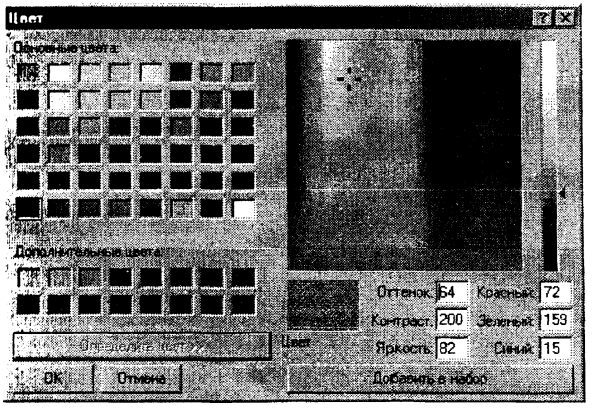
Министерство образования Республики Беларусь

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Интеллектуальные информационные технологии»

ТИПОВЫЕ ДИАЛОГОВЫЕ ОКНА

Методическое пособие для студентов специальности 1 - 40 02 01, 1 - 40 03 01



БРЕСТ 2011

УДК 681.3 (075.8)
ББК с57

Целью пособия является знакомство студентов с базовыми понятиями, организацией и использованием типовых диалоговых окон при программировании интерфейсов оконных приложений в системе Microsoft Visual Studio C++ .

Авторы-составители: Г.Л. Муравьев, доцент, к.т.н.,
С.В. Мухов, доцент, к.т.н.

1. ВВЕДЕНИЕ

1.1. Оконный интерфейс

Интерфейсы, как удобное и строго организованное отражение процесса обработки информации, являются основой взаимодействия современных информационных систем. При этом, если интерфейс какого-либо объекта не меняется (стабилен, стандартизирован), то это даёт возможность модифицировать сам объект, программу, приложение, не перестраивая принципы его взаимодействия с другими объектами.

Оконный, графический интерфейс – способ организации полноэкранного интерфейса программы, при котором каждая составная часть располагается в отдельном окне. Несколько окон, одновременно находящихся на экране, могут перекрываться, располагаясь выше, ниже или в стороне друг относительно друга.

Пользовательский графический интерфейс предназначен обеспечить интерактивный доступ к приложению со стороны пользователей. Для этого функции поддержки оконного интерфейса должны отображать экран с располагающимися на нем окнами и распределять ввод пользователя между ними. При существовании нескольких равноправных окон ввод пользователя осуществляется в том, которое в данный момент является активным.

Программы с полной реализацией оконного интерфейса раздельно работают с отдельными подзадачами в разных окнах. Например, многооконный редактор Word позволяет работать с одним документом в нескольких окнах и с разными документами, одновременно загруженными и отображаемыми каждый в своем окне.

В ОС Windows для этого используются специальные графические объекты. Как правило, это окна различных типов, включая диалоговые окна (типовые и пользовательские), служащие подложкой, где могут располагаться различные элементы управления (ЭУ) и сами элементы управления, представляющие собой специализированные окна.

Основные элементы управления – это: кнопки, переключатели, списки (комбинированные, графические), рамки, линии прокрутки, линейки с ползунками, окна редактирования, индикаторы и т.д.

Окна имеют прямоугольную форму, обрамление (рамку) и цвет фона, отличный от цвета основного экрана. Окна имеют заголовок (с пояснением его назначения) и содержат ЭУ в качестве органов управления.

Применяются различные эффекты для придания ощущения объемности интерфейса, в том числе:

- тени, имитируемые путем затемнения со сдвигом области экрана под самим окном (в графическом режиме тени также могут отбрасывать и другие элементы интерфейса, например, курсор мыши);
- создание иллюзии выпуклых и вдавленных структур для линий, надписей, пониженных или повышенных областей, например, для кнопок, рамок и т.п. путем использования линий повышенной и пониженной яркости и полутоновых переходов для имитации криволинейных поверхностей;
- полная или частичная прозрачность окна в виде просвечивания сквозь "подложку" или другие окна, что применимо только в графическом режиме и др.

Графические объекты, формирующие визуальное представление интерфейса приложения, построены в соответствии со стандартом GDI, называются графическими ресурсами и могут быть описаны в отдельном файле. С информационной точки зрения, ре-

сурсы – это описания графических объектов, достаточные для визуализации соответствующего ресурса. Это текстовые описания (например, описания меню, диалоговых окон), выполненные в терминах специальных декларативных команд. Это описания, представленные в графическом формате (например, пиктограммы, битовые образы). В модульном представлении им соответствуют файлы описаний ресурсов, не разделяемые с другими приложениями.

В Windows-приложениях для хранения описаний ресурсов используется ресурсный файл типа ResourceScript, который должен быть создан и подключен к приложению командой #include. А сами ресурсы, таким образом, могут создаваться "вручную" (например, путем описания меню в текстовом редакторе), либо с помощью редакторов ресурсов.

Также для выполнения часто используемых, стандартных действий применяются стандартные или типовые диалоговые окна Windows (стандартизованный элемент интерфейса). Это является проявлением тенденции стандартизации элементов интерфейса в целом и оконного интерфейса в частности, что позволяет упростить как процессы разработки ПО, так и процессы адаптации пользователя к новому программному продукту.

1.2. Диалоговые окна

Диалоговое окно (англ. dialog box) в графическом пользовательском интерфейсе представляет собой разновидность окон, предназначенных для вывода информации и (или) получения ответа от пользователя.

Диалоговое окно получает сообщения от Windows и в том числе сообщения от элементов управления диалогового окна. Таким образом пользователи осуществляют интерактивное взаимодействие с приложением.

Диалоговые окна подразделяют на модальные и немодальные в зависимости от характера их функционирования, в зависимости от того, блокируют ли они возможность взаимодействия пользователя с приложением или системой в целом до тех пор, пока не получат от него ответ.

Модальные диалоговые окна перехватывают фокус и требуют завершения для дальнейшей работы приложения. Модальные диалоговые окна применяют и в других случаях, когда приложению для продолжения начатой работы требуется дополнительная информация, либо просто подтверждение от пользователя на согласие выполнить запрошенную последовательность действий.

Немодальные окна позволяют приложению работать без закрытия самого окна. Немодальные (англ. modeless) диалоговые окна используются в случаях, когда выводимая в окно информация не является существенной для дальнейшей работы системы. Поэтому окно может оставаться открытым, в то время как работа пользователя с системой продолжается.

Стиль конкретного диалогового окна определяется комбинацией стилей windows-окон и стилей диалоговых окон.

Простейшим типом диалогового окна является окно сообщения (англ. message box), которое выводит сообщение и требует от пользователя подтвердить, что сообщение прочитано. Для этого обычно необходимо нажать кнопку OK. Окно сообщения предназначено для подтверждения системой выполнения команды, вывода сообщения об

ошибке и тому подобных случаев, не требующих от пользователя каких-либо специальных действий.

Диалоговые окна в MFC функционально базируются на классе CDialog, производном от класса CWnd.

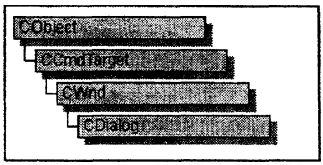


Рисунок 1 – Иерархия классов для работы с диалоговыми окнами

Основные члены класса: конструкторы CDialog, методы инициализации немодального окна Create, CreateIndirect, модального окна InitModalIndirect. Методы DoModal, EndDialog для активизации и закрытия модального окна, void CDialog::GotoDlgCtrl (CWnd* УказательЭУ_ПолучателяФокуса) для передачи фокуса конкретному ЭУ окна и др. Кроме этого, класс включает подменяемые методы: OnOk( ), OnCancel( ), OnInitDialog( ).

Объект CDialog есть комбинация шаблона окна и CDialog-производного класса. Шаблон может быть выполнен: а) в виде ресурса, когда диалоговое окно визуально описывается в редакторе ресурсов; б) в виде ресурса, когда диалоговое окно задано текстовым описанием; в) шаблон может быть создан программно в памяти.

В пользовательском классе окна, как правило, необходимо добавить члены-данные для поддержки ЭУ (ввода-вывода данных), обработчики сообщений ЭУ, предусмотреть инициализацию (обработка сообщения WM_INITDIALOG), описать конструкторы.

1.3. Типовые диалоговые окна

Стандартные диалоговые окна можно видеть в интерфейсах широко используемых приложений общего назначения типа Word, Excel и других. Это следующие окна:

- открытие-сохранение файла (Open-Save File Dialog);
- выбор цвета (Color Dialog);
- настройка параметров страницы (PageSetup Dialog);
- настройка печати (Print Dialog);
- настройка шрифта (Font Dialog);
- поиск-замена (FindReplace Dialog) и другие.

При работе в системе Microsoft Visual Studio C++ разработчик может использовать их при разработке собственного приложения через два средства встраивания:

- средства Win API;
- используя соответствующие классы библиотеки MFC (их описания, например, содержатся в заголовочном файле commdlg.h).

Библиотека MFC определяет понятие каркаса приложения как "скелетную" программу, автоматически создаваемую по заданному макету интерфейса и полностью берущую на себя рутинную работу по его обслуживанию. При этом каркас должен иметь определенную типовую структуру, поэтому для его генерации и изменения в Visual C++ предусмотрены специализированные средства – мастера (Application Wizards).

