Модуль №5

Занятие №2

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Дополнительные немодальные диалоги.
3. Стандартные диалоги «Найти» и «Заменить».
4. Практическая часть.
5. Подведение итогов.
6. Домашнее задание.

1. Повторение пройденного материала

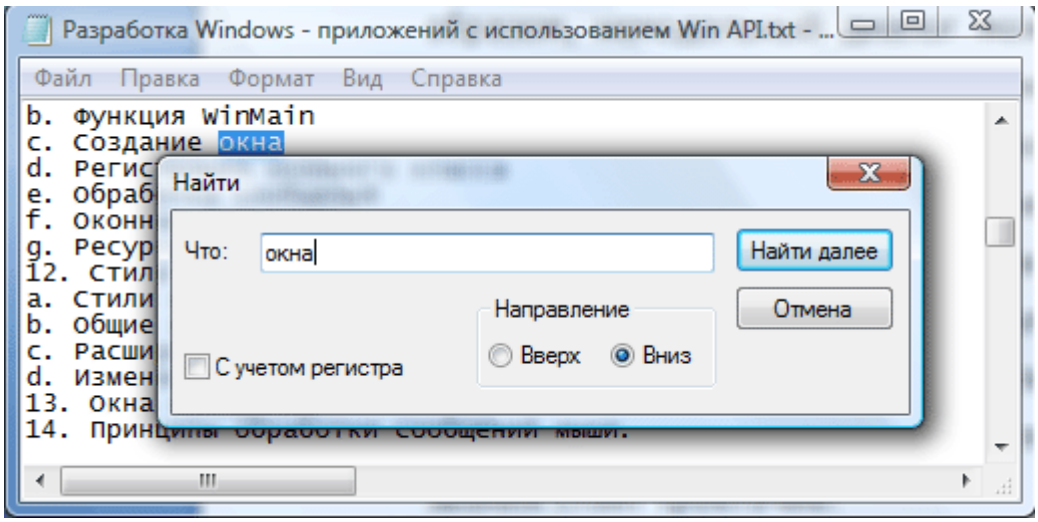
Данное занятие необходимо начать с краткого повторения материала предыдущего занятия. При общении со слушателями можно использовать следующие контрольные вопросы:

1. Для чего предназначены дополнительные диалоговые окна?
2. Какой отличительной особенностью обладает модальное диалоговое окно?
3. Какая функция API позволяет отобразить модальное диалоговое окно?
4. В чем заключается механизм обмена данными между главным окном приложения и дополнительным модальным диалоговым окном?
5. Посредством какой функции можно получить дескриптор родительского окна?

6. Какая функция API предназначена для создания и отображения модального диалога «**Открыть**»?
7. Какая функция API предназначена для создания и отображения модального диалога «**Сохранить**»?

2. Дополнительные немодальные диалоги

Напомнить слушателям, что отличительной особенностью дополнительного модального окна является блокирование доступа к остальным окнам приложения до тех пор, пока модальное окно не будет закрыто. В отличие от модального окна немодальный диалог не задерживает выполнение программы, то есть для ее продолжения не требуется завершение диалога. При этом допускается переключение между диалогом и другими окнами приложения. Таким образом, немодальный диалог может получать и терять фокус ввода. Диалоги этого типа предпочтительней использовать в тех случаях, когда они содержат элементы управления, которые должны быть в любой момент доступны пользователю. Типичным примером немодального диалога может послужить диалоговое окно «Найти» приложения «Блокнот».



В рамках данного занятия необходимо рассмотреть со слушателями основные принципы взаимодействия приложения с дополнительными немодальными диалогами.

