Модуль №5

Занятие №1

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Дополнительные модальные диалоги.
3. Стандартные диалоги «Открыть» и «Сохранить как».
4. Практическая часть.
5. Подведение итогов.
6. Домашнее задание.

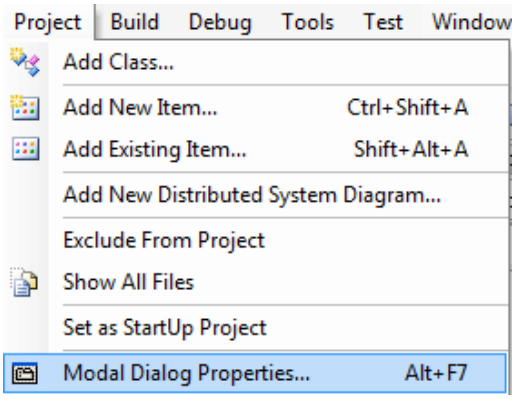
1. Повторение пройденного материала

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

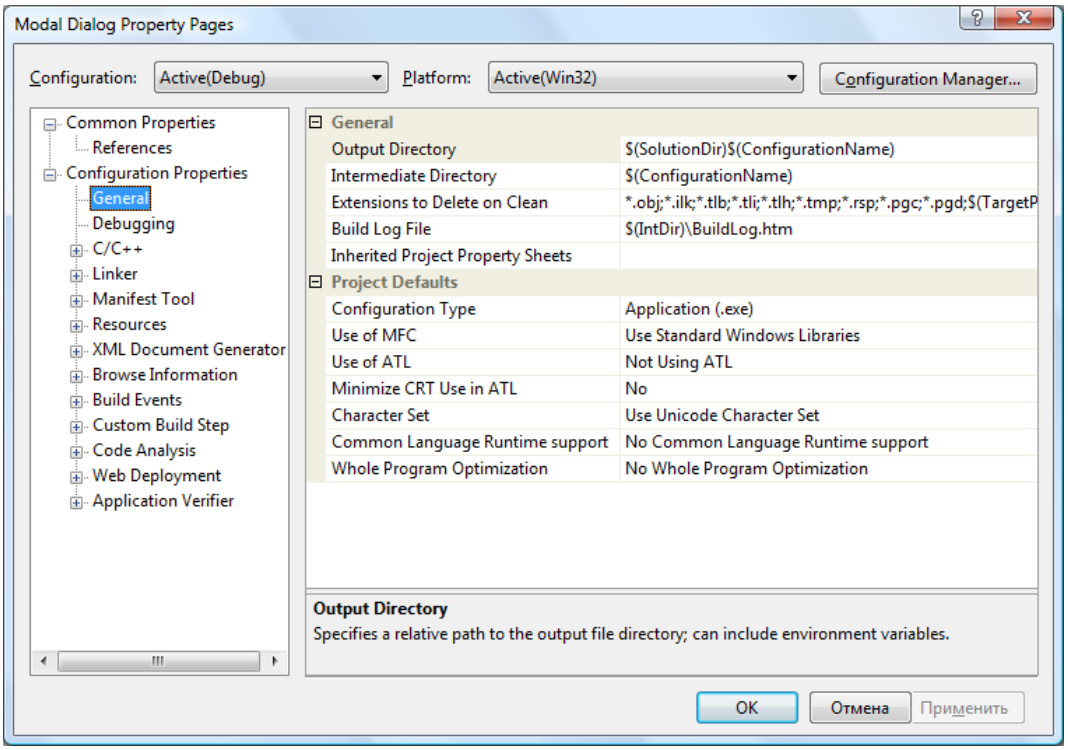
1. Что такое контекстное меню?
2. При каком событии в оконную процедуру приходит сообщение **WM_CONTEXTMENU**?
3. Какая функция API позволяет отобразить контекстное меню?
4. Что такое акселераторы?
5. Как добавлять акселераторы к проекту?
6. Какая функция API позволяет загрузить в память приложения таблицу акселераторов?
7. С какой целью в приложении используется функция API **TranslateAccelerator**?

2. Дополнительные модальные диалоги

Обычно дополнительные диалоговые окна предназначены для задания определённых настроек приложения. Чаще всего дополнительные диалоги вызываются при выборе пунктов меню, у которых в конце названия стоит троеточие.



Ярким примером дополнительного диалогового окна является диалог «**Property Pages**» для указания настроек проекта в среде Microsoft Visual Studio.



