedHwnd)

{

int x = GetDlgItemInt(hWnd, IDC\_X\_EDIT, NULL, TRUE);

int y = GetDlgItemInt(hWnd, IDC\_Y\_EDIT, NULL, TRUE);

int width = GetDlgItemInt(hWnd, IDC\_WIDTH\_EDIT, NULL, TRUE);

int height = GetDlgItemInt(hWnd, IDC\_HEIGHT\_EDIT, NULL, TRUE);

int zOrder = GetDlgItemInt(hWnd, IDC\_WINDOWPOS\_EDIT, NULL, TRUE);

// Move and resize the window

SetWindowPos(selectedHwnd, NULL, x, y, width, height, SWP\_NOZORDER | SWP\_NOACTIVATE);

// Change the z-order

HWND hwndInsertAfter = HWND\_TOP;

if (zOrder > 0) {

HWND hwnd = GetTopWindow(NULL);

while (hwnd && zOrder > 0) {

hwndInsertAfter = hwnd;

hwnd = GetNextWindow(hwnd, GW\_HWNDNEXT);

zOrder--;

}

}

SetWindowPos(selectedHwnd, hwndInsertAfter, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE | SWP\_NOACTIVATE);

}

}

break;

case IDC\_SEARCH\_BUTTON:

SearchWindows();

break;

case IDC\_SET\_STYLE:

if (selectedHwnd) {

wchar\_t styleText[16];

GetWindowText(hwndStyleEdit, styleText, 16);

LONG\_PTR newStyle = wcstol(styleText, NULL, 16);

SetWindowLongPtr(selectedHwnd, GWL\_STYLE, newStyle);

SetWindowPos(selectedHwnd, NULL, 0, 0, 0, 0, SWP\_NOMOVE | SWP\_NOSIZE | SWP\_NOZORDER | SWP\_FRAMECHANGED);

}

break;

case IDC\_LISTBOX:

if (HIWORD(wParam) == LBN\_SELCHANGE) {

int index = SendMessage(hwndList, LB\_GETCURSEL, 0, 0);

if (index != LB\_ERR) {

TCHAR title[256];

SendMessage(hwndList, LB\_GETTEXT, index, (LPARAM)title);

// Get HWND the window to the title

selectedHwnd = FindWindow(NULL, title);

if (selectedHwnd) {

SetWindowText(hwndTitleEdit, title); // Set title name

RECT rect;

GetWindowRect(selectedHwnd, &rect);

SetDlgItemInt(hWnd, IDC\_X\_EDIT, rect.left, TRUE);

SetDlgItemInt(hWnd, IDC\_Y\_EDIT, rect.top, TRUE);

SetDlgItemInt(hWnd, IDC\_WIDTH\_EDIT, rect.right - rect.left, TRUE);

SetDlgItemInt(hWnd, IDC\_HEIGHT\_EDIT, rect.bottom - rect.top, TRUE);

// Enumerate and list child windows

SendMessage(hwndChildList, LB\_RESETCONTENT, 0, 0); // Reset child list

EnumChildWindows(selectedHwnd, EnumChildProc, reinterpret\_cast<LPARAM>(hwndChildList));

// Display window style

PrintWindowStyle(selectedHwnd);

}

}

}

break;

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Message handler for about box.

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

// EnumWindows callback function

BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)

{

HWND hwndList = reinterpret\_cast<HWND>(lParam);

char title[256];

GetWindowTextA(hwnd, title, sizeof(title));

if (IsWindowVisible(hwnd) && strlen(title) > 0 &&

strcmp(title, "Windows Input Experience") != 0 &&

strcmp(title, "Program Manager") != 0 &&

strcmp(title, "C:\\Users\\nasaa\\OneDrive\\Документы\\Rainmeter\\Skins\\Mond\\Clock\\Clock.ini") != 0)

{

SendMessageA(hwndList, LB\_ADDSTRING, 0, (LPARAM)title);

}

return TRUE;

}

// EnumChildProc callback function

BOOL CALLBACK EnumChildProc(HWND hwnd, LPARAM lParam)

{

HWND hwndList = reinterpret\_cast<HWND>(lParam);

char title[256];

GetWindowTextA(hwnd, title, sizeof(title));

if (IsWindowVisible(hwnd) && strlen(title) > 0)

{

SendMessageA(hwndList, LB\_ADDSTRING, 0, (LPARAM)title);

}

return TRUE;

}

// Function to create a new window

void CreateNewWindow()

{

// Register a new window class

const wchar\_t CLASS\_NAME[] = L"NewWindowClass";