Использование классов библиотеки MFC ускоряет и упрощает разработку приложений. MFC является надстройкой над Win API, то есть классы типовых диалоговых окон (представленные в таблице 1) инкапсулируют соответствующие структуры данных Win API для информационной поддержки окон (например, структуру типа OPENFILENAME для типого окна – открытие файла) и методы работы с ними. Причем непосредственная рутинная работа по вызову функций типа GetOpenFileName и других может быть сокрыта от разработчика и по смыслу идентична методу DoModal.

Таблица 1 – Стандартные диалоги

Класс стандартного окна	Назначение
CColorDialog	выбор цветов
CFileDialog	выбор имени файла для открытия или сохранения
CFindReplaceDialog	управление поиском и заменой фрагментов в текстовом файле
CFontDialog	выбор фонта
CPrintDialog	настройка печати
CPageSetupDialog	настройка параметров страницы

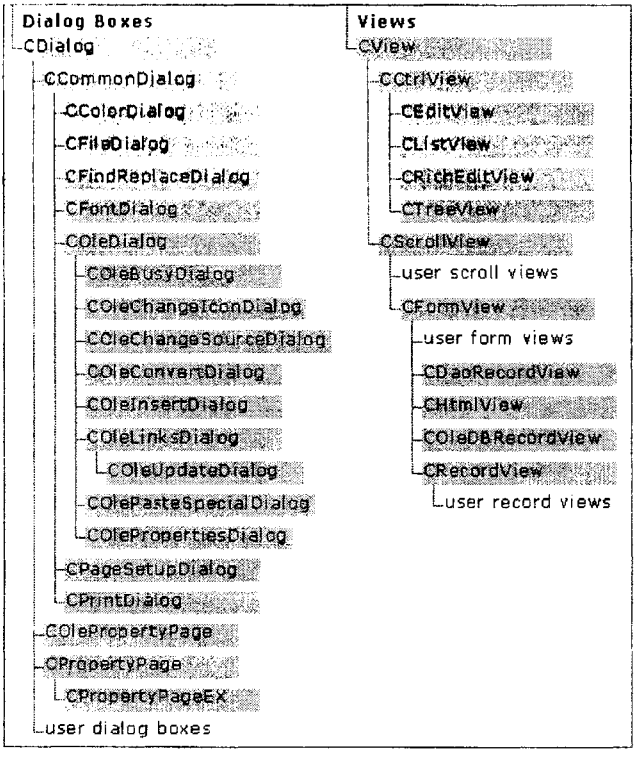


Рисунок 2 – Фрагмент иерархии классов, наследуемой от CDialog

С типовым MFC-приложением связывается определяющий его на верхнем уровне объект, принадлежащий классу, производному от класса CWinApp. При этом оконный

интерфейс приложения строится как система взаимодействующих окон различных стилей и типов, производных от класса CWnd. Так, для приложений с однодокументной архитектурой интерфейс строится на базе классов CFrameWnd, CDialog, семейства классов элементов управления (ЭУ) и др. Диалоговые окна интерфейса, являющиеся подложкой для размещения и компоновки различных ЭУ, производятся от класса CDialog.

Соответственно для обеспечения функциональности типовых диалоговых окон применяются члены и методы таких приведенных в таблице 1 классов, как CFileDialog, CPrintDialog, CColorDialog, CFindReplaceDialog, CFontDialog, COleDialog, CPageSetupDialog и других, производных от класса CDialog.

2. ИСПОЛЬЗОВАНИЕ ТИПОВЫХ ДИАЛОГОВЫХ ОКОН

При создании типового диалогового окна средствами MFC используется тот факт, что в MFC для каждого типа стандартных диалогов реализован соответствующий класс, обеспечивающий функционирование конкретного типового диалога. Поэтому для работы с диалогом необходимо создать объект – экземпляр соответствующего класса и, после необязательной предварительной настройки, активизировать диалоговое окно, методом DoModal.

Обработка результатов выбора-настройки пользователя, произведенных во время работы с типовым диалогом, может осуществляться как через чтение данных из самой типовой структуры (метод, присущий программированию в Win API), так и с помощью вызова методов класса. Все описанные ниже классы представлены в файле <afxdlgs.h>.

2.1. Окно выбора цвета

При работе с окном средствами MFC используется класс CColorDialog. Он отвечает за создание модального диалогового окна выбора и (или) создания цветов, в котором пользователь может выбрать цвет из готовой палитры или воспользоваться предоставленным спектром.

Внешний вид окна представлен на рисунке ниже.

Ниже представлено описание класса CColorDialog, поддерживающего функциональность рассматриваемого окна

```
class CColorDialog : public CCommonDialog
{
    DECLARE_DYNAMIC (CColorDialog)
public:
// Attributes
    // color chooser parameter block
    CHOOSECOLOR m_cc;
// Constructors
    CColorDialog(COLORREF clrInit = 0, DWORD dwFlags = 0,
                 CWnd* pParentWnd = NULL);
// Operations
    virtual int DoModal( );
    // Set the current color while dialog is displayed
    void SetCurrentColor(COLORREF clr);
```

```
    // Helpers for parsing information after successful return
    COLORREF GetColor() const;
    static COLORREF* PASCAL GetSavedCustomColors();
// Overridable callbacks
protected:
    friend UINT CALLBACK _AfxCommDlgProc(HWND, UINT, WPARAM, LPARAM);
    virtual BOOL OnColorOK();
};
```

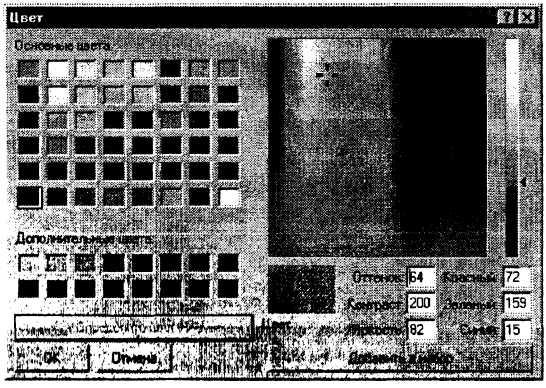


Рисунок 3 – Вид типового диалового окна выбора и (или) создания цветов

Член класса CHOOSECOLOR **m_cc** представляет собой структуру, в которой хранится описание текущего состояния диалогового окна.

Описание структуры CHOOSECOLOR

```
typedef struct tagCHOOSECOLORA
{
    DWORD          lStructSize;
    HWND           hwndOwner;
    HWND           hInstance;
    COLORREF       rgbResult;
    COLORREF*      lpCustColors;
    DWORD          Flags;
    LPARAM         lCustData;
    LPCCHOOKPROC   lpfnHook;
    LPCSTR         lpTemplateName;
}   CHOOSECOLOR, *LPCHOOSECOLOR .
```

Здесь:

lStructSize – определяет размер структуры в байтах и обычно инициализируется значением sizeof (CHOOSECOLOR);

hwndOwner – указатель на окно-владелец диалогового (может быть NULL при его отсутствии);

hInstance – определяет идентификатор блока памяти (если установлен флаг CC_ENABLETEMPLATEHANDLE) или дескриптор модуля (если установлен флаг

CC_TEMPLATE), содержащего шаблон диалогового окна. Если не установлен ни один из флагов, то содержимое поля игнорируется;

rgbResult – поле (значение красного, зеленого и синего – RGB), содержащее цвет, выбранный пользователем. Если указанный пользователем цвет не входит в список доступных, то система подберет ближайший подходящий цвет. Если установлен флаг *CC_RGBINIT*, то значение *rgbResult* определяет выбор по умолчанию. Если флаг *CC_RGBINIT* не установлен или rgbResult имеет нулевое значение, то цветом по умолчанию выбирается черный цвет. При завершении работы диалогового окна в это поле записывается код последнеего цвета, выбранного пользователем;

lpCustColors – указатель на массив из 16 параметров, содержащих значения RGB, для заполнения пользовательской палитры диалогового окна; если пользователь изменит эту палитру, то изменения сохранятся в указанном массиве;

Flags – набор битовых флагов, с помощью которого можно настроить диалоговое окно. Значения флагов можно комбинировать по "Или", соединяя соответствующим символом "|". Некоторые из ранее не упомянутых значений флагов приведены ниже:

а) CC_ANYCOLOR – определяет, что в диалоговом окне будут отображены все доступные цвета в палитре базовых цветов;

б) CC_FULLOPEN – определяет, что в диалоговом окне будут отображаться дополнительные средства управления, позволяющие пользователю создавать свои собственные цвета, не входящие в базовую палитру; если флаг не установлен, то для получения дополнительных компонентов управления пользователю придется нажимать кнопку «Определить цвет»;

в) CC_PREVENTFULLOPEN – отключает кнопку «Определить цвет»;

д) CC_SOLIDCOLOR – определяет, что в палитре базовых цветов будут содержаться только «базовые» цвета.

Базовые методы класса:

- конструктор класса (параметр **clrinit** определяет первоначально выбранный цвет в формате RGB)

CColorDialog::**CColorDialog** (*COLORREF* clrinit = 0, DWORD dwFlags = 0, CWnd *pParentWnd = NULL);

- метод, отвечающий за вывод на экран диалогового окна выбора цвета

virtual int CColorDialog::**DoModal**( ) .