Другими примерами дополнительных диалогов могут служить диалоги «Открыть», «Сохранить как», «Шрифт» приложения «Блокнот».

Напомнить слушателям, что диалоговые окна бывают **модальными** и **немодальными**. Наиболее часто используются модальные диалоговые окна. Это означает, что приложение дожидается завершения диалога и только затем его выполнение будет продолжено. При этом модальный диалог до своего завершения не позволяет пользователю работать с другими окнами этого же приложения, однако разрешает переключаться на работу с другими приложениями.

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

Следует отметить, что для создания и отображения дополнительного модального 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 **DialogBox** для отображения модального диалога.

Рассмотреть со слушателями следующее приложение, в котором демонстрируются основные принципы обмена данными между главным окном приложения и дополнительным модальным 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, hEdit2, hStatic2;
};


                               // MainModalDialog.cpp
#include "MainModalDialog.h"
#include "AdditionalModalDialog.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);
	hEdit2 = GetDlgItem(hwnd, IDC_EDIT2);
	hStatic2 = GetDlgItem(hwnd, IDC_STATIC2);
	SetWindowText(hEdit1, TEXT("Передача данных дополнительному диалогу!"));
	SetWindowText(hEdit2, TEXT("Передача данных дополнительному диалогу!"));
	return TRUE;
}

void CMainModalDialog::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                     UINT codeNotify)
{
	if(id == IDC_BUTTON1)
	{
```

```
            CAdditionalModalDialog dlg;
            TCHAR buffer[200];
            GetWindowText(hEdit1, buffer, 200);
            // передача данных через public-поле класса CAdditionalModalDialog
            // дополнительного диалога
            _tcscpy(dlg.text, buffer);
            INT_PTR result = DialogBox(GetModuleHandle(NULL),
                                       MAKEINTRESOURCE(IDD_DIALOG2), hwnd,
                                       CAdditionalModalDialog::DlgProc);
            if(result == IDOK)
            {
      // Переданные от дополнительного диалога данные отображаются на статике
                  SetWindowText(hStatic1, dlg.text);
            }
      }
      else if(id == IDC_BUTTON2)
      {
            TCHAR buffer[200];
            GetWindowText(hEdit2, buffer, 200);
            // передача данных через конструктор класса CAdditionalModalDialog
            // дополнительного диалога
            CAdditionalModalDialog dlg(buffer);
            INT_PTR result = DialogBox(GetModuleHandle(NULL),
                                       MAKEINTRESOURCE(IDD_DIALOG2), hwnd,
                                       CAdditionalModalDialog::DlgProc);
            if(result == IDOK)
            {
      // Переданные от дополнительного диалога данные отображаются на статике
                  SetWindowText(hStatic2, dlg.text);
            }
      }
}

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;
}


                        // AdditionalModalDialog.h

#pragma once
#include "header.h"

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

```
	static CAdditionalModalDialog* 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);
	TCHAR text[200];
	HWND hEdit, hStatic;
};

                        // AdditionalModalDialog.cpp
#include "AdditionalModalDialog.h"

CAdditionalModalDialog* CAdditionalModalDialog::ptr = NULL;

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

CAdditionalModalDialog::CAdditionalModalDialog(LPCTSTR lpStr)
{
	ptr = this;
	_tcscpy(text, lpStr);
}

CAdditionalModalDialog::~CAdditionalModalDialog(void)
{

}

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

BOOL CAdditionalModalDialog::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                              LPARAM lParam)
{
	hEdit = GetDlgItem(hwnd, IDC_EDIT1);
	hStatic = GetDlgItem(hwnd, IDC_STATIC1);
	// Переданные от главного диалога данные отображаются на статике
	SetWindowText(hStatic, text);
	SetWindowText(hwnd, TEXT("Дополнительный модальный диалог"));
	SetWindowText(hEdit, TEXT("Передача данных главному диалогу!"));
	return TRUE;
}

void CAdditionalModalDialog::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                           UINT codeNotify)
{
	if(id == IDOK)
	{
		TCHAR buffer[200];
		GetWindowText(hEdit, buffer, 200);
		_tcscpy(text, buffer);
		HWND hParent = GetParent(hwnd);
		// Передача данных главному диалогу
		SetWindowText(hParent, TEXT("Привет от дочернего окна!"));
```

```
            EndDialog(hwnd, IDOK);
      }
      else if(id == IDCANCEL)
      {
            EndDialog(hwnd, IDCANCEL);
      }
}

BOOL CALLBACK CAdditionalModalDialog::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;
}


                              // ModalDialog.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);
}
```

Анализируя вышеприведенный код, следует выделить два способа передачи данных дополнительному диалоговому окну:

- передача данных через **public**-поле **text** класса **CAdditionalModalDialog** дополнительного диалога;
- передача данных через параметрический конструктор класса **CAdditionalModalDialog** дополнительного диалога.