WNDCLASS wc = { };

wc.lpfnWndProc = DefWindowProc;

wc.hInstance = hInst;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the new window

HWND hwndNew = CreateWindowEx(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"New Window", // Window title

WS\_OVERLAPPEDWINDOW, // Window style

// Size and position

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, // Parent window

NULL, // Menu

hInst, // Instance handle

NULL // Additional application data

);

if (hwndNew)

{

ShowWindow(hwndNew, SW\_SHOW);

}

}

// Function to search windows based on the input in the search edit box

void SearchWindows()

{

wchar\_t searchText[256];

GetWindowText(hwndSearchEdit, searchText, 256);

SendMessage(hwndList, LB\_RESETCONTENT, 0, 0);

EnumWindows([](HWND hwnd, LPARAM lParam) -> BOOL {

HWND hwndList = reinterpret\_cast<HWND>(lParam);

char title[256];

GetWindowTextA(hwnd, title, sizeof(title));

wchar\_t wTitle[256];

MultiByteToWideChar(CP\_ACP, 0, title, -1, wTitle, 256);

// Get title for search

wchar\_t searchText[256];

GetWindowText(hwndSearchEdit, searchText, 256);

// Checking for windows exclusion "Windows Input Experience", "Program Manager" and Rainmeter Clock

if (IsWindowVisible(hwnd) && wcslen(wTitle) > 0 &&

wcscmp(wTitle, L"Windows Input Experience") != 0 &&

wcscmp(wTitle, L"Program Manager") != 0 &&

wcscmp(wTitle, L"C:\\Users\\nasaa\\OneDrive\\Документы\\Rainmeter\\Skins\\Mond\\Clock\\Clock.ini") != 0 &&

wcsstr(wTitle, searchText) != NULL)

{

SendMessageW(hwndList, LB\_ADDSTRING, 0, (LPARAM)wTitle);

}

return TRUE;

}, reinterpret\_cast<LPARAM>(hwndList));

}

void PrintWindowStyle(HWND hwnd) {

if (hwnd) {

LONG\_PTR style = GetWindowLongPtr(hwnd, GWL\_STYLE);

wchar\_t styleText[16];

swprintf(styleText, 16, L"%08X", style);

SetWindowText(hwndStyleEdit, styleText);

// Get and display z-order

HWND hwndPrev = GetNextWindow(hwnd, GW\_HWNDPREV);

int zOrder = 0;

while (hwndPrev) {

zOrder++;

hwndPrev = GetNextWindow(hwndPrev, GW\_HWNDPREV);

}

SetDlgItemInt(GetParent(hwndStyleEdit), IDC\_WINDOWPOS\_EDIT, zOrder, TRUE);

