Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №2

на тему

**РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN32 И GDI**

Студент В. А. Сидорко

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 10](#_Toc146752071)

[Список использованных источников 11](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 12](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Изучить принципы расширенного использования Win32 и GDI, научиться формировать сложные изображения, создавать и использовать элементы управления, обрабатывать различные сообщения, изучить механизм перехвата сообщений (winhook). Для достижения цели будет создано оконное приложение для чтения и редактирования текстовых документов с поддержкой настраиваемых тем оформления (стили текста, фон текста).

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Win32 API* – это исходная платформа для собственных *Windows* приложений на языке C/C++, которым требуется прямой доступ к *Windows* и оборудованию. Для этого *Win32 AP*I предоставляет множество 32-разрядных функций, структур данных и числовых констант, следующих соглашениям языка Си.

*GDI* (*Graphics Device Interface*) – интерфейс графических устройств *Microsoft Windows*, позволяющий приложениям использовать графику и отформатированный текст как на видео-дисплее, так и на принтере. Приложения на основе *Windows* не обращаются к графическому оборудованию напрямую. Вместо этого оно обращается к функциям *GDI*,а *GDI* транслирует эти обращения к программным драйверам физических устройств, обеспечивая аппаратную независимость приложений.

*GDI* предоставляет следующую функциональность: вывод графических объектов на экран или принтер, вывод в метафайл, вывод текста различными шрифтами, в том числе *[TrueType](https://ru.wikipedia.org/wiki/TrueType" \o "TrueType)* и *[OpenType](https://ru.wikipedia.org/wiki/OpenType" \o "OpenType)*, а также шрифтами, вшитыми в принтер, набор операций с битовыми картами (битмапами), набор операций векторной графики

Взаимодействие приложения с *GDI* осуществляется при непременном участии еще одного посредника – контекста устройства.

*Контекст устройства* (*device context*) – это внутренняя структура данных, которая определяет набор графических объектов и ассоциированных с ними атрибутов, а также графических режимов, влияющих на вывод. Если необходимо рисовать на устройстве графического вывода (экране дисплея или принтере), то сначала нужно получить дескриптор контекста устройства. Возвращая этот дескриптор после вызова соответствующих функций, *Windows* тем самым предоставляет разработчику право на использование данного устройства. После этого дескриптор контекста устройства передается как параметр в функции *GDI*, чтобы идентифицировать устройство, на котором должно выполняться рисование.

При выводе текста приложения не работают напрямую с физическими шрифтами. С физическими шрифтами работают шрифтовые драйверы, находящиеся в системе на одном уровне с драйверами графических устройств. В программах же используются так называемые логические шрифты. Логический шрифт представляет собой объект *GDI*, описывающий требования к шрифту со стороны приложения. *GDI* анализирует запрошенные параметры и подбирает наиболее подходящий шрифт из тех, которые зарегистрированы в системе. Объект логического шрифта находится под управлением *GDI* вместе с другими логическими объектами. Приложения работают с логическими шрифтами только через их дескрипторы, имеющие тип *HFONT*.

Система *Windows* поддерживает цветовое пространство *RGB* и пространство индексов палитры, соответствующее базовым возможностям видеоадаптера. Цвет кодируется в любом из этих пространств при помощи типа данных *COLORREF*, представляющего четырехбайтное значение, где старший байт задает один из трех возможных форматов: 0 — *RGB*, 1 — *PALETTEINDEX*, 2 — *PALETTERGB*.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение для чтения и редактирования текстовых документов с возможностью настройки стилей текста и цвета фона (рисунок 1).