Следует отметить, что для создания и отображения дополнительного немодального диалогового окна требуется выполнить следующие действия:

- определить диалог в файле описания ресурсов;
- определить в приложении диалоговую процедуру, обеспечивающую обработку сообщений для дополнительного диалогового окна;
- вызвать функцию API **CreateDialog** для создания немодального диалога;
- установить значение **True** для свойства **Visible** в свойствах шаблона диалога для отображения немодального окна (другой способ отображения немодального окна - вызов функции API **ShowWindow**);

Рассмотреть со слушателями следующее приложение, в котором демонстрируются основные принципы обмена данными между главным окном приложения и дополнительным немодальным диалоговым окном.

```
                                    // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"


                                // MainModalDialog.h
#pragma once
#include "header.h"

class CMainModalDialog
{
public:
      CMainModalDialog(void);
public:
      ~CMainModalDialog(void);
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
```

```
	static CMainModalDialog* ptr;
	BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
	void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
	void Cls_OnClose(HWND hwnd);
	HWND hEdit1, hStatic1;
};
```

```
                          // MainModalDialog.cpp
#include "MainModalDialog.h"
#include "ModelessDlg.h"

CMainModalDialog* CMainModalDialog::ptr = NULL;

CMainModalDialog::CMainModalDialog(void)
{
	ptr = this;
}

CMainModalDialog::~CMainModalDialog(void)
{

}

void CMainModalDialog::Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, IDCANCEL);
}

BOOL CMainModalDialog::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                        LPARAM lParam)
{
	hEdit1 = GetDlgItem(hwnd, IDC_EDIT1);
	hStatic1 = GetDlgItem(hwnd, IDC_STATIC1);
	SetWindowText(hEdit1, TEXT("Передача данных дополнительному диалогу!"));
	return TRUE;
}

void CMainModalDialog::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                     UINT codeNotify)
{
	if(id == IDC_BUTTON1)
	{
		// Проверим, открыто ли дополнительное немодальное окно
		if(CModelessDialog::hAddDialog)
		{
			//Активизируем дополнительное немодальное окно
			SetForegroundWindow(CModelessDialog::hAddDialog);
			return;
		}
		CModelessDialog dlg;
		// Создаем немодальное диалоговое окно
		CModelessDialog::hAddDialog = CreateDialog(GetModuleHandle(NULL),
		MAKEINTRESOURCE(IDD_DIALOG2), hwnd, CModelessDialog::DlgProc);
		// Отображаем окно
		ShowWindow(CModelessDialog::hAddDialog, SW_RESTORE);
		TCHAR buffer[200];
		// Получаем текст с текстового поля ввода
		GetWindowText(hEdit1, buffer, 200);
```

```
            // Отображаем текст на статике дополнительного диалога
            SetWindowText(dlg.hStatic, buffer);
      }
}

BOOL CALLBACK CMainModalDialog::DlgProc(HWND hwnd, UINT message,
                                        WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
      }
      return FALSE;
}
```

```
                              // ModelessDlg.h
#pragma once
#include "header.h"

class CModelessDialog
{
public:
      CModelessDialog(void);
public:
      ~CModelessDialog(void);
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
      static CModelessDialog* ptr;
      BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
      void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
      void Cls_OnClose(HWND hwnd);
      HWND hStatic;
      static HWND hAddDialog; // дескриптор дополнительного диалога
};
```

```
                              // ModelessDlg.cpp
#include "ModelessDlg.h"

CModelessDialog* CModelessDialog::ptr = NULL;
HWND CModelessDialog::hAddDialog = NULL;

CModelessDialog::CModelessDialog(void)
{
      ptr = this;
}

CModelessDialog::~CModelessDialog(void)
{
}

void CModelessDialog::Cls_OnClose(HWND hwnd)
{
      // Разрушаем немодальное диалоговое окно
      DestroyWindow(hwnd);
      hAddDialog = NULL;
}
```