}

}

# Приложение Б

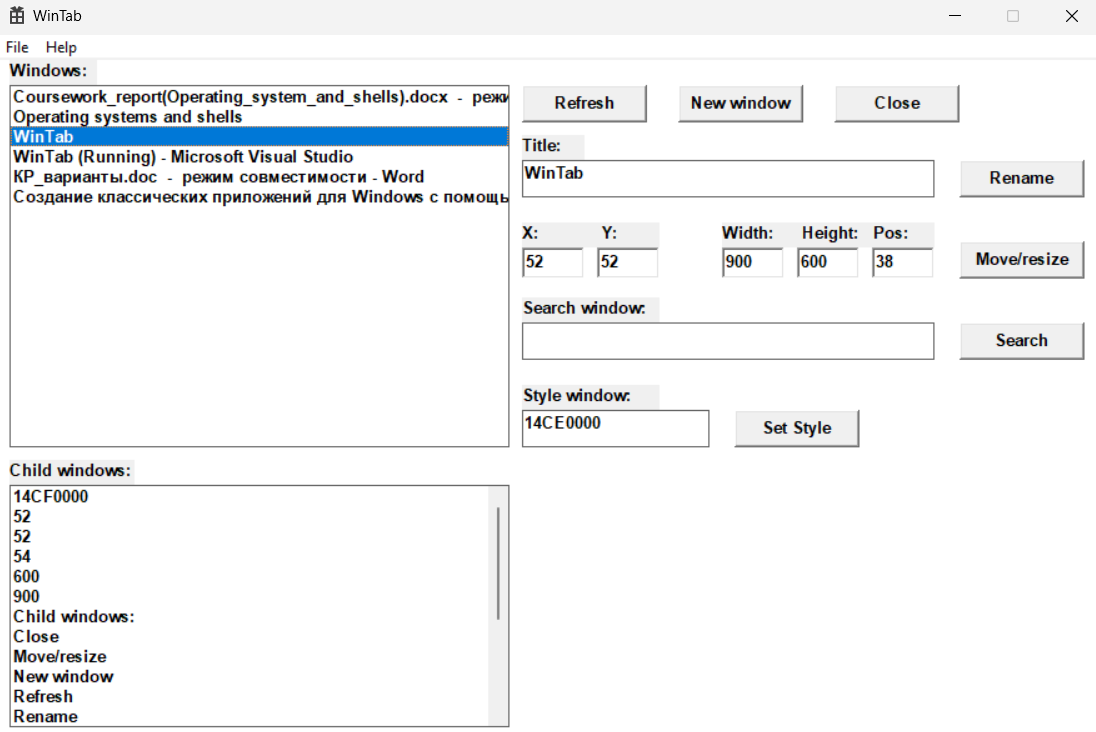


Рисунок 4 – Вывод дочерних окон, названия окна, расположения, размеров и стиля для текущего окна

