Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

на тему

Основы программирования в Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений.

Студент: гр.153504

Пригожий К. А.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146635545)

[2 Теоретические сведения 4](#_Toc146635546)

[3 Результат выполнения программы 5](#_Toc146635547)

[Список использованных источников 9](#_Toc146635548)

[Приложение А 10](#_Toc146635549)

# 1 ЦЕЛЬ РАБОТЫ

Улучшить навыки программирования на платформе *Windows* *API*, включая работу с оконным интерфейсом, обработку событий и взаимодействие с системой. Разработать и реализовать приложение для чтения, редактирования и записи текстовых документов с возможностью выделения и копирования текста в буфер обмена и сохранения изменений в файле. Это позволит углубить знания о работе с элементами управления, вводом и выводом текстовой информации, а также организацией файловых операций.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В ходе разработки приложения для чтения и редактирования текстовых документов с использованием *Windows* *API* и элемента управления *Rich* *Edit* (*richEditControl*) представляется необходимым усвоить основные функции и концепции.

Инициация данного процесса начинается с регистрации класса окна через функцию *RegisterClass*. Этот шаг предполагает определение параметров окна, которое будет создано в приложении. Если вам планируется использовать элемент управления *Rich Edit* (*richEditControl*), необходимо зарегистрировать соответствующий класс.

Затем выполняется создание экземпляра окна, и для этой задачи используются функции *CreateWindow* или *CreateWindowEx*. В случае, когда *richEditControl* применяется, создается элемент управления класса "*RICHEDIT*". Это окно представляет контейнер для элементов управления, в частности, текстового редактора.

Для обеспечения взаимодействия с пользователем, средство *MessageBox* находит применение, исключительно для отображения информационных сообщений или при необходимости получения ввода пользователя. Это может быть полезно, например, для предостережения пользователя о несохраненных изменениях.

Центральная часть кода приложения обычно находится в функции *wWinMain* (или *WinMain*), которая является точкой входа для *Windows*-приложений. Она принимает аргументы командной строки и обычно выполняет инициализацию приложения, включая создание и управление главным окном.

Для работы с файлами, такими как открытие и сохранение текстовых документов, функция *GetOpenFileName* используется для отображения диалогового окна выбора файла. Функции *WriteFile* и *ReadFile* позволяют записывать и читать данные из файла, включая содержимое *richEditControl*.

*RichEditControl* – это элемент управления *Windows*, предоставляющий расширенный текстовый редактор. Он поддерживает управление содержимым, вставку изображений и форматирование текста.

Все перечисленные функции и концепции представляют собой основу для создания приложения для работы с текстовыми документами на платформе *Windows*. Важно также осознавать необходимость обработки сообщений, обновления интерфейса и других аспектов разработки *Windows*-приложений.

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

При запуске программы можно увидеть текстовый редактор, который является оконным приложением с простым интерфейсом (рисунок 1).

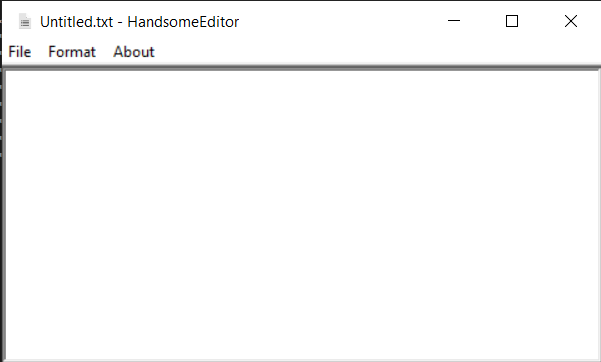


Рисунок 1 – Графический интерфейс программы

В заголовке программы можно увидеть название открытого файла(по умолчанию *Untitled.txt*) и название приложения «*HandsomeEditor*».

При нажатии на кнопку «*About*» всплывает окно с информацией о разработчике (рисунок 2).

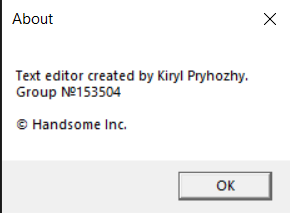


Рисунок 2 – Окно информации о разработчике

При нажати на кнопку «*File*» октрывается меню работы с файлами (рисунок 3).