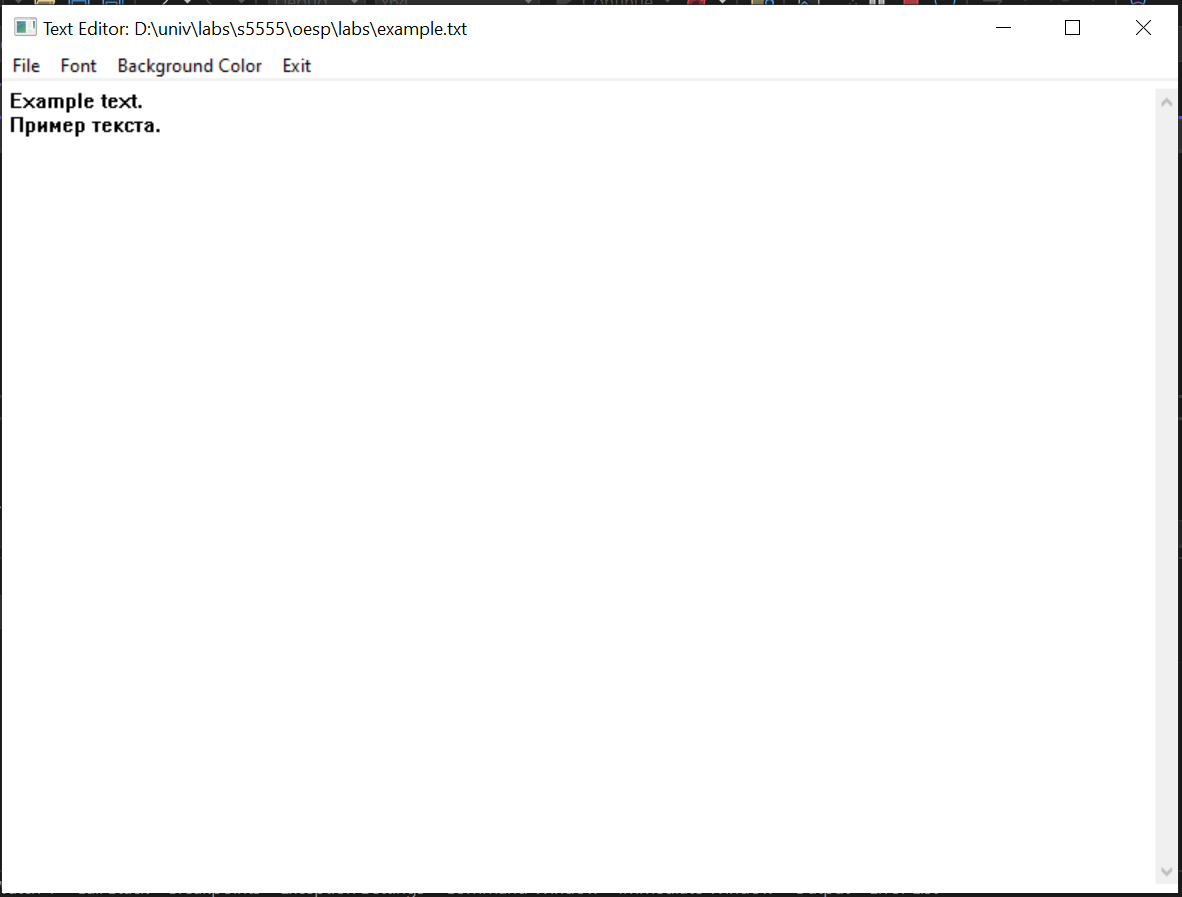


Рисунок 1 – Окно приложения

Меню окна содержит кнопки *«File»* (загрузка и сохранение файлов)и *«Exit»* (завершение работы программы), а также кнопки настройки тем оформления: *«Font»* и *«Background Color»*. *«Font»* создает окно настройки стилей текста (рисунок 2).