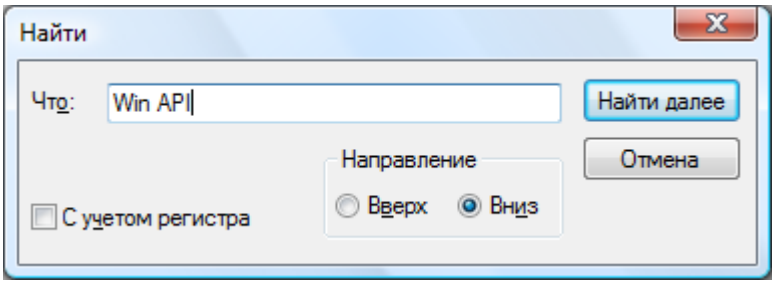
```
BOOL CModelessDialog::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                       LPARAM lParam)
{
      hStatic = GetDlgItem(hwnd, IDC_STATIC1);
      SetWindowText(hwnd, TEXT("Дополнительный немодальный диалог"));
      SetWindowText(GetDlgItem(hwnd, IDC_EDIT1),
                              TEXT("Передача данных главному диалогу!"));
      return TRUE;
}

void CModelessDialog::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                    UINT codeNotify)
{
      if(id == IDOK)
      {
            TCHAR buffer[200];
            // Получаем текст с текстового поля ввода
            GetWindowText(GetDlgItem(hwnd, IDC_EDIT1), buffer, 200);
            // Получаем дескриптор родительского (главного) окна
            HWND hParent = GetParent(hwnd);
            // Получаем дескриптор статика главного диалога
            HWND hStatic = GetDlgItem(hParent, IDC STATIC1);
            // Отображаем текст на статике главного диалога
            SetWindowText(hStatic, buffer);
      }
      else if(id == IDCANCEL)
      {
            // Разрушаем немодальное диалоговое окно
            DestroyWindow(hwnd);
            hAddDialog = NULL;
      }
}

BOOL CALLBACK CModelessDialog::DlgProc(HWND hwnd, UINT message,
                                       WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
      }
      return FALSE;
}

                           // ModelessDialogApp.cpp
#include "MainModalDialog.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                  int nCmdShow)
{
      CMainModalDialog dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        CMainModalDialog::DlgProc);
}
```

В вышеприведенном приложении для активизации дополнительного окна была применена функция WinAPI **SetForegroundWindow**.

```
BOOL SetForegroundWindow(HWND hWnd);
```

2. Стандартные диалоги «Найти» и «Заменить»

Стандартный диалог «**Найти**» предоставляет пользователю возможность поиска в тексте определенной последовательности символов.

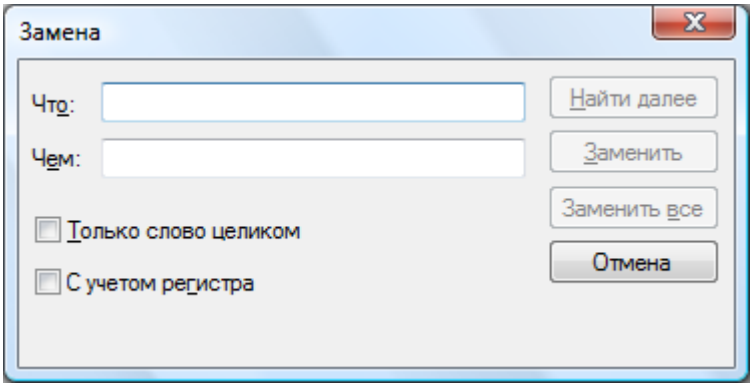


Для создания и отображения немодального диалога «**Найти**» предназначена функция API **FindText**.

```
HWND FindText(
   LPFINDREPLACE lpfr // указатель на структуру FINDREPLACE, которая содержит
// информацию, используемую для инициализации диалога «Найти»
);
```

В случае успешного создания диалогового окна данная функция вернет дескриптор диалога.

Стандартный диалог «**Заменить**» предоставляет пользователю возможность поиска в тексте определенной последовательности символов и замены ее на другую последовательность символов.



Для создания и отображения немодального диалога «**Заменить**» предназначена функция API **ReplaceText**.

```
HWND ReplaceText(
   LPFINDREPLACE lpfr // указатель на структуру FINDREPLACE, которая содержит
// информацию, используемую для инициализации диалога «Найти»
);
```

В случае успешного создания диалогового окна данная функция вернет дескриптор диалога.