При любом из вышеперечисленных вариантов после создания объекта класса **CAdditionalModalDialog** дополнительного диалога в **public**-поле **text** будут находится данные, переданные из главного диалога. Впоследствие эти данные будут отображены на статическом элементе управления.

Продолжая анализировать код, следует отметить два способа передачи данных главному диалоговому окну:

- передача данных через **public**-поле **text** класса **CAdditionalModalDialog** дополнительного диалога;
- непосредственное обращение к главному окну приложения.

Второй вариант предполагает получение дескриптора главного окна приложения.

```
INT_PTR result = DialogBox(GetModuleHandle(NULL),
MAKEINTRESOURCE(IDD_DIALOG2), hwnd /* дескриптор главного окна приложения */,
CAdditionalModalDialog::DlgProc);
```

Анализируя третий параметр функции **DialogBox** для отображения дополнительного диалога, очевидно, что главное диалоговое окно является родительским окном для дополнительного диалога. Это позволяет с помощью функции API **GetParent** получить дескриптор родительского (главного) окна.

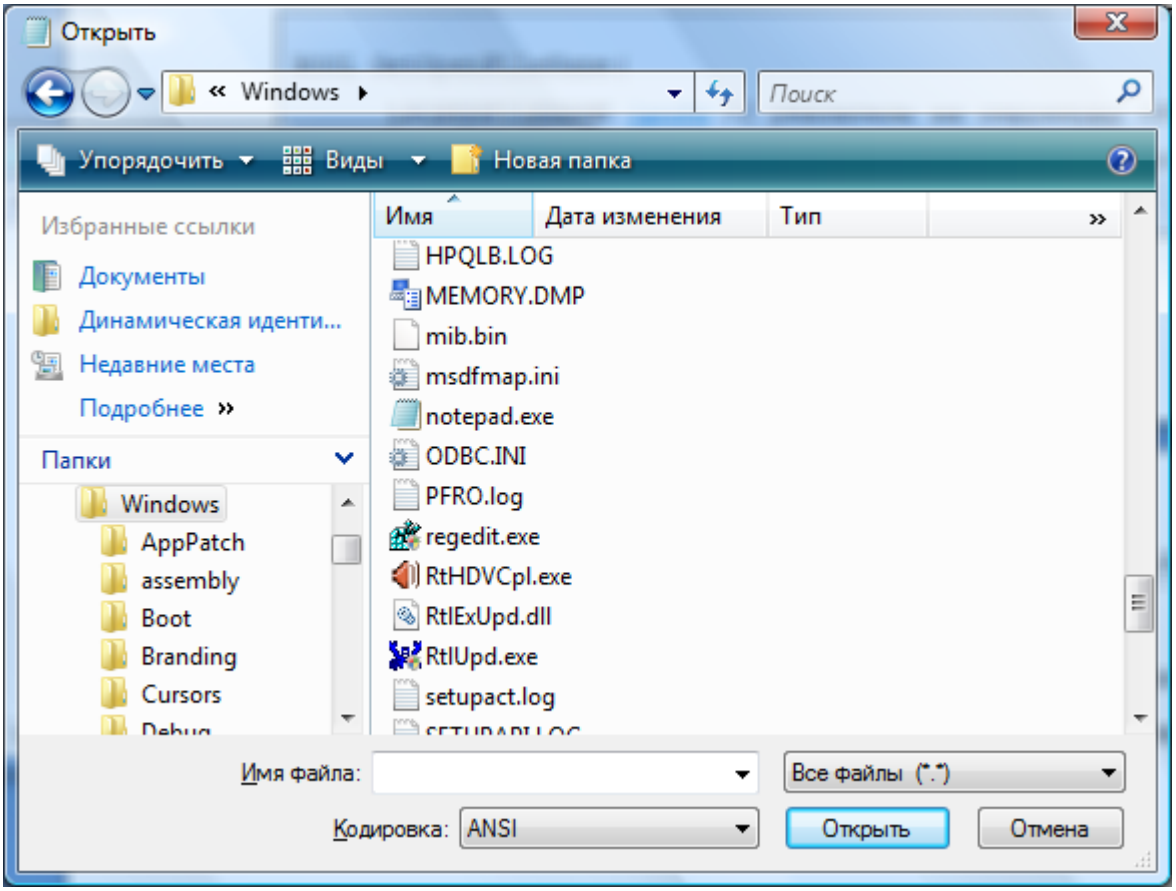
```
HWND GetParent(
     HWND hWnd // дескриптор дочернего окна
);
```