Внешний вид и некоторые особенности работы диалогового окна можно настроить вручную заранее. Метод возвращает значение IDOK при успешном завершении (выборе цвета) и IDCANCEL при закрытии окна кнопкой «Отмена». В последнем случае можно вызвать функцию Windows CommDlgExtendedError для проверки наличия ошибки;

- метод, позволяющий получить информацию о цвете, выбранном пользователем

COLORREF CColorDialog::**GetColor**( ) ,

вызывается после завершения метода CColorDialog::DoModal;

- статический метод, возвращающий указатель на массив, содержащий 16 дополнительных цветов, созданных пользователем

static *COLORREF** CColorDialog::**GetSavedCustomColors**( ) ,

вызывается после завершения метода CColorDialog::DoModal;

- метод, который вызывается из обработчика CColorDialog::OnColorOK для принудительного выбора цвета

clr.void CColorDialog::**SetCurrentColor**(COLORREF clr) ;

- обработчик нажатия кнопки OK в диалоговом окне

virtual BOOL CColorDialog::**OnColorOK**( ) .

Переопределение этой функции практически не требуется, т.к. реализация по умолчанию обеспечивает полную проверку правильности введенного цвета. При переопределении функции для нормального завершения работы диалогового окна она должна возвращать значение FALSE.

Настройка диалогового окна. Работать с диалогом можно, в простейшем случае, просто создав объект класса CColorDialog. Однако воспользоваться всеми предоставляемыми возможностями можно, только заполняя поля структуры m_cc типа CHOOSECOLOR.

Ниже приведен фрагмент настройки диалогового окна выбора цвета и заполнения полей структуры m_cc

```
#include <afxdlgs.h>
// создаем объект
CColorDialog colorDlg;
// устанавливаем инициализирующее значение цвета
// здесь R,G,B – составляющие инициализирующего цвета
colorDlg.m_cc.rgbResult = RGB(R, G, B);
// устанавливаем цвета дополнительной палитры
// устанавливаем флаг CC_RGBINIT
COLORREF init [16];
Init [0] = 124004;
Init [1] = 54423;
Init [2] = 51244;
Init [3] = 82446;
ini t [4] = 0;
colorDlg.m_cc.lpCustColors = init;
// добавляем установки указанных флагов
colorDlg.m_cc.Flags |= CC_PREVENTFULLOPEN;
colorDlg.m_cc.Flags |= CC_RGBINIT;
colorDlg.m_cc.Flags |= FR_RGBINIT; .
```

Управление диалоговым окном выбора цвета. После создания объекта класса CColorDialog и необязательной предварительной настройки окно запускается методом DoModal. В качестве результата DoModal возвращает целое число, которое принимает одно из двух значений IDOK или IDCANCEL. Значение IDOK указывает на тот факт, что пользователь осуществил выбор цвета. Таким образом, работу с окном можно схематично изобразить следующим образом:

```
#include <afxdlgs.h>
. . .
CColorDialog ColorDlg;
```

```
if (ColorDlg.DoModal() == IDOK)
{
    // обработка результата выбора
}
else
    // альтернативная обработка .
```

Обработка результата осуществляется считыванием полей структуры m_cc типа CHOOSECOLOR, где после выбора пользователя будет сохранена информация о выбранном цвете, а также при помощи вызова методов самого класса CColorDialog.

При работе с окном в процедурном стиле используется функция ВыбратьЦвет (< УказательНаСтруктуруТипаCHOOSECOLOR >)

BOOL **ChooseColor**(LPCHOOSECOLOR lpcc) ,

подключаемая в <commdlg.h>.

Для вызова функции ChooseColor необходимо в качестве параметра передать указатель на структуру типа CHOOSECOLOR.

Для корректной работы функции CHOOSECOLOR указанная структура должна быть проинициализирована. Пример инициализации представлен ниже:

```
CHOOSECOLOR cc;
COLORREF crCustColors[16]; // массив цветов пользовательской палитры
// очистка некорректных значений полей
memset (&cc,0,sizeof(CHOOSECOLOR));
// задание размера структуры
cc.lStructSize    = sizeof (CHOOSECOLOR);
// указание окна-владельца
cc.hwndOwner  = hWnd;
// необязательная настройка (значение по умолчанию)
cc.rgbResult      = RGB (0x80, 0x80, 0x80);
// необязательная настройка для сохранения пользовательской палитры цветов
for (i = 0; i < 16; i++)
{
        crCustColors[i] = RGB((i * 30), (i * 50), (i * 40));
}
cc.lpCustColors   = crCustColors;
// выбор по умолчанию цвета, указанного в rgbResult Flags
cc.Flags = CC_RGBINIT;
cc.Flags = CC_PREVENTFULLOPEN;
cc.Flags = CC_FULLOPEN;    .
```

Типовые действия. При работе пользователя с диалоговым окном выбора цвета можно выделить несколько типовых действий, совершаемых пользователем. Ниже описаны изменения в структуре данных, которые соответствуют действиям пользователя:

1. Пользователь выбрал необходимый ему цвет из палитры базовых цветов или воспользовался расширенными возможностями. При этом он не сохранял новый цвет в палитру дополнительных цветов. В результате обработки события функция *ChooseColor*

возвращает истинное значение, а поле *rgbResult* содержит RGB-представление выбранного цвета.

2. Пользователь выбрал необходимый ему цвет из палитры базовых цветов или воспользовался расширенными возможностями и при этом сохранил новый цвет (новые цвета) в палитру дополнительных цветов. В результате обработки события функция *ChooseColor* возвращает истинное значение, а поле *rgbResult* содержит RGB-представление выбранного цвета. Кроме этого, в массиве, на который указывает *lpCustColor*, хранится модифицированная дополнительная палитра.

2.2. Окно открытия и сохранения файлов

При работе с окном средствами MFC используется класс CFileDialog. Он отвечает за создание и обеспечение функционирования двух стандартных диалоговых окон: открытия и сохранения файлов. Создает диалоговое окно, в котором пользователь может выбрать диск, каталог и имя файла, который он хочет открыть или в котором он хочет сохранить информацию.

Внешний вид окна представлен на рисунке ниже.

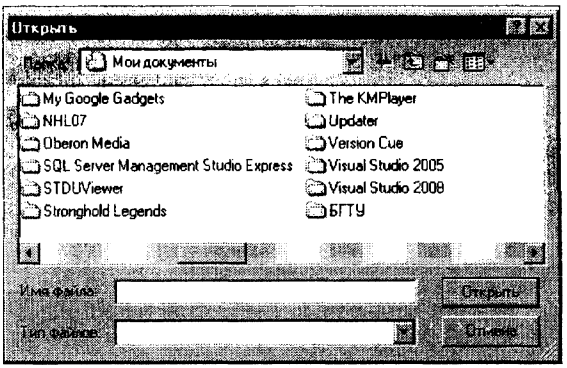


Рисунок 4 – Вид типового диалогового окна открытия и сохранения файлов

Ниже представлено описание класса CFileDialog, поддерживающего функциональность рассматриваемого окна

```
class CFileDialog : public CCommonDialog
{
    DECLARE_DYNAMIC(CFileDialog)
public:
// Attributes
    OPENFILENAME m_ofn;
// Constructors
    CFileDialog(BOOL bOpenFileDialog,
            LPCTSTR lpszDefExt = NULL,
            LPCTSTR lpszFileName = NULL,
            DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
            LPCTSTR lpszFilter = NULL,
            CWnd* pParentWnd = NULL);
```

```
// Methods
        virtual int DoModal();
        CString GetPathName() const;
        CString GetFileName() const;
        CString GetFileExt() const;
        CString GetFileTitle() const;
        BOOL GetReadOnlyPref() const;
        POSITION GetStartPosition() const;
        CString GetNextPathName(POSITION& pos) const;
        void SetTemplate(UINT nWin3ID, UINT nWin4ID);
        void SetTemplate(LPCTSTR lpWin3ID, LPCTSTR lpWin4ID);
        CString GetFolderPath() const;
        void SetControlText(int nID, LPCSTR lpsz);
        void HideControl(int nID);
        void SetDefExt(LPCSTR lpsz);
// Overridable callbacks
protected:
        friend UINT CALLBACK _AfxCommDlgProc(HWND, UINT, WPARAM, LPARAM);
        virtual UINT OnShareViolation(LPCTSTR lpszPathName);
        virtual BOOL OnFileNameOK();
        virtual void OnLBSelChangedNotify(UINT nIDBox, UINT iCurSel, UINT nCode);
        virtual void OnInitDone();
        virtual void OnFileNameChange();
        virtual void OnFolderChange();
        virtual void OnTypeChange();
protected:
        BOOL m_bOpenFileDialog;
        CString m_strFilter;
        TCHAR m_szFileName[_MAX_PATH];
        OPENFILENAME* m_pofnTemp;
        virtual BOOL OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult);
};
```

Член-данное класса *OPENFILENAME* **m_ofn** – структура, в которой хранится описание текущего состояния диалогового окна.