Обе вышеописанные функции в качестве параметра принимают указатель на структуру **FINDREPLACE**, которая содержит информацию, используемую для инициализации диалога.

```
typedef struct {
    DWORD lStructSize; // длина структуры в байтах
    HWND hwndOwner; // дескриптор окна, которое владеет диалогом
    HINSTANCE hInstance; // дескриптор приложения, если установлен флаг
// FR_ENABLETEMPLATE, либо дескриптор объекта памяти, содержащего шаблон
// диалогового окна, если установлен флаг FR_ENABLETEMPLATEHANDLE
    DWORD Flags; // набор флагов, используемых для инициализации диалога
    LPTSTR lpstrFindWhat; // указатель на буфер, содержащий строку для поиска
    LPTSTR lpstrReplaceWith; // указатель на буфер, содержащий строку для
// замены
    WORD wFindWhatLen; // размер в байтах буфера, содержащего строку для
// поиска
    WORD wReplaceWithLen; // размер в байтах буфера, содержащего строку для
// замены
```

```
    LPARAM lCustData; // данные, которые передаются в HOOK-процедуру
    LPFRHOOKPROC lpfnHook; // адрес HOOK-процедуры (должен быть включен флаг
// FR_ENABLEHOOK)
    LPCTSTR lpTemplateName; // Указатель на строку, которая именует  ресурс
// шаблона диалога в модуле, идентифицированном полем hInstance
} FINDREPLACE, *LPFINDREPLACE;
```

Поле **Flags** структуры **FINDREPLACE** представляет собой комбинацию из одного или нескольких флагов, объединенных с помощью поразрядной операции «ИЛИ», и используемых для инициализации диалога. Ниже представлены наиболее часто используемые флаги.

- **FR_DOWN** - если флаг установлен, то из радио-кнопок направления выбрана кнопка «**Вниз**» (**Down**), которая указывает, что пользователь желает искать от текущего положения каретки в тексте до конца документа. Если флажок **FR_DOWN** не установлен, то выбрана кнопка «**Вверх**» (**Up**), что означает, что пользователь желает осуществлять поиск к началу документа.
- **FR_MATCHCASE** - если флаг установлен, то в чекбоксе «**С учетом регистра**» (**Match Case**) стоит галочка, показывая, что пользователь желает, чтобы поиск был чувствителен к регистру. Если **FR_MATCHCASE** не установлен, то в чекбоксе галочки нет, так что поиск должен быть не чувствительным к регистру.
- **FR_WHOLEWORD** - если флаг установлен, то в чекбоксе «**Только слово целиком**» (**Match Whole Word Only**) стоит галочка, показывая, что будет осуществляться поиск только для целого слова, которое соответствует поисковой последовательности символов. Если **FR_WHOLEWORD** не установлен, то в чекбоксе галочки нет, и будет осуществляться поиск фрагмента слова, который соответствует поисковой последовательности символов.

Важно отметить, что перед созданием диалогового окна «**Найти**» или «**Заменить**» необходимо вызвать функцию API

RegisterWindowMessage, чтобы получить идентификатор зарегистрированного сообщения **FINDMSGSTRING**.

```
UINT RegisterWindowMessage(
       LPCTSTR lpString // указатель на строку, идентифицирующую сообщение,
// которое необходимо зарегистрировать
);
```

В дальнейшем можно использовать этот идентификатор, чтобы определять и обрабатывать сообщения, посылаемые из немодального диалогового окна.

При щелчке мышью по кнопке немодального диалогового окна «**Найти далее**» (**Find Next**), «**Заменить**» (**Replace**) или «**Заменить все**» (**Replace All**), процедура диалогового окна передаст сообщение **FINDMSGSTRING** в оконную процедуру окна владельца (главного диалога). Это обусловлено тем, что при создании немодального диалогового окна в поле **hwndOwner** структуры **FINDREPLACE** указывается дескриптор окна владельца.

Параметр **lParam** сообщения **FINDMSGSTRING** указывает на структуру **FINDREPLACE**, которая была определена при создании немодального диалогового окна. В сообщении **FINDMSGSTRING** элемент **Flags** структуры **FINDREPLACE** включает в себя один из ниже перечисленных флажков, чтобы указать на событие, которое вызвало сообщение:

- **FR_DIALOGTERM** - диалоговое окно закрывается. После того как главное окно обработает это сообщение, дескриптор немодального диалогового окна станет недопустимым.
- **FR_FINDNEXT** – произведен щелчок мышью по кнопке «**Найти далее**» в диалоговом окне «**Найти**» или «**Заменить**». При этом поле **lpstrFindWhat** определяет последовательность символов для поиска.