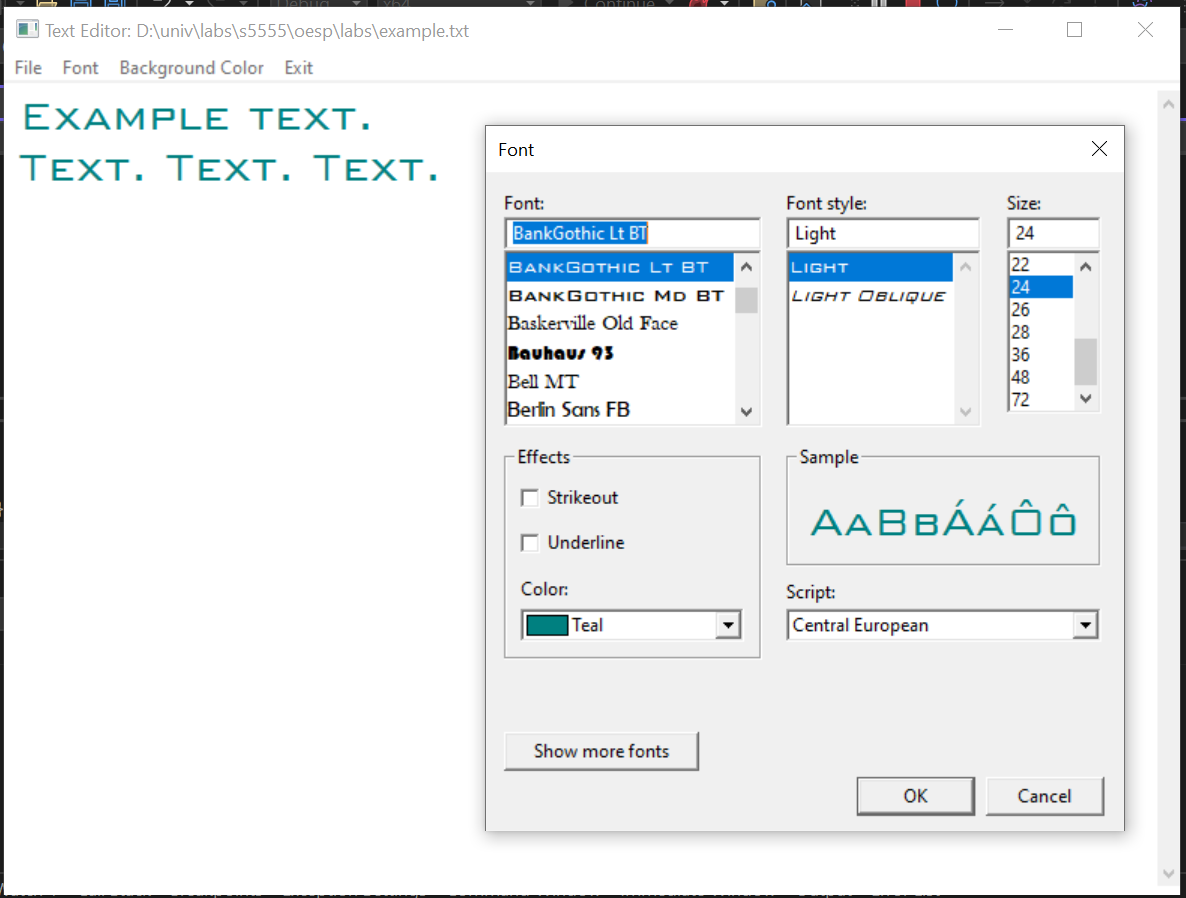


Рисунок 2– Настройка стилей текста

*«Background Color»* – окно настройки цвета фона (рисунок 3).