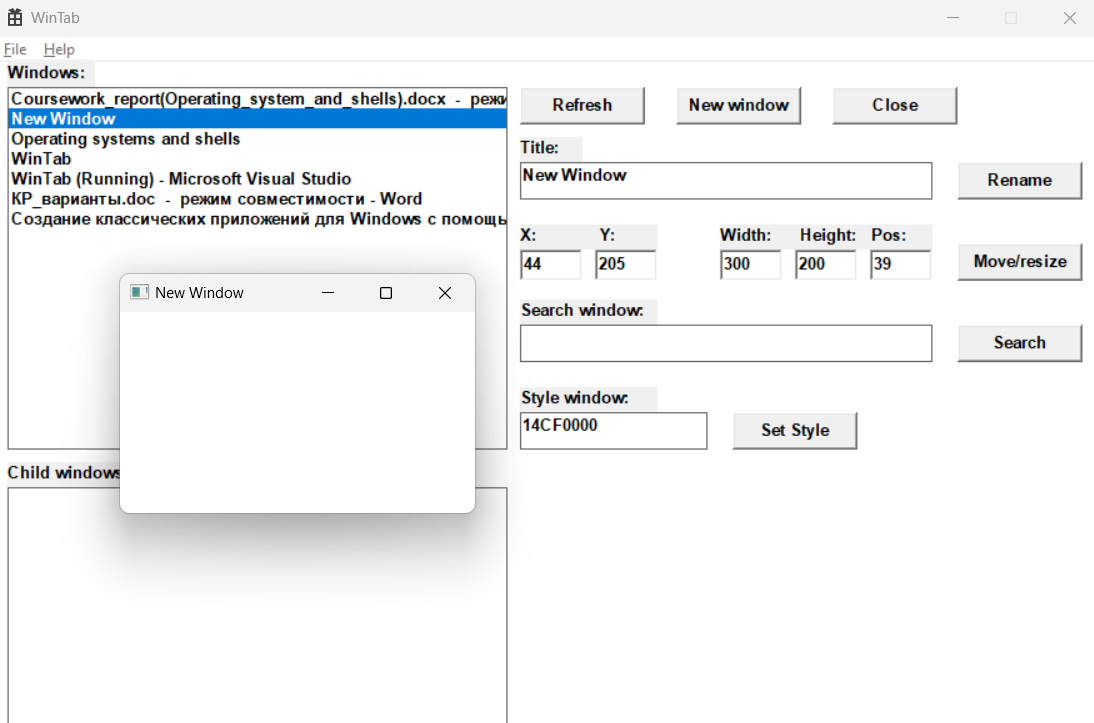


Рисунок 5 – Создание нового окна и изменение его размеров

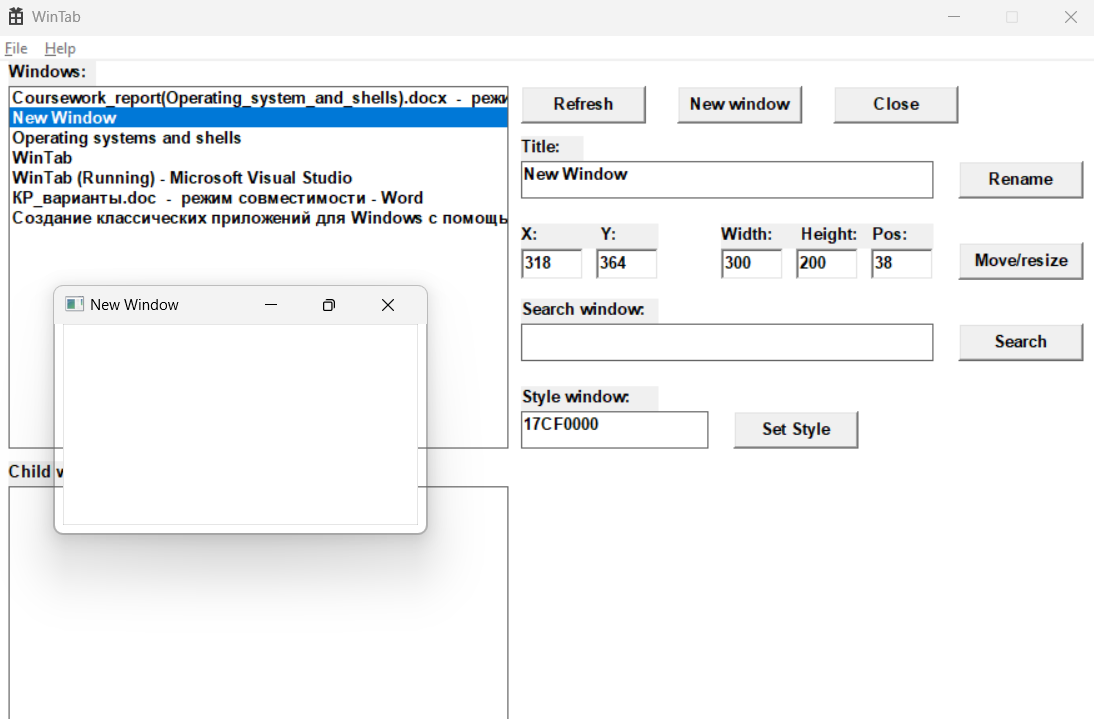


Рисунок 6 - Изменение стиля нового окна

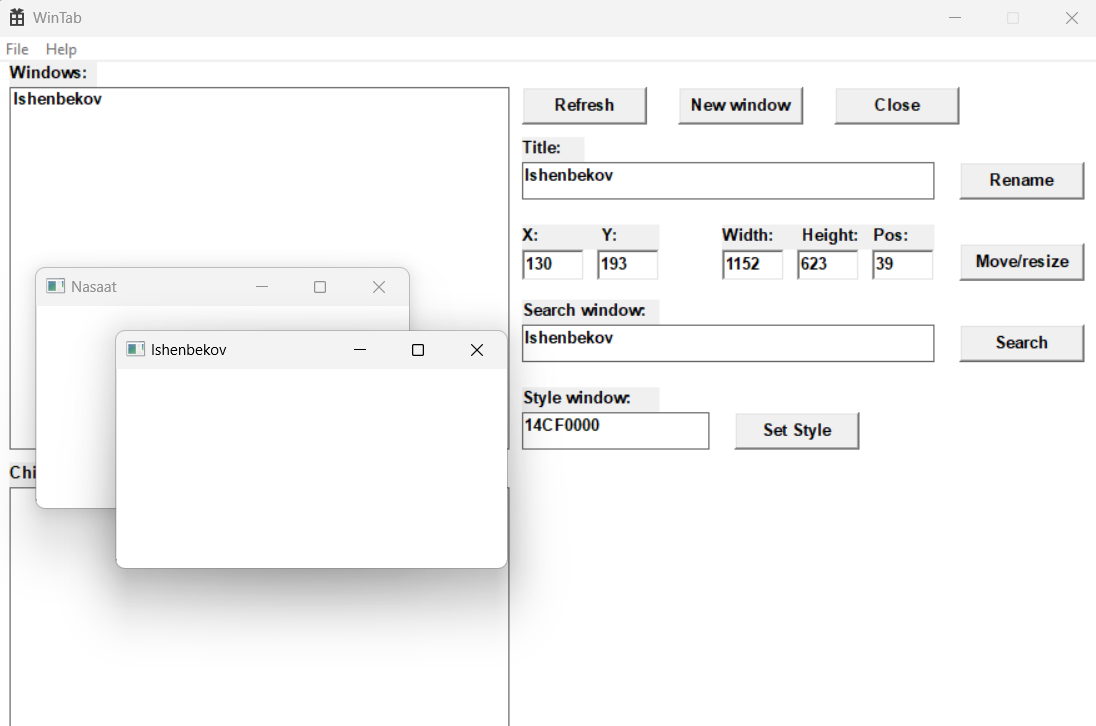


Рисунок 7 – Поиск активного окна в системе