- **FR_REPLACE** - произведен щелчок мышью по кнопке «**Заменить**» в диалоговом окне «**Заменить**». При этом поле **lpstrFindWhat** устанавливает последовательность символов, которую надо заменить, а поле **lpstrReplaceWith** устанавливает последовательность символов, на которую надо заменить.
- **FR_REPLACEALL** - произведен щелчок мышью по кнопке «**Заменить все**» в диалоговом окне «**Заменить**». При этом поле **lpstrFindWhat** устанавливает последовательность символов, которую надо заменить, а поле **lpstrReplaceWith** устанавливает последовательность символов, на которую надо заменить.

В качестве примера, демонстрирующего использование стандартных диалогов «**Найти**» и «**Заменить**», рассмотреть со слушателями следующее приложение.

```
                                    // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"
                                  // MainDialog.h
#pragma once
#include "header.h"

class CMainDialog
{
public:
      CMainDialog(void);
public:
      ~CMainDialog(void);
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
      static CMainDialog* ptr;
      void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
      void Cls_OnClose(HWND hwnd);
      BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
      void OnFind();
      void OnReplace();
      void MessageFromFindReplace();
      TCHAR bufFind[100], alltext[65536], bufReplace[100];
      HWND hEdit, hFR, hDialog;
      FINDREPLACE fr;
};
                                  // MainDialog.cpp
#include "MainDialog.h"
```

```
// идентификатор зарегистрированного сообщения FINDMSGSTRING
UINT WM_FR = RegisterWindowMessage(FINDMSGSTRING);

CMainDialog* CMainDialog::ptr = NULL;

CMainDialog::CMainDialog(void)
{
	ptr = this;
	hFR = NULL;
	ZeroMemory(&fr, sizeof(fr));
}

CMainDialog::~CMainDialog(void)
{
}

void CMainDialog::Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, 0);
}

BOOL CMainDialog::Cls OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
{
	hDialog = hwnd;
	hEdit = GetDlgItem(hwnd, IDC_EDIT1);
	return TRUE;
}

void CMainDialog::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                UINT codeNotify)
{
	if(id == IDC_FIND)
		OnFind();
	else if(id == IDC_REPLACE)
		OnReplace();
}

void CMainDialog::OnFind()
{
	// Проверим, открыто ли окно поиска
	if(hFR)
	{
		//Активизируем окно поиска
		SetForegroundWindow(hFR);
		return;
	}
	// обнуляем структуру FINDREPLACE
	ZeroMemory(&fr, sizeof(fr));
	DWORD start, end;
	// получим весь текст, находяшийся в текстовом поле ввода
	GetWindowText(hEdit, alltext, 65536);
	// получим границы выделения фрагмента текста
	SendMessage(hEdit, EM_GETSEL, WPARAM(&start), LPARAM(&end));
	// скопируем в буфер выделенный фрагмент текста
	_tcsncpy(bufFind, alltext+start, end-start);
	bufFind[end - start] = TEXT('\0');
	fr.lStructSize = sizeof(fr);
```

```
	// главный диалог является окном-владелецем
	fr.hwndOwner = hDialog;
	// указатель на буфер, содержащий строку для поиска
	fr.lpstrFindWhat = bufFind;
	fr.wFindWhatLen = 100;
	// поиск от текущего положения каретки в тексте до конца документа
	fr.Flags = FR_DOWN;
	// отображаем диалог Найти
	hFR = FindText(&fr);
}


void CMainDialog::OnReplace()
{
	// Проверим, открыто ли окно замены
	if(hFR)
	{
		// Активизируем окно замены
		SetForegroundWindow(hFR);
		return;
	}
	// обнуляем структуру FINDREPLACE
	ZeroMemory(&fr, sizeof(fr));
	DWORD start, end;
	// обнуляем буфер, предназначенный для хранения замещающей строки
	ZeroMemory(&bufReplace, sizeof(bufReplace));
	// получим весь текст, находяшийся в текстовом поле ввода
	GetWindowText(hEdit, alltext, 65536);
	// получим границы выделения фрагмента текста
	SendMessage(hEdit, EM_GETSEL, WPARAM(&start), LPARAM(&end));
	// скопируем в буфер выделенный фрагмент текста
	_tcsncpy(bufFind, alltext + start, end - start);
	bufFind[end - start] = TEXT('\0');
	fr.lStructSize = sizeof(fr);
	// главный диалог является окном-владелецем
	fr.hwndOwner = hDialog;
	// указатель на буфер, содержащий строку для поиска
	fr.lpstrFindWhat = bufFind;
	fr.wFindWhatLen = 100;
	// указатель на буфер, содержащий строку для замены