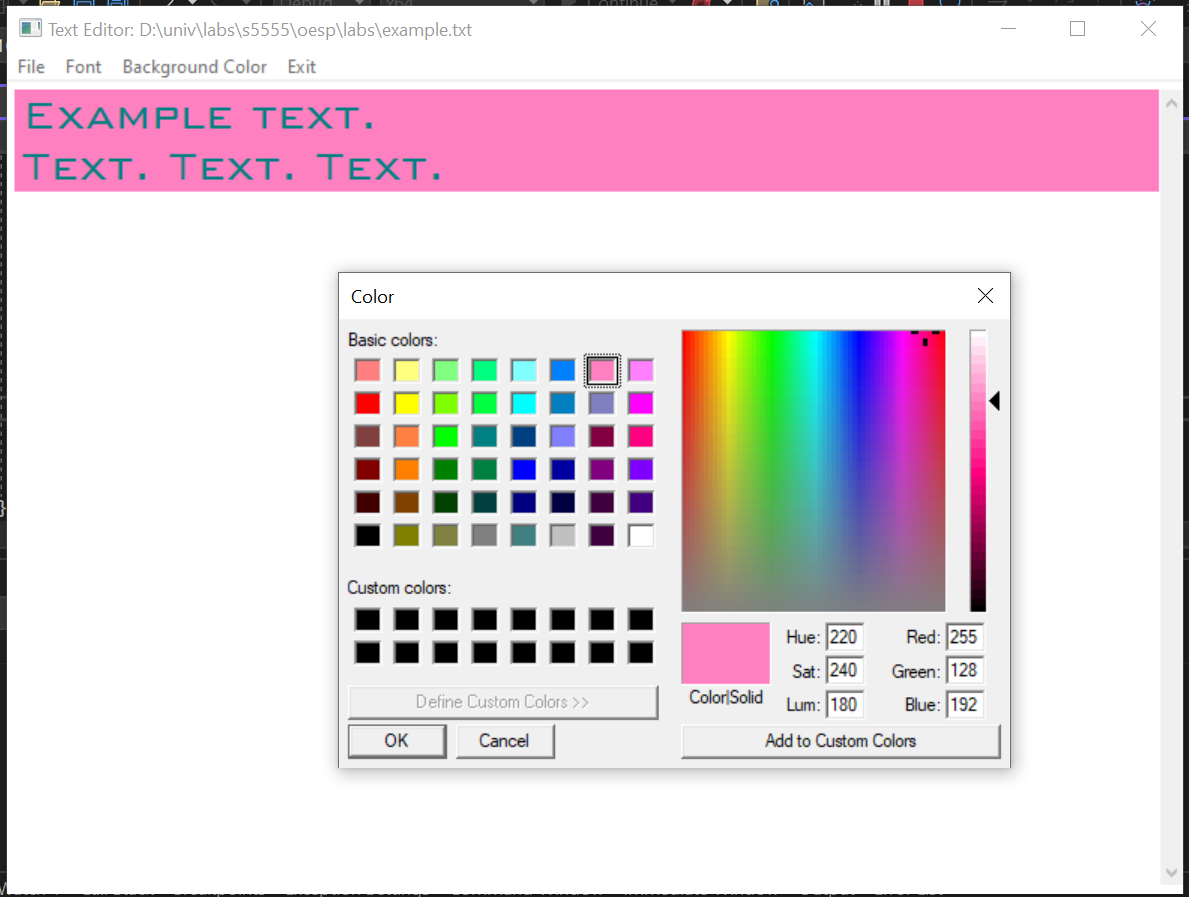


Рисунок 3– Настройка цвета фона

# ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены и применены на практике знания о принципах расширенного использования Win32 и GDI для отображения текстовых данных и обработке различных сообщений. В результате проделанной работы было создано оконное приложение для чтения, редактирования и сохранения текстовых документов с возможностью настройки стиля текста и цвета фона.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник по программированию для API Win32 [Электронный ресурс]. – Режим доступа : <https://learn.microsoft.com/ru-ru/windows/win32/api/>.
2. Щупак, Ю.А. Win32 API. Разработка приложений для Windows /Ю.А. Щупак. – СПб : Питер, 2008. – 592 с.
3. Text and Fonts in Win32 API [Электронный ресурс]. – Режим доступа : http://www.winprog.org/tutorial/fonts.html.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл main.cpp

#include <Windows.h>

#include <WinUser.h>

#define OnMenuOpen 1

#define OnMenuSave 2

#define OnMenuExit 3

#define OnChangeFont 4

#define OnChangeBkColor 5

WNDCLASS NewWindowClass(HINSTANCE hInst, LPCWSTR name, WNDPROC procedure);