Структура OpenFileName имеет следующий вид:

```
typedef struct tagOFN
{
        DWORD        lStructSize;
        HWND         hwndOwner;
        HINSTANCE    hInstance;
        LPCTSTR      lpstrFilter;
        LPTSTR       lpstrCustomFilter;
        DWORD        nMaxCustFilter;
        DWORD        nFilterIndex;
        LPTSTR       lpstrFile;
```

```
    DWORD       nMaxFile;
    LPTSTR      lpstrFileTitle;
    DWORD       nMaxFileTitle;
    LPCTSTR     lpstrInitialDir;
    LPCTSTR     lpstrTitle;
    DWORD       Flags;
    WORD        nFileOffset;
    WORD        nFileExtension;
    LPCTSTR     lpstrDefExt;
    LPARAM      lCustData;
    LPOFNHOOKPROC lpfnHook;
    LPCTSTR     lpTemplateName;
    #if (_WIN32_WINNT >= 0x0500)
    void *      pvReserved;
    DWORD       dwReserved;
    DWORD       FlagsEx;
    #endif // (_WIN32_WINNT >= 0x0500)
} OPENFILENAME, *LPOPENFILENAME; .
```

Описание функционального назначения некоторых полей структуры OpenFileName и их возможных значений приведено ниже:

- DWORD lStructSize – определяет размер структуры в байтах и обычно инициализируется значением sizeof (OPENFILENAME);
- HWND hWndOwner – дескриптор родительского окна (может быть NULL);
- LPCTSTR lpstrFilter – указатель строки, буфера, содержащего определенным образом форматированные пары строк, задающие фильтры окна (строки-фильтры разбиты на пары символом '\0', где первая строка выводится в качестве подсказки, вторая – определяет характер отбора файлов);
- LPTSTR lpstrCustomFilter – указатель на строку-фильтр, используемую по умолчанию;
- DWORD nMaxCustFilter – размерность буфера lptstrCustomFilter в байтах-символах;
- DWORD nFilterIndex – номер текущей выбранной строки-фильтра (нумерация для строк lpstrFilter начинается с единицы);
- LPTSTR lpstrFile – указатель на строку, содержащую имя файла, которое зафиксировано в окошке редактирования диалогового окна (или NULL). После успешного завершения функций GetOpenFileName или GetSaveFileName это указатель буфера, содержащего полное имя выбора пользователя;
- DWORD nMaxFile – размер буфера, строки lpstrFile в байтах-символах;
- LPTSTR lpstrFileTitle – указатель на буфер, в котором содержится имя выбранного пользователем файла (без указания пути);
- DWORD nMaxFileTitle – размерность буфера lpstrFileTitle в байтах-символах;
- LPCTSTR lpstrInitialDir – указатель на строку, в которой хранится имя начального каталога поиска (алгоритм выбора начального каталога поиска зависит от используемой платформы, например, в Windows XP начальный каталог поиска определяется параметром lpstrFile, а только потом, при неудаче, параметром lpstrInitialDir);
- LPCTSTR lpstrTitle – указатель на строку, которая будет помещена в заголовок диалогового окна, при ее отсутствии используется стандартное название Open;

- LPCSTR lpstrDefExt – указатель на буфер, в котором хранится расширение "по умолчанию" (если пользователь не укажет расширение файла, то оно будет дополнено; строка должна начинаться с символа '.' и может иметь неограниченную длину, но лишь первые три символа будут дописаны);

- DWORD Flags – набор битовых флагов, с помощью которого можно настроить диалоговое окно (обычно флаги комбинируются в список, разделяясь знаком "|", а по завершении работы идентифицируют пользовательские настройки).

Список некоторых часто используемых флагов представлен ниже:

- OFN_ALLOWMULTISELECT разрешает множественный выбор, т.е. выбор сразу набора файлов из одного каталога (буфер lpstrFile будет содержать путь к каталогу и набор имен файлов);

- OFN_FILEMUSTEXIST указывает на то, что пользователь может ввести имя только существующего файла, иначе будет выведено предупреждающее сообщение (данный флаг используется совместно с флагом OFN_PATHMUSTEXIST);

- OFN_FORCESHOWHIDDEN определяет, что вне зависимости от установок системы отображаются скрытые и системные файлы, но файлы имеющие одновременно атрибут скрытый и системный, не отображаются;

- OFN_NONETWORKBUTTON скрывает кнопку Сеть;

- OFN_NOREADONLYRETURN указывает на то, что возвращаемый файл должен не иметь атрибут "только чтение" и он не находится в каталоге, защищенном от записи;

- OFN_PATHMUSTEXIST указывает, что выбранный пользователем путь должен существовать, иначе будет выведено предупреждающее сообщение.

Базовые методы класса:

- конструктор класса

CFileDialog:: **CFileDialog** (BOOL *bOpenFileDialog*, LPCSTR *lpszDefExt*=NULL,
LPCSTR *lpszFileName*=NULL, DWORD *dwFlags*= OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT, LPCTSTR *lpszFilter*=NULL,
CWnd **pParentWnd*=NULL) ,

где, если параметр *bOpenFileDialog* равен TRUE, то создается диалогое окно открытия файла, если FALSE – создается окно сохранения файла (значения других параметров определяются их функцией в структуре *OPENFILENAME*);

- метод вывода на экран модального диалогового окна (возвращает код завершения операции открытия-сохранения файла – IDOK или IDCANCEL)

virtual int CFileDialog::**DoModal** ( ) ;

- метод обновления текущего состояния диалогового окна до описанного в структуре m_ofn

void CFileDialog:: **ApplyOFNToShellDialog** ( ) ;

- метод, позволяющий получить структуру типа *OPENFILENAME*, связанную с диалоговым окном

OPENFILENAME& CFileDialog:: **GetOFN** ( ) ;

- метод, устанавливающий значение фильтра расширений видимых по умолчанию файлов (аналогичен по действию заданию соответсвующего параметра при заполнении структуры *OPENFILENAME*)

void CFileDialog:: **SetDefExt** (LPCSTR *lpsz*) ;

- метод, обновляющий значение структуры *m_ofn* в соответствии с текущим состоянием диалогового окна

void CFileDialog:: **UpdateOFNFromShellDialog** ( ) ;

- метод-обработчик извещения *CDN_SELCHANGE* о том, что пользователь выбрал другой файл

virtual void CFileDialog:: **OnFileNameChange** ( ) ;

- метод-обработчик извещения *CDN_FOLDERCHANGE* о том, что пользователь выбрал другой каталог

virtual void CFileDialog:: **OnFolderChange** ( ) ;

- метод, позволяющий получить полное имя файла, указанного пользователем

virtual CString CFileDialog:: **GetPathName** ( ) ;

- метод, позволяющий получить имя выбранного файла с расширением

virtual CString CFileDialog:: **GetFileName** ( ) ;

- метод, позволяющий получить расширение указанного файла

virtual CString CFileDialog:: **GetFileExt** ( ) ;

- метод, позволяющий получить имя указанного файла без расширения

virtual CString CFileDialog:: **GetFileTitle** ( ) ,

при этом, если при использовании этих функций был установлен флаг OFN_ALLOWMULTISELECT, то возвращаемая строка содержит последовательность строк, первая из которых является полным путем к этой группе файлов, а последующие определяют имена выбранных файлов. В этом случае для извлечения имен файлов используют нижеследующие функции;

- метод, возвращающий позицию первого имени файла в списке, если был разрешен множественный выбор, или NULL, если список пуст

POSITION CFileDialog:: **GetStartPosition** ( ) ;

- метод, возвращающий следующее полное имя файла из группы файлов, выбранных в диалоговом окне (значение NULL возвращается, если достигнут конец списка)

CString CFileDialog:: **GetNextPathName** (POSITION *&pos*) ;

- метод, позволяющий определить, был ли установлен флажок Read Only (Только чтение)

BOOL CFileDialog:: **GetReadOnlyPref** ( ) .

Настройка диалогового окна. Работать с окном можно, в простейшем случае, просто создав объект описанного класса CFileDialog и указав значение флага bOpenFileDialog, определяющее тип окна – окно открытия или сохранения файла. Однако в общем случае для использования всех возможностей необходимо заполнить соответствующие поля структуры m_ofn типа OPENFILENAME.

Ниже приведен фрагмент настройки диалогового окна открытия файлов и заполнения полей структуры m_ofn

```
#include <afxdlgs.h>
...
CFileDialog fileDlg ( TRUE );
TCHAR strName[ MAX_PATH ];
strName[0] = (TCHAR) NULL;
fileDlg.m_ofn.lpstrFile = strName;
CString str;
str += "All files \0  *.*";
str += "C++ code files \0  *.h;*.cpp";
str += "C# code files \0  *.cs";
str += "C code files \0  *.h;*.c";
str += (TCHAR)NULL;
fileDlg.m_ofn.lpstrFilter = str;
fileDlg.m_ofn.nFilterIndex = 0;
fileDlg.m_ofn.lpstrFile = "C:\boot.ini"
fileDlg.m_ofn.lpstrInitialDir = "C:\";
fileDlg.m_ofn.lpstrTitle = "Example 1 (Open)";
fileDlg.m_ofn.Flags |= OFN_ALLOWMULTISELECT | OFN_FILEMUSTEXIST |
                       OFN_PATHMUSTEXIST | OFN_NOREADONLYRETURN;
```

Результаты настройки показаны на рисунке 5.

Ниже приведен фрагмент настройки диалогового окна сохранения файлов и заполнения полей структуры m_ofn

```
#include <afxdlgs.h>
...
CFileDialog fileDlg(FALSE);
TCHAR strName[MAX_PATH];
strName[0] = (TCHAR)NULL;
fileDlg.m_ofn.lpstrFile = strName;
CString str;
str += "All files \0  *.*";
str += "Text files \0  *.txt;*.doc;*.rtf";
str += (TCHAR)NULL;
fileDlg.m_ofn.lpstrFilter = str;
fileDlg.m_ofn.nFilterIndex = 0;
fileDlg.m_ofn.lpstrFile = "C:\boot.ini"
fileDlg.m_ofn.lpstrInitialDir = "C:\";
fileDlg.m_ofn.lpstrTitle = "Example 2 (Save)";
fileDlg.m_ofn.Flags |= OFN_PATHMUSTEXIST | OFN_OVERWRITEPROMPT;
```

Результаты настройки показаны на рисунке 6.