3. Стандартные диалоги «Открыть» и «Сохранить как»

Стандартный диалог «**Открыть**» предоставляет пользователю возможность выбора файла или группы файлов для открытия. Для создания и отображения модального диалога «**Открыть**» предназначена функция API **GetOpenFileName**.

```
BOOL GetOpenFileName(
      LPOPENFILENAME lpofn // указатель на структуру OPENFILENAME, которая
      // содержит информацию, используемую для инициализации диалога
);
```

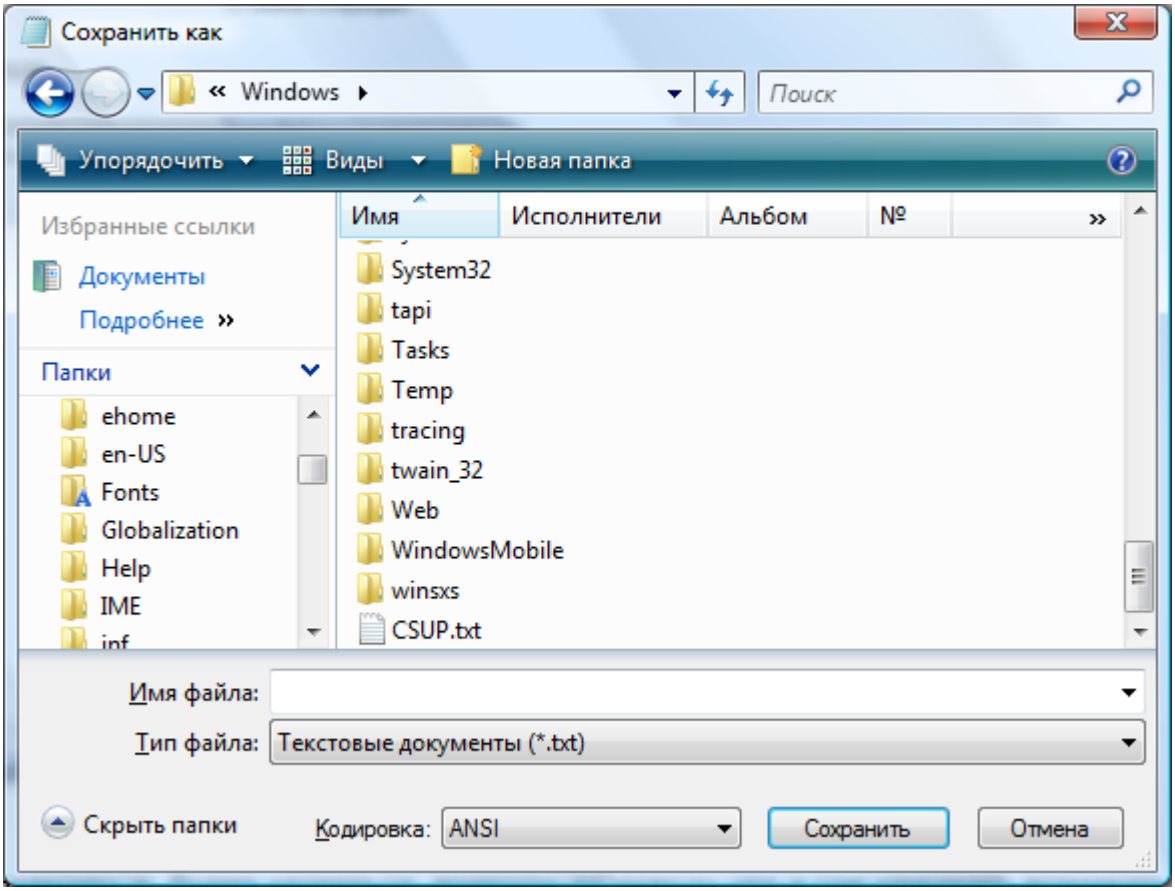
Данная функция вернет ненулевое значение, если при выборе файла будет нажата кнопка **OK** (**Открыть**).