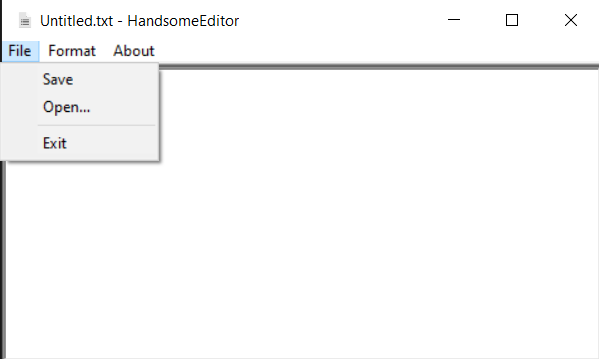


Рисунок 3 – Меню работы с файлами

При нажатии кнопки «*Save»* открывается диалоговое окно для выбора файла, в который будет произведено сохранение (рисунок 4).

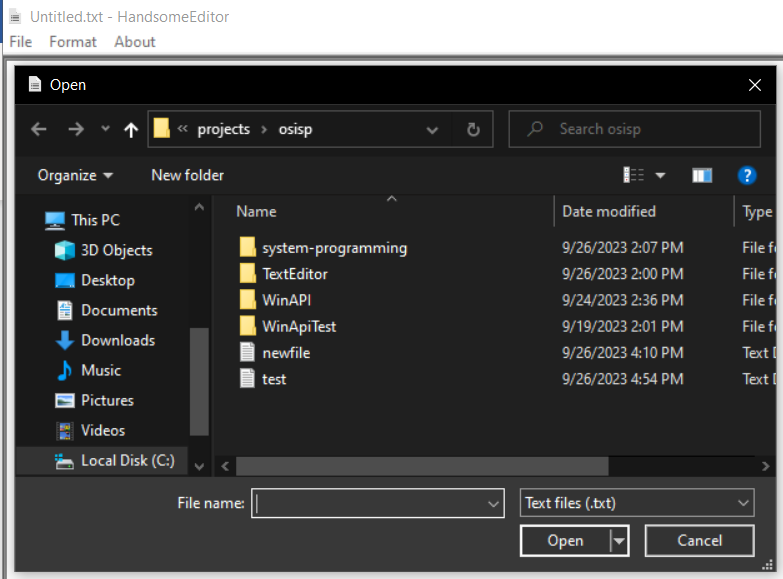


Рисунок 4 – Диалоговое окно для сохранения файла

Если файл не был выбран всплывает окно с сообщением, говорящим, что во время выбора файла произошла ошибка (рисунок 5).

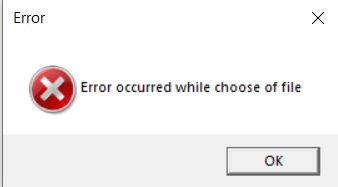


Рисунок 5 – Окно ошибки при невыбранном файле

При успешном сохранении в файл имя файла в названии изменяется на имя выбранного файла (рисунок 6).

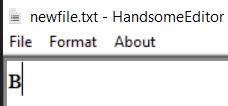


Рисунок 6 – Название главного окна после сохранения в файл

Также при изменении содержимого поля ввода в конец названия файла в имени главного окна добавляется знак «\*», который убирается при сохранении в файл (рисунок 7).

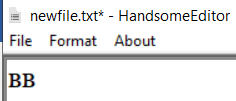


Рисунок 7 – Новое название главного окна, после изменения содержимого поля ввода

При выходе из программы с измененным содержимым поля ввода всплывает окно, которое спрашивает о необходимости сохранить изменения в файл (рисунок 8).

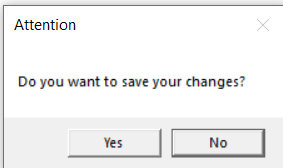


Рисунок 8 – Диалоговое окно, спрашивающее о необходимости сохранения изменений

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Что такое окно? [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/what-is-a-window>.

[2] Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming>.

[3] Сообщения окна [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messages>.

[4] Элемент контроля ввода RichEdit [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/controls/rich-edit-controls>

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл actions.h**

#pragma once

#include "headers.h"

#include <string>

void ProcessWMCommand(HWND hWnd, WPARAM wParam, LPARAM lParam);

/\*

Returns configured CHOOSECOLOR struct

\*/

CHOOSECOLOR ConfigureChooseColor(HWND hWnd, COLORREF initialColor);

**Файл actions.cpp**

#include "actions.h"

void ProcessWMCommand(HWND hWnd, WPARAM wParam, LPARAM lParam)

{

switch (LOWORD(wParam)) {

case ID\_FILE\_SAVE:

SaveFile(hWnd);

break;

case ID\_FILE\_LOAD:

if (contentAlreadyChanged and MessageBox(hWnd, L"Do you want to save your changes?", L"Attention", MB\_YESNO) == IDYES) {

SaveFile(hWnd);