Управление диалоговым окном. После создания объекта класса CFileDialog и необязательной предварительной настройки диалог запускается путем вызова метода DoModal. Возвращаемое методом значение IDOK указывает на тот факт, что пользователь осуществил выбор файла (-ов). Обработка результата осуществляется считыванием полей структуры m_ofn типа OPENFILENAME. Так после выбора пользователя будет

сохранена информация о выбранном файле, а также при помощи вызова методов самого класса CFileDialog. Кроме того, при множественном выборе (т.е. когда пользователь выбрал более одного файла), для доступа к результатам выбора следует использовать методы GetStartPosition и GetNextPathName.

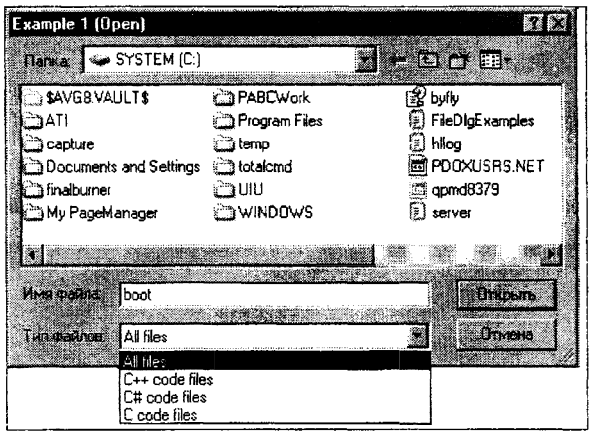


Рисунок 5 – Настроенное окно открытия файлов

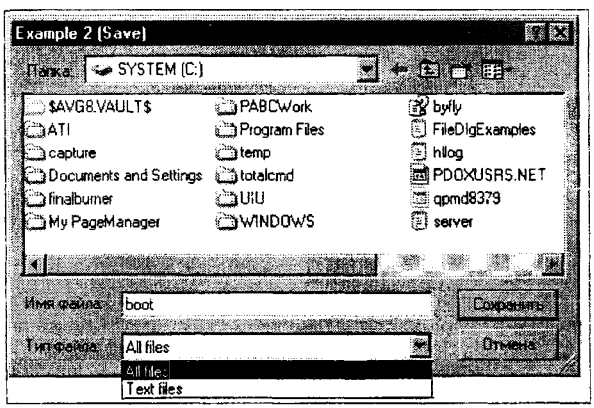


Рисунок 6 – Настроенное окно сохранения файлов

При работе с окном в процедурном стиле используются две разные функции: для сохранения GetSaveFileName и для открытия файла GetOpenFileName.

При работе с окном открытия используется функция ПолучитьИмяОткрытогоФайла (< УказательНаСтруктуруТипаOPENFILENAME >)

BOOL GetOpenFileName(LPOPENFILENAME *СсылкаНаСтруктуру* **)** ,

подключаемая в <commdlg.h>.

Здесь *СсылкаНаСтруктуру* – указатель типа LPOPENFILENAME на структуру типа OPENFILENAME.

При запуске функции в соответствии с установками структуры типа OPENFILE-NAME инициализируется, визуализируется и активизируется диалоговое окно Открытия, которое ждет выбора пользователя. После сделанного выбора функция автоматически завершает свою работу, закрывает окно и заполняет вышеуказанную структуру, которая и содержит информацию о пользовательском выборе файла.

Соответственно необходимо:

1. До начала работы подготовить структуру типа OPENFILENAME. Например,

```
char ИмяФайла [128];
char Фильтр [ ] = "Файлы данных (*.doc) \0 *.dat\0 Все файлы (*.*)\0*.*\0";
OPENFILENAME СтруктураВыбранногоФайла;
memset(&СтруктураВыбранногоФайла, 0, sizeof(OPENFILENAME));
СтруктураВыбранногоФайла.lStructSize = sizeof(OPENFILENAME);
СтруктураВыбранногоФайла.hwndOwner = ДескрипторРодительскогоОкна;
СтруктураВыбранногоФайла.lpstrFilter = Фильтр;
СтруктураВыбранногоФайла.lpstrFile = ИмяФайла;
СтруктураВыбранногоФайла.nMaxFile = sizeof(ИмяФайла);
СтруктураВыбранногоФайла.Flags= OFN_PATHMUSTEXIST | OFN_FILE MUSTEXIST; .
```

2. Выполнить функцию для запуска окна и получения выбора пользователя. Например,

```
GetOpenFileName(&СтруктураВыбранногоФайла) .
```

3. Если выбор состоялся, то обработать его, использовав полученное имя файла. Например,

```
if ( GetOpenFileName (&СтруктураВыбранногоФайла ) )
{
      ОБРАБОТАТЬ_ФАЙЛ  ИмяФайла
} .
```

Ниже представлен пример инициализации структуры OPENFILENAME:

```
OPENFILENAME Name;
// обнуляется содержимое
Memset ( &Name,0,sizeof ( OPENFILENAME ) );
// инициализируются строки фильтров. Строки, подсказка отделяются  символом \0
// расширения файлов разделяются символом ;
char szFilter[ ] = "Файлы-тексты программ (*.CPP, *.C, *.H)\0*.cpp; *.h;*.c\Все файлы
(*.*)\0*.*\0";
// место хранения имени выбранного файла
char szFile[150];
// задается размер структуры
Name.lStructSize = sizeof ( OPENFILENAME );
// задается указатель на окно-владелец
Name.hwndOwner = hWnd;
// инициализируются указатели
Name.lpstrFilter = szFilter;
Name.lpstrFile=szFile;
```

```
// задается максимальная длина имени выбранного файла
Name.nMaxFile = sizeof(szFile);
// инициализация начального каталога для выбора файлов
Name.InitialDir="c:\\windows\\";
// указатель буфера с нестандартным заголовоком диалогового окна
Name.lpstrTitle="Новое название окна";
// задается набор флагов, которые определяют вид окна и особенности его работы
// нельзя выбрать файл из несуществующего каталога или несуществующий файл
Name.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST |
// отключение опции «Только чтение»
OFN_HIDEREADONLY |
// можно выбрать несколько файлов
OFN_ALLOWMULTISELECT;  .
```

При работе с окном сохранения используется функция ПолучитьИмяСохраняемогоФайла (< УказательНаСтруктуруТипаOPENFILENAME >)

BOOL **GetSaveFileName**(LPOPENFILENAME *lpofn*) ,

подключаемая в <commdlg.h>.

Для вызова функции GetSaveFileName необходимо в качестве параметра передать указатель на структуру типа OPENFILENAME. Для корректной работы функции указанная структура должна быть проинициализирована аналогично описанной в GetOpenFileName.

При работе с диалоговым окном Сохранение в структуре OPENFILENAME дополнительно используется значение флага (параметр Flags) OFN_OVERWRITEPROMPT. При его установке, если пользователь выбрал уже существующий файл для сохранения, то по нажатии кнопки Сохранить на экран будет выведено сообщение о том, что файл с таким именем уже существует и пользователь должен подтвердить перезапись файла.

Типовые действия. При работе пользователя с диалоговым окном Открытия можно выделить несколько типовых действий, совершаемых пользователем. Ниже описаны изменения в структуре данных, которые соответствуют действиям пользователя:

1. Пользователь перешел в каталог с нужным ему файлом и, выделив его, нажал кнопку «Открыть» или напрямую в строке имени файла указал полное имя существующего файла. В результате обработки события функция GetOpenFileName возвращает истинное значение, окно закрывается и полное имя выбранного файла (группы файлов) сохраняется в поле lpstrFile, а имя файла (без указания пути) сохраняется в поле lpstrFileTitle структуры типа OPENFILENAME.

Если пользователь выбрал более одного файла (эта возможность определяется наличием значения флага OFN_ALLOWMULTISELECT в наборе флагов Flags структуры OPENFILENAME), то строка lpstrFile будет содержать путь к директории и последовательное указание имен всех выбранных файлов, разделенных пробелом.

2. Пользователь нажал кнопку Отмена или произошла ошибка при открытии указанного файла. В результате обработки события функция GetOpenFileName возвращает ложное значение и значения полей lpstrFile и lpstrFileTitle не изменяются. Полную информацию об ошибке можно получить с помощью функции CommDlgExtendedError.

3. Пользователь указал не существующий каталог или имя несуществующего файла. Обработка такого события зависит от предварительной настройки диалогового окна. Ес-

ли в поле Flags структуры типа OPENFILENAME были указаны значения OFN_FILEMUSTEXIST и OFN_PATHMUSTEXIST, то на экране появится предупреждающее сообщение и пользователю будет предложено произвести повторный выбор. В противном случае в буфере lpstrFile будет сохранено имя несуществующего файла.

4. Пользователь изменил строку фильтра файлов. Обработка такого события производится путем выборки файлов текущей директории по новой маске, а номер выбранной строки фильтра сохраняется в поле nFilterIndex, где нумерация ведется, начиная с единицы.

Если функция GetOpenFileName вернула ложное значение, то произошла ошибка. Дополнительную информацию об ошибках можно получить, используя функцию

DWORD **CommDlgExtendedError** (void) ,

которая возвращается целое число, кодирующее ошибку.