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

void AddMenus(HWND hWnd);

void AddWidgets(HWND hWnd);

void InitOpenParams(HWND hWnd);

BOOL LoadTextToEdit(HWND hEdit, LPCWSTR pszFileName);

BOOL SaveFileFromEdit(HWND hEdit, LPCTSTR pszFileName);

HGDIOBJ SetFont(HWND hWnd);

HGDIOBJ SetBkColorEdit(HWND hWnd);

HWND hEditControl;

OPENFILENAME ofn;

char szFileName[MAX\_PATH];

HGDIOBJ g\_hfCurrentFont = GetStockObject(DEFAULT\_GUI\_FONT);

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

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

COLORREF crCustomBkColors[16] = { 0 };

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd) {

WNDCLASS MainClass = NewWindowClass(hInst, L"Main Window Class", WndProc);

if (!RegisterClass(&MainClass)) { return -1; }

MSG MainMessage = { 0 };

CreateWindow(L"Main Window Class", L"Text Editor: New Document",

WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 50, 800, 600, NULL, NULL, hInst, NULL);

while (GetMessage(&MainMessage, NULL, NULL, NULL)) {

TranslateMessage(&MainMessage);

DispatchMessage(&MainMessage);

}

}

WNDCLASS NewWindowClass(HINSTANCE hInst, LPCWSTR name, WNDPROC procedure) {

WNDCLASS windowClass = {};

windowClass.hInstance = hInst;

windowClass.lpszClassName = name;

windowClass.lpfnWndProc = procedure;

return windowClass;

}

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

{

switch (uMsg)

{

case WM\_COMMAND:

switch (wParam)

{

case OnMenuOpen:

if (GetOpenFileName(&ofn)) {

if (!LoadTextToEdit(hEditControl, ofn.lpstrFile)) {

MessageBox(hWnd, L"Failed to open file", L"Error", MB\_OK);

}

else {

WCHAR pszNewTitle[MAX\_PATH + 14] = { 0 };

wcscpy\_s(pszNewTitle, L"Text Editor: ");

SetWindowText(hWnd, lstrcat(pszNewTitle, ofn.lpstrFile));

}

}

break;

case OnMenuSave:

if (GetSaveFileName(&ofn)) {

if (!SaveFileFromEdit(hEditControl, (LPCTSTR)szFileName)) {

MessageBox(hWnd, L"Failed to save file", L"Error", MB\_OK);

}

else {

WCHAR pszNewTitle[MAX\_PATH + 14] = { 0 };

wcscpy\_s(pszNewTitle, L"Text Editor: ");

SetWindowText(hWnd, lstrcat(pszNewTitle, ofn.lpstrFile));

}

}

break;

case OnMenuExit:

DestroyWindow(hWnd);

break;

case OnChangeFont:

SendMessage(hEditControl, WM\_SETFONT, (WPARAM)SetFont(hWnd), TRUE);

break;

case OnChangeBkColor:

SendMessage(hEditControl, WM\_SETFONT, (WPARAM)SetBkColorEdit(hWnd), TRUE);

SendMessage(hEditControl, WM\_SETFONT, (WPARAM)g\_hfCurrentFont, TRUE);

break;

default:

break;

}

break;

case WM\_CREATE:

AddMenus(hWnd);

AddWidgets(hWnd);

InitOpenParams(hWnd);

break;

case WM\_CTLCOLOREDIT:

SetTextColor((HDC)wParam, crTextColor);

SetBkColor((HDC)wParam, crBkColor);

return (LRESULT)GetStockObject(DC\_BRUSH);

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

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

}

void AddMenus(HWND hWnd) {

HMENU RootMenu = CreateMenu();

HMENU FileMenu = CreateMenu();

AppendMenu(RootMenu, MF\_POPUP, (UINT\_PTR)FileMenu, L"File");