}

LoadFile(hWnd);

break;

case ID\_ABOUT:

MessageBoxA(hWnd, "Text editor created by Kiryl Pryhozhy. \nGroup №153504\n\n© Handsome Inc.", "About", MB\_OK);

break;

case ID\_EXIT:

PostQuitMessage(0);

break;

}

}

**Файл appWindow.h**

#pragma once

#include "headers.h"

//functions

WNDCLASSEX CreateMainWindowClass(HINSTANCE hInstance, WNDPROC windowProc);

HWND CreateMainWindow(HINSTANCE hInstance);

HWND CreateTextField(HWND parent);

bool IsTextFieldChanged(WPARAM wParam, LPARAM lParam);

void AppendAsteriskToFilename();

void RemoveAsteriskFromFilename();

**Файл appWindow.cpp**

#include "appWindow.h"

#include <string>

WNDCLASSEX CreateMainWindowClass(HINSTANCE hInstance, WNDPROC windowProc) {

WNDCLASSEX windowClass{};

windowClass.cbSize = sizeof(WNDCLASSEX);

windowClass.lpfnWndProc = windowProc;

windowClass.hInstance = hInstance;

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

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

windowClass.lpszMenuName = MAKEINTRESOURCE(IDR\_MENU1);

windowClass.lpszClassName = MAIN\_WINDOW\_CLASS\_NAME;

windowClass.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

return windowClass;

}

HWND CreateMainWindow(HINSTANCE hInstance)

{

std::wstring windowTitle = std::wstring(DEFAULT\_FILENAME) + WINDOW\_TITLE\_POSTFIX;

return CreateWindowEx(

WS\_EX\_CLIENTEDGE,

MAIN\_WINDOW\_CLASS\_NAME,

windowTitle.c\_str(),

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, MAIN\_WINDOW\_WIDTH, MAIN\_WINDOW\_HEIGHT,

NULL, NULL, hInstance, NULL);

}

HWND CreateTextField(HWND parent)

{

LoadLibrary(TEXT("Msftedit.dll"));

charFormat.cbSize = sizeof(CHARFORMAT);

charFormat.yHeight = fontSize \* 20;

charFormat.crTextColor = fontColor;

charFormat.dwMask = CFM\_COLOR | CFM\_SIZE | CFM\_FACE;

return CreateWindowEx(

WS\_EX\_CLIENTEDGE,

MSFTEDIT\_CLASS, L"",

WS\_BORDER | WS\_CHILD | WS\_VISIBLE | ES\_MULTILINE | WS\_VSCROLL | WS\_TABSTOP,

0, 0, TEXT\_FIELD\_WIDTH, TEXT\_FIELD\_HEIGHT,

parent, NULL, NULL, NULL

);

}

bool IsTextFieldChanged(WPARAM wParam, LPARAM lParam)

{

bool r1 = HIWORD(wParam) == EN\_CHANGE;

bool r2 = (HWND)lParam == hTextField;

OutputDebugStringA(std::to\_string(r1).c\_str());

OutputDebugStringA(std::to\_string(r2).c\_str());

return r1 and r2;

}

void AppendAsteriskToFilename()

{

int titleLength = GetWindowTextLength(hMainWindow);

std::wstring windowTitle(titleLength + 1, '\0');

GetWindowText(hMainWindow, &windowTitle[0], titleLength + 1);

int pos = windowTitle.find(WINDOW\_TITLE\_POSTFIX);

if (pos != -1) {

windowTitle.insert(pos, L"\*");

SetWindowText(hMainWindow, windowTitle.c\_str());

contentAlreadyChanged = true;

}

}

void RemoveAsteriskFromFilename()

{

int titleLength = GetWindowTextLength(hMainWindow);

std::wstring windowTitle(titleLength + 1, L'\0');

GetWindowText(hMainWindow, &windowTitle[0], titleLength + 1);

size\_t asteriskPos = windowTitle.find('\*');

if (asteriskPos != std::string::npos) {

windowTitle.erase(asteriskPos, 1);

}

SetWindowText(hMainWindow, windowTitle.c\_str());

}

**Файл fileUtil.h**

#pragma once

#include "headers.h"

bool OpenFileDialog(HWND hWnd);

/\*

Reads data from passed file and set it to text field

\*/

void ReadDataFromFile(wchar\_t filePath[]);

void SaveDataFromTextFieldToFile(wchar\_t filePath[]);

void SaveFile(HWND hWnd);

void LoadFile(HWND hWnd);

**Файл fileUtil.cpp**

#include "fileUtil.h"

bool OpenFileDialog(HWND hWnd)