Список наиболее часто встречающихся ошибок представлен в таблице ниже

При работе пользователя с диалоговым окном Сохранения можно выделить несколько типовых действий, совершаемых пользователем. Ниже описаны изменения в структуре данных, которые соответствуют действиям пользователя:

Таблица 2 – Коды ошибок

Код ошибки	Описание
FNERR_BUFFERTOOSMALL	не хватает памяти поля *lpstrFile* для хранения значения, указанного пользователем
FNERR_INVALIDFILENAME	неправильное имя файла
CDERR_DIALOGFAILURE	диалоговое окно не может быть создано (неверные параметры инициализации)
CDERR_FINDRESFAILURE	аварийный вызов диалогового окна, т.к. не найден ресурс
CDERR_INITIALIZATION	ошибка инициализации диалогового окна (как правило, недостаток свободной памяти)
CDERR_LOADRESFAILURE	ошибка при загрузке ресурса

1. Пользователь перешел в нужный директорий и указал имя еще несуществующего файла или указал полное имя еще несуществующего файла вручную и нажал кнопку Сохранить. В результате обработки этого события функция GetSaveFileName вернёт истинное значение, диалоговое окно закрывается, полное имя файла сохраняется в поле lpstrFile, а имя файла (без указания пути) сохраняется в поле lpstrFileTitle структуры типа OpenFileName.

2. Пользователь нажал кнопку Отмена. В результате обработки события функция GetSaveFileName возвращает ложное значение, а значения полей lpstrFile и lpstrFileTitle не меняются.

3. Пользователь указал несуществующий каталог. Обработка события зависит от предварительной настройки диалогового окна. Так, если в поле Flags структуры типа OPENFILENAME было указано значение OFN_FILEMUSTEXIST, то на экране появится предупреждающее сообщение и пользователю будет предложено произвести повторный выбор.

4. Пользователь изменил строки фильтра файлов. В результате обработки события производится выборка файлов текущей директории по новой маске, а номер выбранной строки фильтра сохраняется в поле nFilterIndex.

5. Пользователь выбрал или указал полное имя уже существующего файла. Обработка такого события зависит от предварительной настройки. Так, если в поле Flags струк-

туры типа OPENFILENAME было указано значение OFN_OVERWRITEPROMPT, то на экране появится предупреждающее сообщение и пользователю будет предложено подтвердить перезапись файла.

2.3. Примеры использования

Пример. Создается экземпляр MFC-класса CFileDialog. Передаваемая конструктору строка фильтра определяет, что стандартное диалоговое окно открытия файлов создаст фильтр для отображения только файлов с расширением *.ddt, хотя пользователь вправе выбрать и фильтр *.* ("All files" – "все файлы"). Диалоговое окно активизируется вызовом метода CDialog::DoModal() базового класса

```
CString strFilter = "Data Files (*.ddt)|*.ddt|All Files (*.*)|*.*||";
CFileDialog myFileDialog( TRUE, NULL, NULL,
        OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT, strFilter);
int Code = myFileDialog.DoModal( );  .
```

Пример. Создается приложение в процедурном стиле с окном с рамкой. По сообщению WM_ LBUTTONDOWN выводится окно типа "Сохранить как" для получения пользовательского указания места в файловой системе компьютера и имени файла. По сообщению WM_RBUTTONDOWN выводится окно типа "Открыть" для выбора открываемого файла.

Соответственно для навигации по структуре папок и выбора нужного файла для открытия используется функция GetOpenFileName, поддерживающая окно типа "Открыть", а для сохранения файла используется функция GetSaveFileName, поддерживающая окно типа "Сохранить как".

Примерный текст приложения приведен ниже. В качестве обработки можно выводить выбор пользователя в окне MessageBox.

```
#include <commdlg.h>
...
char szFileName[128];
char szFilter[] = "Файлы данных (*.doc) \0*.dat\0Все файлы (*.*)\0*.*\0";
LRESULT CALLBACK   WndProc(HWND, UINT, WPARAM, LPARAM);
...
int  WINAPI WinMain (...) {  ... }
LRESULT CALLBACK WndProc (HWND hWnd, UINT Message,
                          WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    OPENFILENAME ofn;
    switch (Message)
    {
    case WM_RBUTTONDOWN:
        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lStructSize = sizeof(OPENFILENAME);
        ofn.hwndOwner = hWnd;
```

```
            ofn.lpstrFilter = szFilter;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = sizeof(szFileName);
            ofn.Flags=OFN_PATHMUSTEXIST|OFN_FILEMUSTEXIST;
            if (GetOpenFileName(&ofn) )
            {
                  //Команды по обработке файла с именем szFileName
                  break;
            }
            else
                  break;
      case WM_LBUTTONDOWN:
            memset(&ofn, 0, sizeof(OPENFILENAME));
            ofn.lStructSize = sizeof(OPENFILENAME);
            ofn.hwndOwner=hWnd;
            ofn.lpstrFilter = szFilter;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = sizeof(szFileName);
            ofn.Flags=OFN_PATHMUSTEXIST|OFN_FILEMUSTEXIST;
            if (GetSaveFileName(&ofn) )
            {
                  //Команды по обработке файла с именем szFileName
                  break;
            }
            else
                  break;
      case WM_PAINT:
            ...
      case WM_DESTROY:
            PostQuitMessage(0);
            break;
      default:
            return (DefWindowProc(hWnd, Message, wParam, lParam));
      }
      return 0;
}
```

Пример. Создается приложение в процедурном стиле с окном с рамкой. По сообщению WM_RBUTTONDOWN визуализируется диалоговое окно типа "Открыть" с последующим открытием выбранного файла, считыванием и выводом записей файла в виде строк в окне сообщений. По сообщению WM_LBUTTONDOWN визуализируется диалоговое окно типа "Сохранить как" с последующим сохранением записей из строк "Запись1", "Запись2", "Запись3", "Запись4", "Запись50" в файле с заданным именем в окне Сохранения.

Для открытия выбранного файла может использоваться функция CreateFile. Например, для открытия файла с именем szFileName для выполнения операций чтения

```
hFile = CreateFile(szFileName, GENERIC_READ, 0,NULL, OPEN_EXISTING,0,NULL) ,
```

а для открытия файла с именем szFileName для выполнения операций записи

```
hFile = CreateFile(szFileName, GENERIC_READ | GENERIC_WRITE, 0, NULL,
                   CREATE_ALWAYS, 0, NULL) .
```

Для чтения и записи данных используются функции ReadFile, WriteFile. Например, для чтения одной записи – строки из файла szFileName (дескриптор hFile) во внутреннюю строку char StrIn

```
ReadFile(hFile, StrIn, 9, &dwCount, NULL); ,
```

а для записи строки в файл используется команда

```
WriteFile(hFile, StrOut, sizeof( *StrOut ), &dwCount, NULL); .
```

Примерный текст приложения приведен ниже.

```
#include "stdafx.h"
#include <string.h>
#include <stdlib.h>
#include <wingdi.h>
//#include  "resource.h"
#include <commdlg.h>
char szFile[128];
LRESULT CALLBACK       WndProc(HWND, UINT, WPARAM, LPARAM);
char szProgName[ ]="ProgName";
int  WINAPI WinMain(...)  {  ... }
LRESULT CALLBACK WndProc(HWND hWnd, UINT messg, WPARAM wParam,
                                        LPARAM lParam)
{
        HDC hdc;
        PAINTSTRUCT ps;
        OPENFILENAME ofn;
        char szFilter[ ] = "Файлы данных (*.DAT) \0*.dat\0Все файлы (*.*)\0*.*\0";
        DWORD nCnt;
        char StrOut[5][9] = {"Запись1","Запись2","Запись3","Запись4","Запись50"};
        char StrIn[5][7] = {"","","","","",""};
        int i;
        DWORD dwCount;
        HANDLE hFile;
        switch (messg)
        {
        case WM_RBUTTONDOWN:
                memset(&ofn, 0, sizeof(OPENFILENAME));
                ofn.lStructSize = sizeof(OPENFILENAME);
                ofn.hwndOwner=hWnd;
                ofn.lpstrFilter = szFilter;
                ofn.lpstrFile = szFile;
                ofn.nMaxFile = sizeof(szFile);
                ofn.Flags=OFN_PATHMUSTEXIST|OFN_FILEMUSTEXIST;
                if (GetOpenFileName(&ofn) )
```

```
        {
            hFile = CreateFile(szFile, GENERIC_READ, 0, NULL, OPEN_EXISTING,
                                0, NULL);
            for (i=0; i<5; i++)
            {
                ReadFile(hFile, Strin+i, 9, &dwCount, NULL);
                MessageBox(hWnd,(char*)(Strin+i),"Считана запись",MB_OK);
            };
            CloseHandle(hFile);
            break;
        }
        else
            break;
    case WM_LBUTTONDOWN:
        memset(&ofn, 0, sizeof(OPENFILENAME));
        ofn.lStructSize = sizeof(OPENFILENAME);
        ofn.hwndOwner=hWnd;
        ofn.lpstrFilter = szFilter;
        ofn.lpstrFile = szFile;
        ofn.nMaxFile = sizeof(szFile);
        ofn.Flags = OFN_PATHMUSTEXIST|OFN_FILEMUSTEXIST;
        if (GetSaveFileName(&ofn) )
        {
            hFile = CreateFile(szFile, GENERIC_READ | GENERIC_WRITE,
                0, NULL, CREATE_ALWAYS, 0, NULL);
            for (i=0;i<5;i++)
            {
               WriteFile(hFile, StrOut+i, /*sizeof(*(StrOut+i))*/ 9, &dwCount, NULL);
            };
            CloseHandle(hFile);
            break;
        }
        else
            break;
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
            ......
        EndPaint(hWnd, &ps);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc(hWnd, messg, wParam, lParam));
    }
    return 0;
} .
```

2.4. Окно поиска-замены

При работе с окном средствами MFC используется класс CFindReplaceDialog. Он отвечает за создание диалоговых окон для поиска строк (Find) и для поиска-замены строк (Find-Replace). Это типовое окно является немодальным, в отличие от других типовых диалоговых окон, реализованных в MFC.