	fr.lpstrReplaceWith = bufReplace;
	fr.wReplaceWithLen = 100;
	// поиск от текущего положения каретки в тексте до конца документа
	fr.Flags = FR_DOWN;
	// отображаем диалог Заменить
	hFR = ReplaceText(&fr);
}

void CMainDialog::MessageFromFindReplace()
{

	if(fr.Flags & FR_REPLACEALL)
		MessageBox(hDialog, TEXT("Нажата кнопка \"Заменить всё\""),
				TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
	if(fr.Flags & FR_REPLACE)
	{
		MessageBox(hDialog, TEXT("Нажата кнопка \"Заменить\""),
				TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
```

```
            // заменяем выделенный фрагмент текста на строку, находящуюся в
            // буфере bufReplace
            SendMessage(hEdit, EM_REPLACESEL,
                        WPARAM(TRUE),(LPARAM)bufReplace);
      }
      if(fr.Flags & FR_FINDNEXT)
      {
            MessageBox(hDialog, TEXT("Нажата кнопка \"Найти далее\""),
                        TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
            DWORD Start, End;
            // выполняем поиск искомой строки
            TCHAR * p = _tcsstr(alltext, bufFind);

            if(p)
            {
                  Start = p - alltext;
                  End = Start + _tcslen(bufFind);
                  // выделяем найденную строку
                  SendMessage(hEdit, EM_SETSEL, Start, End);
            }
      }
      if(fr.Flags&FR DIALOGTERM)
      {
            hFR = NULL;
            MessageBox(hDialog, TEXT("Закрывается диалог поиска и замены!"),
                        TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
            return;
      }
      if(fr.Flags & FR_MATCHCASE)
            MessageBox(hDialog, TEXT("Установлен флажок регистрозависимости"),
                        TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
      if(fr.Flags & FR_WHOLEWORD)
           MessageBox(hDialog, TEXT("Установлен флажок поиска слова целиком"),
                        TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
      if(fr.Flags & FR_DOWN)
           MessageBox(hDialog, TEXT("Выбран режим поиска в направлении вниз"),
                        TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
      else
          MessageBox(hDialog, TEXT("Выбран режим поиска в направлении вверх"),
                        TEXT("Поиск и замена"), MB_OK | MB_ICONINFORMATION);
}

BOOL CALLBACK CMainDialog::DlgProc(HWND hwnd, UINT message, WPARAM wParam,
                                   LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
      }

      // обработка сообщений, посылаемых из немодального диалогового окна
      if(message == WM_FR)
            ptr->MessageFromFindReplace();
      return FALSE;
}
```

```
                              // Standard Dialog.cpp
#include "MainDialog.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                     int nCmdShow)
{
      CMainDialog dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                          CMainDialog::DlgProc);
}
```

3. Практическая часть

Дополнить приложение «Текстовый редактор» из прошлого занятия следующей функциональностью:

- возможность поиска в тексте определенной последовательности символов, используя стандартный диалог «Найти»;
- возможность замены искомой последовательности символов на другую последовательность символов, используя стандартный диалог «Заменить».

4. Подведение итогов

Подвести общие итоги занятия. Напомнить слушателям, какой отличительной особенностью обладает дополнительное немодальное диалоговое окно. Повторить основные принципы взаимодействия приложения с дополнительными немодальными диалогами. Отметить основные особенности работы со стандартными диалогами «Найти» и «Заменить». Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

5. Домашнее задание

Дополнить приложение «Текстовый редактор» немодальным диалоговым окном, показывающим статистическую информацию о введенном тексте: количество символов, количество строк, количество знаков препинания и т.д. При этом информация в немодальном окне должна меняется динамически по мере внесения изменений в текст.