{

OPENFILENAME openFileName{};

wchar\_t filename[FILE\_PATH\_BUFFER\_SIZE] = { '\0' };

// Initialize OPENFILENAME

ZeroMemory(&openFileName, sizeof(openFileName));

openFileName.lStructSize = sizeof(openFileName);

openFileName.hwndOwner = hWnd;

openFileName.lpstrFile = filename;

openFileName.lpstrFile[0] = '\0';

openFileName.nMaxFile = 256;

openFileName.lpstrFilter = L"Text files (.txt)\0\*.txt\0All files\0\*.\*\0";

openFileName.nFilterIndex = 1;

openFileName.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

openFileName.lpstrInitialDir = FILE\_PATH;

// Display the Open dialog box.

if (GetOpenFileName(&openFileName) == TRUE) {

wcscpy\_s(FILE\_PATH, FILE\_PATH\_BUFFER\_SIZE, filename);

return true;

}

else {

MessageBoxA(hWnd, "Error occurred while choose of file", "Error", MB\_ICONERROR);

return false;

}

}

void ReadDataFromFile(wchar\_t filePath[])

{

HANDLE hFile = CreateFile(filePath, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {

OutputDebugStringA("Error occured while opening the file");

return;

}

DWORD fileSize = GetFileSize(hFile, NULL);

char\* contentArray = new char[fileSize + 1]; // +1 for null terminator

DWORD bytesRead = 0;

if (!ReadFile(hFile, contentArray, fileSize, &bytesRead, NULL)) {

OutputDebugStringA("Error reading from the file.");

}

contentArray[bytesRead] = '\0';

SetWindowTextA(hTextField, contentArray);

delete[] contentArray;

CloseHandle(hFile);

}

void SaveDataFromTextFieldToFile(wchar\_t filePath[])

{

HANDLE FileToSave = CreateFile(filePath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

int saveLength = GetWindowTextLength(hTextField) + 1;

char\* data = new char[saveLength];

saveLength = GetWindowTextA(hTextField, data, saveLength);

DWORD bytesWritten;

if (!WriteFile(FileToSave, data, saveLength, &bytesWritten, NULL)) {

OutputDebugStringA("Error when writing to the file.");

}

CloseHandle(FileToSave);

delete[] data;

}

void SaveFile(HWND hWnd)

{

if (!OpenFileDialog(hWnd))

return;

SaveDataFromTextFieldToFile(FILE\_PATH);

RemoveAsteriskFromFilename();

contentAlreadyChanged = false;

std::wstring filename = std::wstring(PathFindFileName(FILE\_PATH)) + WINDOW\_TITLE\_POSTFIX;

SetWindowText(hMainWindow, filename.c\_str());

}

void LoadFile(HWND hWnd)

{

if (!OpenFileDialog(hWnd))

return;

ReadDataFromFile(FILE\_PATH);

std::wstring filename = std::wstring(PathFindFileName(FILE\_PATH)) + WINDOW\_TITLE\_POSTFIX;

SetWindowText(hMainWindow, filename.c\_str());

contentAlreadyChanged = false;

}

**Файл global.h**

#ifndef GLOBALS\_H

#define GLOBALS\_H

#include "headers.h"

#include <Windows.h>

#include <string>

#include <richedit.h>

extern HWND hTextField;

extern HWND hMainWindow;

extern bool contentAlreadyChanged;

extern int fontSize;

extern std::string font;

extern COLORREF fontColor;

extern COLORREF backgroundColor;

extern CHARFORMAT charFormat;

#define FILE\_PATH\_BUFFER\_SIZE 256

#define BUFFER\_SIZE 64000

// constants

extern const wchar\_t MAIN\_WINDOW\_CLASS\_NAME[];

extern const int MAIN\_WINDOW\_WIDTH;

extern const int MAIN\_WINDOW\_HEIGHT;

extern const int TEXT\_FIELD\_WIDTH;

extern const int TEXT\_FIELD\_HEIGHT;

extern const wchar\_t DEFAULT\_FILENAME[];

extern wchar\_t WINDOW\_TITLE\_POSTFIX[];

extern wchar\_t FILE\_PATH[];

#endif

**Файл global.cpp**

#include "global.h"

HWND hTextField{};

HWND hMainWindow{};

bool contentAlreadyChanged = false;

int fontSize = 18;

std::string font = "Times New Roman";

COLORREF fontColor = RGB(0, 0, 0);

COLORREF backgroundColor = RGB(255, 255, 255);