Ниже представлено описание класса CFindReplaceDialog, поддерживающего функциональность рассматриваемого окна.

```
class CFindReplaceDialog : public CCommonDialog
{
    DECLARE_DYNAMIC(CFindReplaceDialog)
public:
// Attributes
    FINDREPLACE m_fr;
// Constructors
    CFindReplaceDialog();
    BOOL Create(BOOL bFindDialogOnly, LPCTSTR lpszFindWhat,
                LPCTSTR lpszReplaceWith = NULL,
                DWORD dwFlags = FR_DOWN,
                CWnd* pParentWnd = NULL);
    static CFindReplaceDialog* PASCAL GetNotifier(LPARAM lParam);
// Operations
    CString GetReplaceString() const;
    CString GetFindString() const;
    BOOL SearchDown() const;
    BOOL FindNext() const;
    BOOL MatchCase() const;
    BOOL MatchWholeWord() const;
    BOOL ReplaceCurrent() const;
    BOOL ReplaceAll() const;
    BOOL IsTerminating() const;
// Implementation
protected:
    virtual void PostNcDestroy();
protected:
    TCHAR m_szFindWhat[128];
    TCHAR m_szReplaceWith[128];
};
```

Член-данное класса FINDREPLACE **m_fr** представляет собой структуру, в которой хранится описание текущего состояния настроек диалогового окна. Описание структуры FINDREPLACE представлено ниже

```
typedef struct tagFINDREPLACEA
{
    DWORD           lStructSize;
    HWND            hwndOwner;
```

```
    HINSTANCE        hInstance;
    DWORD            Flags;
    LPSTR            lpstrFindWhat;
    LPSTR            lpstrReplaceWith;
    WORD             wFindWhatLen;
    WORD             wReplaceWithLen;
    LPARAM           lCustData;
    LPFRHOOKPROC     lpfnHook;
    LPCSTR           lpTemplateName;
} FINDREPLACE;  ,
```

где

lStructSize – размер структуры в байтах;

hwndOwner – дескриптор окна-владельца диалогового окна (или NULL);

lpstrFindWhat – указатель на буфер, содержащий строку, введенную пользователем в поле поиска (Find What); при этом место под буфер (не менее 80 символов) выделяется либо динамически либо в глобальном или статическом массиве;

lpstrReplaceWith – указатель на буфер, который содержит строку, введенную пользователем в строке замены (Replace with – заменить на); память под буфер так же, как и в поле *lpstrFindWhat*, должна быть выделена заблаговременно;

wFindWhatLen – размер буфера, на который указывает переменная *lpstrFindWhat*, в байтах;

wReplaceWithLen – размер буфера, на который указывает переменная *lpstrReplaceWith*, в байтах;

Flags – настройки (флаги) инициализации диалогового окна, которые можно использовать в виде комбинации следующих значений:

- FR_DIALOGTERM для включения сообщений о закрытии окна;
- FR_DOWN переключатель направления поиска (при инициализации устанавливается в положение Down – Вниз, указывая, что поиск будет осуществляться от позиции курсора до конца документа; в противном случае переключатель устанавливается в положение Up – Вверх);
- FR_FINDNEXT фиксирует, что пользователь нажал кнопку Next – Следующий, при этом поле lpstrFindWhat содержит строку для поиска;
- FR_HIDEUPDOWN отключает отображение в окне переключателя направления поиска;
- FR_HIDEMATCHCASE отключает отображение в окне флажка «Учитывать регистр» (Match case);
- FR_HIDEWHOLEWORD предписывает не отображать в окне флажок искать «Только слово целиком» (Match Whole Word Only);
- FR_MATCHCASE устанавливает при инициализации окна флажок «Учитывать регистр»;
- FR_NOMATCHCASE сбрасывает при инициализации окна флажок «Учитывать регистр»;
- FR_NOUPDOWN блокирует при инициализации окна переключатель направления поиска;

- FR_NOWHOLEWORD блокирует при инициализации окна флажок «Только слово целиком»;

- FR_REPLACE указывает, что если пользователь нажал кнопку «Заменить», то при этом в поле *lpstrFindWhat* хранится искомая подстрока, а в поле *lpstrReplaceWith* хранится строка, на которую нужно заменить искомую строку;

- FR_REPLACEALL указывает, что пользователь нажал кнопку «Заменить все»;

- FR_SHOWHELP включает отображение в диалоговом окне кнопки получения справочной информации Help (?);

- FR_WHOLEWORD устанавливает при инициализации окна флажок «Только слово целиком».

Базовые методы класса:

- конструктор объекта класса, который, как и для других немодальных окон, необходимо создавать динамически

CFindReplaceDialog::**CFindReplaceDialog**( ) ;

- метод, позволяющий создать объект

CFindReplaceDialog:: **Create** (BOOL *bFindDialogOnly*, LPCTSTR *lpszFindWhat*,
LPCTSTR *lpszReplaceWith* = NULL, DWORD *dwFlags* = FR_DOWN,
CWnd **pParentWnd*=NULL) ,

здесь, если флаг *bFindDialogOnly* = TRUE, то создается окно поиска, иначе – окно поиска-замены;

- метод, позволяющий продолжать поиск, если это требуется

BOOL CFindReplaceDialog::**FindNext** ( ) ;

- статический метод, возвращающий указатель на текущий объект окна поиска-замены для получения возможности доступа к его функциям и структуре *m_fr*

static CFindReplaceDialog* CFindReplaceDialog::**GetNotifier** (LPARAM *lparam*) ;

- метод, возвращающий заданную по умолчанию строку для поиска

CString CFindReplaceDialog::**GetFindString** ( ) ;

- метод, возвращающий заданную по умолчанию строку для поиска, на которую будут заменяться найденные строки

CString CFindReplaceDialog::**GetReplaceString** ( ) ;

- метод, позволяющий определить – завершает ли пользователь работу с диалоговым окном (если результат TRUE, то необходимо вызвать метод DestroyWindow для разрушения окна и установить указатель на объект класса диалога в NULL)

BOOL CFindReplaceDialog::**IsTerminating** ( ) ;

- метод, позволяющий определить – хочет ли пользователь при поиске учитывать регистр

BOOL CFindReplaceDialog::**MatchCase** ( ) ;

- метод, позволяющий определить – хочет ли пользователь осуществить поиск всего слова целиком

BOOL CFindReplaceDialog::**MatchWholeWord** ( ) ;

- метод, позволяющий определить – хочет ли пользователь осуществить автоматическую замену для всех найденных подстрок

BOOL CFindReplaceDialog::**ReplaceAll** ( ) ;

- метод, позволяющий определить – хочет ли пользователь заменить найденную подстроку

BOOL CFindReplaceDialog::**ReplaceCurrent** ( ) ;

- метод, позволяющий определить направление поиска (если TRUE, то поиск осуществляется по направлению к концу документа, FALSE – поиск осуществляется по направлению к началу документа)

BOOL CFindReplaceDialog::**SearchDown** ( ) .

2.5. Окно настройки шрифта

При работе с этим окном средствами MFC используется класс CFontDialog. Он отвечает за создание и обеспечение функционирования стандартного окна настройки шрифта. Внешний вид окна представлен на рисунке ниже.

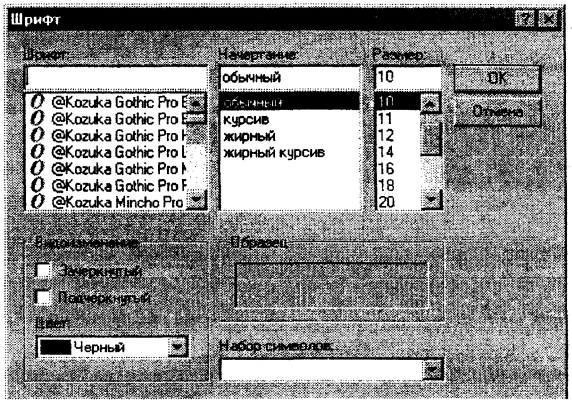


Рисунок 7 – Диалоговое окно настройки шрифта

Ниже представлено описание класса CFontDialog, поддерживающего функциональность рассматриваемого окна.

```
class CFontDialog : public CCommonDialog
{
    DECLARE_DYNAMIC(CFontDialog)
public:
// Attributes
    // font choosing parameter block
    CHOOSEFONT m_cf;
// Constructors
    CFontDialog(LPLOGFONT lplfInitial = NULL,
            DWORD dwFlags = CF_EFFECTS | CF_SCREENFONTS,
            CDC* pdcPrinter = NULL,  CWnd* pParentWnd = NULL);
```

```
// Operations
        virtual int DoModal( );
        void GetCurrentFont(LPLOGFONT lplf);
        CString GetFaceName( ) const;
        CString GetStyleName( ) const;
        int GetSize( ) const;
        COLORREF GetColor( ) const;
        int GetWeight( ) const;
        BOOL IsStrikeOut( ) const;
        BOOL IsUnderline( ) const;
        BOOL IsBold( ) const;
        BOOL IsItalic( ) const;
// Implementation
        LOGFONT m_lf;
protected:
        TCHAR m_szStyleName[64];
};
```