Стандартный диалог «**Сохранить как**» предоставляет пользователю возможность выбора файла для сохранения. Для создания и отображения модального диалога «**Сохранить как**» предназначена функция API **GetSaveFileName**.

```
BOOL GetSaveFileName(
      LPOPENFILENAME lpofn // указатель на структуру OPENFILENAME, которая
      // содержит информацию, используемую для инициализации диалога
);
```

Данная функция вернет ненулевое значение, если при выборе файла будет нажата кнопка **OK** (**Сохранить**).



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

```
typedef struct tagOFN {
  DWORD         lStructSize; // размер структуры в байтах
  HWND          hwndOwner; // дескриптор окна, которое владеет диалогом
  HINSTANCE     hInstance; // дескриптор приложения, если установлен флаг
// OFN_ENABLETEMPLATE, либо дескриптор объекта памяти, содержащего шаблон
// диалогового окна, если установлен флаг OFN_ENABLETEMPLATEHANDLE
  LPCTSTR       lpstrFilter; // указатель на буфер, содержащий одну или
// несколько пар текстовых строк, задающих фильтры для выбора имен файлов
  LPTSTR        lpstrCustomFilter; // указатель на статический буфер, который
// содержит пару текстовых строк фильтра для сохранения модели фильтра,
// выбранного пользователем
  DWORD         nMaxCustFilter; // размер буфера в байтах
  DWORD         nFilterIndex; // индекс текущего выбранного фильтра
```

```
  LPTSTR        lpstrFile; // указатель на буфер, содержащий имя файла для
// инициализации текстового поля File Name диалога. После выбора файла и
// закрытия диалога буфер будет содержать полный путь к выбранному файлу
  DWORD         nMaxFile; // размер буфера в байтах
  LPTSTR        lpstrFileTitle; // указатель на буфер, содержащий имя и
// расширение выбранного файла
  DWORD         nMaxFileTitle; //  размер буфера в байтах
  LPCTSTR       lpstrInitialDir; // указатель на строку, которая определяет
// начальный каталог
  LPCTSTR       lpstrTitle; // указатель на строку, которая определяет
// заголовок диалогового окна
  DWORD         Flags; // комбинация из одного или нескольких флагов,
// объединенных  с помощью операции побитового "или" (|), позволяющих
// настроить внешний вид диалога
  WORD          nFileOffset; // отсчитываемое от нуля смещение от начала пути
// до имени файла в строке lpstrFile
  WORD          nFileExtension; // отсчитываемое от нуля смещение от начала
// пути до расширения имени файла в строке lpstrFile
  LPCTSTR       lpstrDefExt; // строка или указатель на буфер, который
// содержит  расширение имени файла, используемое по умолчанию
  LPARAM        lCustData; // данные, которые передаются в HOOK-процедуру
  LPOFNHOOKPROC lpfnHook; // адрес HOOK-процедуры (должен быть включен флаг
// OFN_ENABLEHOOK)
  LPCTSTR       lpTemplateName; // Указатель на строку, которая именует
// ресурс шаблона диалога в модуле, идентифицированном полем hInstance
#if (_WIN32_WINNT >= 0x0500)
  void *        pvReserved; // зарезервирован – должен быть 0
  DWORD         dwReserved; // зарезервирован – должен быть 0
  DWORD         FlagsEx; // нулевое значение либо флаг OFN_EX_NOPLACESBAR
#endif // (_WIN32_WINNT >= 0x0500)
} OPENFILENAME, *LPOPENFILENAME;
```

Поле **Flags** структуры **OPENFILENAME** представляет собой комбинацию из одного или нескольких флагов, объединенных с помощью поразрядной операции «ИЛИ», и позволяющих настроить внешний вид диалога. Ниже представлены наиболее часто используемые флаги.

- **OFN_ALLOWMULTISELECT** - разрешает выбор нескольких файлов одновременно.
- **OFN_CREATEPROMPT** - если пользователь определяет несуществующий файл, то выводится диалог, в котором предлагается создать файл.
- **OFN_EXTENSIONDIFFERENT** - устанавливается после закрытия диалога и указывает, что расширение выбранного файла отличается от того, которое было ранее определено в поле **lpstrDefExt**. Этот флаг не устанавливается, если в поле **lpstrDefExt** был установлен **NULL** или файл не имеет расширения.
- **OFN_FILEMUSTEXIST** - определяет, что в поле **FileName** пользователь может вводить только имена существующих файлов. В противном случае на экран будет выводиться предупреждающее сообщение. Если этот флаг определен, то автоматически устанавливается флаг **OFN_PATHMUSTEXIST**.
- **OFN_HIDEREADONLY** - предписывает убрать из диалога флажок **Read Only** (только чтение).
- **OFN_NONETWORKBUTTON** - скрывает и блокирует кнопку **Network** (сеть).
- **OFN_NOREADONLYRETURN** - определяет, что выбранный файл не имеет атрибута «только чтение» и не располагается в защищенном от записи каталоге.
- **OFN_NOTESTFILECREATE** - определяет, что файл не создается перед закрытием диалога.
- **OFN_NOVALIDATE** - определяет, что стандартные диалоги допускают наличие неразрешенных символов в именах возвращаемых файлов.