CHARFORMAT charFormat{ 0 };

const wchar\_t DEFAULT\_FILENAME[] = L"Untitled.txt";

wchar\_t WINDOW\_TITLE\_POSTFIX[] = L" - HandsomeEditor";

wchar\_t FILE\_PATH[FILE\_PATH\_BUFFER\_SIZE];

// constants

const wchar\_t MAIN\_WINDOW\_CLASS\_NAME[] = L"MainClassName";

const int MAIN\_WINDOW\_WIDTH = 500;

const int MAIN\_WINDOW\_HEIGHT = 300;

const int TEXT\_FIELD\_WIDTH = 400;

const int TEXT\_FIELD\_HEIGHT = 250;

**Файл headers.h**

#pragma once

#include <Windows.h>

#include "resource.h"

#include "fileUtil.h"

#include "main.h"

#include "resource.h"

#include "actions.h"

#include "appWindow.h"

#include "global.h"

#include "shlwapi.h"

**Файл resource.h**

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by TextEditor.rc

//

#define IDR\_MENU1 101

#define IDC\_RICHEDIT 102

#define IDI\_ICON1 103

#define ID\_FILE\_SAVE 40001

#define ID\_FILE\_LOAD 40002

#define ID\_HELP 40007

#define ID\_EXIT 40008

#define ID\_ABOUT 40009

#define ID\_FONT\_FONTSIZE 40010

#define ID\_FONTSIZE\_12 40012

#define ID\_FONTSIZE\_15 40013

#define ID\_FONTSIZE\_18 40014

#define ID\_FONTSIZE\_20 40015

#define ID\_FONT\_FONT 40017

#define ID\_FONT\_TIMESNEWROMAN 40018

#define ID\_FONTSIZE\_22 40020

#define ID\_FONTSIZE\_28 40021

#define ID\_FONTSIZE\_32 40022

#define ID\_FONT\_CASCADIA\_CODE 40023

#define ID\_FONT\_ARIAL 40024

#define ID\_FONT\_COURIERNEW 40025

#define ID\_FONT\_CALIBRI 40026

#define ID\_FORMAT\_FONTCOLOR 40027

#define ID\_FORMAT\_BACKGROUNDCOLOR 40028

#define ID\_FONTSIZE\_14 40032

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NEXT\_RESOURCE\_VALUE 104

#define \_APS\_NEXT\_COMMAND\_VALUE 40033

#define \_APS\_NEXT\_CONTROL\_VALUE 1001

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

**Файл main.cpp**

#include <Windows.h>

#include "headers.h"

#include <string>

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

{

WNDCLASSEX windowClass = CreateMainWindowClass(hInstance, WindowProc);

if (not RegisterClassEx(&windowClass)) {

MessageBox(NULL, L"Window Registration Failed!", L"Error!", MB\_ICONEXCLAMATION | MB\_OK);

return EXIT\_FAILURE;

}

hMainWindow = CreateMainWindow(hInstance);

if (hMainWindow == NULL) {

MessageBox(NULL, L"Window Creation Failed!", L"Error!", MB\_ICONEXCLAMATION | MB\_OK);

return EXIT\_FAILURE;

}

HICON hSmallIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_ICON1));

SendMessage(hMainWindow, WM\_SETICON, ICON\_SMALL, (LPARAM)hSmallIcon);

ShowWindow(hMainWindow, nCmdShow);

UpdateWindow(hMainWindow);

MSG message;

while (GetMessage(&message, NULL, 0, 0) > 0)

{

TranslateMessage(&message);

DispatchMessage(&message);

}

return message.wParam;

}

LRESULT CALLBACK WindowProc(HWND hWindow, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message) {

case WM\_COMMAND: {

if (not contentAlreadyChanged and IsTextFieldChanged(wParam, lParam))

AppendAsteriskToFilename();

else

ProcessWMCommand(hWindow, wParam, lParam);

return 0;

}

case WM\_CLOSE:

if (contentAlreadyChanged and MessageBox(hWindow, L"Do you want to save your changes?", L"Attention", MB\_YESNO) == IDYES) {

SaveFile(hWindow);

}

DestroyWindow(hWindow);

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_CREATE: {

hTextField = CreateTextField(hWindow);

if (hTextField == NULL) {

MessageBox(NULL, L"Edit control creation failed!", L"Error", MB\_ICONERROR);

return EXIT\_FAILURE;

}

SendMessage(hTextField, EM\_SETEVENTMASK, 0, ENM\_CHANGE);

UpdateFont();

return 0;

}

case WM\_SIZE: {

int newWidth = LOWORD(lParam);

int newHeight = HIWORD(lParam);

MoveWindow(hTextField, 0, 0, newWidth, newHeight, TRUE);

break;

}

default:

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

}

}