AppendMenu(FileMenu, MF\_STRING, OnMenuOpen, L"Open");

AppendMenu(FileMenu, MF\_STRING, OnMenuSave, L"Save");

AppendMenu(RootMenu, MF\_STRING, OnChangeFont, L"Font");

AppendMenu(RootMenu, MF\_STRING, OnChangeBkColor, L"Background Color");

AppendMenu(RootMenu, MF\_STRING, OnMenuExit, L"Exit");

SetMenu(hWnd, RootMenu);

}

void AddWidgets(HWND hWnd) {

hEditControl = CreateWindowA("edit", NULL,

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

5, 5, 780, 530, hWnd, NULL, NULL, NULL);

}

void InitOpenParams(HWND hWnd) {

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"\*.txt\0";

ofn.lpstrFile = (LPWSTR)szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_EXPLORER | OFN\_PATHMUSTEXIST | OFN\_HIDEREADONLY;

ofn.lpstrDefExt = L"txt";

}

BOOL LoadTextToEdit(HWND hEdit, LPCWSTR pszFileName) {

HANDLE hFile;

BOOL bSuccess = false;

hFile = CreateFile(pszFileName, GENERIC\_READ,

FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

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

DWORD dwFileSize = GetFileSize(hFile, NULL);

if (dwFileSize != 0xFFFFFFFF) {

LPSTR pszFileText = (LPSTR)GlobalAlloc(GPTR, dwFileSize + 1);

if (pszFileText != NULL) {

DWORD dwRead;

if (ReadFile(hFile, pszFileText, dwFileSize, &dwRead, NULL)) {

pszFileText[dwFileSize] = { 0 };

if (SetWindowTextA(hEdit, pszFileText)) {

bSuccess = true;

}

}

GlobalFree(pszFileText);

}

}

CloseHandle(hFile);

}

return bSuccess;

}

BOOL SaveFileFromEdit(HWND hEdit, LPCTSTR pszFileName) {

HANDLE hFile;

BOOL bSuccess = false;

hFile = CreateFile(pszFileName, GENERIC\_WRITE, 0,

NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

DWORD dwTextLength = GetWindowTextLengthA(hEdit);

DWORD dwBufferSize = dwTextLength + 1;

LPSTR pszText = (LPSTR)GlobalAlloc(GPTR, dwBufferSize);

if (pszText != NULL) {

if (GetWindowTextA(hEdit, pszText, dwBufferSize) == dwTextLength) {

DWORD dwWritten;

if (WriteFile(hFile, pszText, dwTextLength, &dwWritten, NULL)) {

bSuccess = true;

}

}

GlobalFree(pszText);

}

CloseHandle(hFile);

}

return bSuccess;

}

HGDIOBJ SetFont(HWND hWnd) {

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf;

GetObject(g\_hfCurrentFont, sizeof(LOGFONT), &lf);

cf.Flags = CF\_EFFECTS | CF\_INITTOLOGFONTSTRUCT | CF\_SCREENFONTS;

cf.hwndOwner = hWnd;

cf.lpLogFont = &lf;

cf.rgbColors = crTextColor;

if (ChooseFont(&cf)) {

HFONT hf = CreateFontIndirect(&lf);

if (hf) {

g\_hfCurrentFont = hf;

}

// set current text color

crTextColor = cf.rgbColors;

}

else

{

MessageBox(hWnd, L"Font creation failed!", L"Error", MB\_OK | MB\_ICONEXCLAMATION);

}

return g\_hfCurrentFont;

}

HGDIOBJ SetBkColorEdit(HWND hWnd) {

CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };

cc.Flags = CC\_RGBINIT | CC\_FULLOPEN | CC\_ANYCOLOR;

cc.hwndOwner = hWnd;

cc.rgbResult = crBkColor;

cc.lpCustColors = crCustomBkColors;

if (ChooseColor(&cc))

{

crBkColor = cc.rgbResult;

}

return (HGDIOBJ)crBkColor;

}