- **OFN_OVERWRITEPROMPT** - предписывает диалогу «**Сохранить как**» выводить окно запроса на перезапись, если выбранный файл уже существует.
- **OFN_PATHMUSTEXIST** - определяет, что пользователь может использовать только имеющиеся пути и имена файлов. В противном случае на экран будет выводиться предупреждающее сообщение.
- **OFN_READONLY** - при отображении диалога флажок **Read Only** (только чтение) будет установлен.

В качестве примера, демонстрирующего использование стандартных диалогов «**Открыть**» и «**Сохранить как**», 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);
};


                                  // MainDialog.cpp
#include "MainDialog.h"

CMainDialog* CMainDialog::ptr = NULL;
```

```
CMainDialog::CMainDialog(void)
{
	ptr = this;
}

CMainDialog::~CMainDialog(void)
{

}

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

void CMainDialog::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                UINT codeNotify)
{

	if(id == IDC_SAVE)
	{
		TCHAR FullPath[MAX_PATH] = {0};
		OPENFILENAME open = {sizeof(OPENFILENAME)};
		open.hwndOwner = hwnd;
		open.lpstrFilter = TEXT("Text Files(*.txt)\0*.txt\0");
		open.lpstrFile = FullPath;
		open.nMaxFile = MAX_PATH;
		open.Flags = OFN_OVERWRITEPROMPT | OFN_PATHMUSTEXIST;
		open.lpstrDefExt = TEXT("txt");
		if(GetSaveFileName(&open))
		{
			TCHAR str[300] = TEXT("Вы выбрали для сохранения файл:\n");
			_tcscat(str, FullPath);
			MessageBox(hwnd, str, TEXT("Диалог \"Сохранить\""), MB_OK |
                                MB_ICONINFORMATION);
		}
	}
	else if(id == IDC_OPEN)
	{
		TCHAR FullPath[MAX_PATH] = {0};
		OPENFILENAME open = {sizeof(OPENFILENAME)};
		open.hwndOwner = hwnd;
		open.lpstrFilter =
			TEXT("Text Files(*.txt)\0*.txt\0All Files(*.*)\0*.*\0");
		open.lpstrFile = FullPath;
		open.nMaxFile = MAX_PATH;
		open.lpstrInitialDir = TEXT("C:\\");
		open.Flags = OFN_CREATEPROMPT | OFN_PATHMUSTEXIST |
                        OFN_HIDEREADONLY;
		if(GetOpenFileName(&open))
		{
			TCHAR str[300] = TEXT("Вы выбрали для открытия файл:\n");
			_tcscat(str, FullPath);
			MessageBox(hwnd, str, TEXT("Диалог \"Открыть\""), 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_COMMAND, ptr->Cls_OnCommand);
      }
      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);
}
```

4. Практическая часть

Разработать приложение «Текстовый редактор», обладающее следующей функциональностью:

- возможность создания нового текстового документа;
- возможность открытия существующего текстового документа;
- возможность сохранения текущего документа;
- возможность сохранения текущего документа под новым именем.

5. Подведение итогов

Подвести общие итоги занятия. Напомнить слушателям, для чего предназначены дополнительные диалоговые окна. Отметить, чем отличается модальное диалоговое окно от немодального окна. Повторить ос-

новные принципы взаимодействия приложения с дополнительными модальными диалогами. Выделить основные особенности работы со стандартными диалогами «Открыть» и «Сохранить как». Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

6. Домашнее задание

Написать приложение «Записная книжка», отвечающее следующим требованиям.

- Записная книжка должна хранить ФИО человека, его адрес и телефон.
- Необходимо обеспечить возможность добавления, удаления и модификации записей.
- Для ввода новых данных и модификации данных необходимо использовать дополнительный диалог, причем для модификации данных следует предусмотреть вызов дополнительного диалога с полями, заполненными текущими значениями (текущие значения берутся из той записи, которая на данный момент выделена).
- Обеспечить возможность сохранения данных в файл и загрузку данных из файла.