Член-данное класса *CHOOSEFONT* **m_cf** – основная структура, в которой хранится описание текущих настроек диалогового окна. Описание структуры CHOOSEFONT представлено ниже

```
typedef struct tagCHOOSEFONTA
{
        DWORD       lStructSize;
        HWND        hwndOwner;
        HDC         hDC;
        LPLOGFONTA      lpLogFont;
        INT             iPointSize;
        DWORD       Flags;
        COLORREF    rgbColors;
        LPARAM      lCustData;
        LPCFHOOKPROC    lpfnHook;
        LPCSTR          lpTemplateName;
        HINSTANCE   hInstance;
        LPSTR       lpszStyle;
        WORD        nFontType;
        WORD            __MISSING_ALIGNMENT__;
        INT         nSizeMin;
        INT         nSizeMax;
}   CHOOSEFONT; .
```

Здесь:

lStructSize – размер структуры в байтах;

hwndOwner – дескриптор окна, которое владеет блоком диалога (или NULL);

hDC – дескриптор контекста устройства (принтера), шрифты которого должны быть отражены в окне (это поле используется только, если в поле *Flags* установлен один из флагов CF_BOTH или CF_PRINTERFONTS);

lpLogFont – указатель на структуру *LOGFONT*, которая содержит значения, определяющие параметры шрифта (если при создании окна был установлен флаг CF_INITTOLOGFONTSTRUCT, то параметры этой структуры будут использоваться в качестве начальных, а при завершении работы с диалоговым окном поля этой структуры будут заполнены параметрами шрифта, установленными пользователем;

iPointSize – определяет размер выбранного шрифта в десятых долях пункта (это поле заполняется после закрытия окна);

rgbColors – фиксирует цвет символов шрифта, который будет использован при отображении окна (должен быть установлен флаг CF_EFFECTS);

lpszStyle – указатель на буфер, содержащий данные стиля шрифта (если установлен флаг CF_USESTYLE, то эта строка используется для инициализации стиля шрифта);

nSizeMin определяет минимальный размер шрифта, который пользователь может выбрать (должен быть установлен флаг CF_LIMITSIZE);

nSizeMax определяет максимальный размер шрифта, который пользователь может выбрать (должен быть установлен флаг CF_LIMITSIZE);

nFontType определяет тип выбранного шрифта, позволяя использовать одно из следующих значений:

- BOLD_FONTTYPE – полужирное начертание (эта информация продублирована в поле lfWeight структуры LOGFONT и эквивалентна значению флага FW_BOLD;
- ITALIC_FONTTYPE – курсивное начертание, равносильное значению флага FW_ITALIC;
- REGULAR_FONTTYPE – обычное начертание, равносильное значению флага FW_REGULAR;
- PRINTER_FONTTYPE – использование принтерного шрифта;
- SCREEN_FONTTYPE – экранный шрифт;
- SIMULATED_FONTTYPE – включает возможность GDI для эмулирования заданного шрифта;

Flags – флаги инициализации (они также определяют выбор пользователя после закрытия окна диалога), включающие комбинации следующих значений:

- CF_APPLY предписывает отображение в окне кнопки «Применить»;
- CF_ANSIONLY определяет наличие в списке доступных шрифтов, только шрифтов в кодировке ANSI (для Windows NT определяет, что в списке будут шрифты, использующие наборы символов Windows и Unicode);
- CF_BOTH указывает, что в списке шрифтов присутствуют как экранные так и принтерные шрифты;
- CF_TTONLY указывает, что в списке шрифтов отображаются только масштабируемые шрифты (TrueType);
- CF_EFFECTS определяет необходимость отображения в окне диалога элементов управления, с помощью которых можно выбирать цвет букв, создавать подчеркнутые и другие шрифты (для этого необходимо проинициализировать поле *rgbColor* и поля *lfStrikeOut* и *lfUnderLine* структуры LOGFONT);
- CF_FIXEDPITCHONLY разрешает выбирать только шрифты с фиксированной шириной символов;
- CF_FORCEFONTEXIST приводит к выводу сообщений об ошибке, если пользователь пытается выбрать несуществующий шрифт;

- CF_INITTOLOGFONTSTRUCT определяет использование при инициализации диалогового окна содержимого структуры LOGFONT, адрес которой передается через параметр *lpLogFont*;

- CF_LIMITSIZE предписывает диалоговому окну учитывать содержимое полей *nSizeMin* и *nSizeMax*;

- CF_NOOEMFONT запрещает выбирать векторные шрифты;

- CF_NOFACESEL запрещает отображать имя шрифта при использовании структуры *LOGFONT* при инициализации окна;

- CF_NOSTYLESEL запрещает отображать стиль шрифта при использовании структуры *LOGFONT* при инициализации окна;

- CF_PRINTERFONTS предписывает отображать только шрифты, поддерживаемые принтером, контекст которого задан в поле *hDC*;

- CF_SCALABLEONLY разрешает выбор только масштабируемых шрифтов, которые включают и векторные, масштабируемые шрифты принтера, шрифты TrueType;

- CF_SCREENFONTS предписывает отображать только экранные шрифты, поддерживаемые системой;

- CF_SHOWHELP предписывает отображать кнопку в диалоговом окне кнопку Help (?);

- CF_USESTYLE определяет, что поле *lpszStyle* указывает на буфер, содержащий данные стиля, который используется при инициализации окна;

- CF_WYSIWYG разрешает выбирать только те шрифты, которые доступны и для отображения на экране и для печати на принтере (при этом должны быть установлены флаги CF_BOTH и CF_SCALABLEONLY).

Ниже представлено описание структуры LOGFONT

```
typedef struct tagLOGFONTW
{
    LONG    lfHeight;
    LONG    lfWidth;
    LONG    lfEscapement;
    LONG    lfOrientation;
    LONG    lfWeight;
    BYTE    lfItalic;
    BYTE    lfUnderline;
    BYTE    lfStrikeOut;
    BYTE    lfCharSet;
    BYTE    lfOutPrecision;
    BYTE    lfClipPrecision;
    BYTE    lfQuality;
    BYTE    lfPitchAndFamily;
    WCHAR   lfFaceName[LF_FACESIZE];
} LOGFONT; .
```

Здесь:

lfHeight определяет высоту знакомест или символов шрифта (назначение этого поля может интерпретироваться в зависимости от конкретного значения следующим образом –

значение, отличное от нуля, вызывает преобразование высоты в единицы устройства вывода и используется для выбора шрифта из имеющихся, а значение ноль применяется для нахождения шрифта со значением высоты по умолчанию. То есть при поиске необходимого шрифта выбирается шрифт с наиболее подходящей высотой знакоместа или символа, где высота должна быть меньшей или равной запрашиваемой);

lfWidth определяет среднюю ширину символов в логических единицах;

lfEscapment определяет угол (в десятых долях градуса) между базовой линией выводимого текста и горизонталью;

lfOrientation определяет угол (в десятых долях градуса) между базовой линией каждого выводимого символа и горизонталью;

lfWeight определяет толщину линии начертания сиволов и может принимать значения от 0 до 1000; (0 – по умолчанию), например, в соответствии с таблицей значений, приведенной ниже;

Таблица 3 – Толщина линии начертания сиволов

FW_DONTCARE	0	FW_MEDIUM	500
FW_THIN	100	FW_SEMIBOLD FW_DEMIBOLD	600
FW_EXTRALIGHT FW_ULTRALIGHT	200	FW_BOLD	700
FW_LIGHT	300	FW_EXTRABOLD FW_ULTRABOLD	800
FW_NORMAL FW_REGULAR	400	FW_HEAVY FW_BLACK	900

lfItalic задает курсивное начертание шрифта;

lfUnderline задает подчеркнутое начертание шрифта;

lfStrikeOut задает перечеркнутое начертание шрифта;

lfCharSet определяет таблицу кодировки шрифта (определены следующие таблицы: ANSI_CHARSET, DEFAULT_CHARSET, SYMBOL_CHARSET, SHIFTJIS_CHARSET, GB2312_CHARSET, HANGEUL_CHARSET, CHINESEBIG5_CHARSET, OEM_CHARSET, JOHAB_CHARSET, HEBREW_CHARSET, ARABIC_CHARSET, GREEK_CHARSET, TURKISH_CHARSET, THAI_CHARSET, EASTEUROPE_CHARSET, RUSSIAN_CHARSET, MAC_CHARSET, BALTIC_CHARSET);

lfOutPrecision определяет механизм выбора шрифта, который должен соответствовать заданным параметрам (типу, ширине, высоте и т.п.);

lfClipPresicion определяет механизм отсечения символов шрифта в случае, если символы оказываются частично вне области вывода;

lfQuality определяет желаемое качество вывода, которое зависит и от выбранного шрифта;

lfPitchAndFamily определяет тип и семейство шрифтов;

lfFaceName определяет имя шрифта (до 32 символов).

Базовые методы класса:

- конструктор класса

CFontDialog:: **CFontDialog** (LPLONGFONT *lplfInitial*,
DWORD *dwFlags* = CF_EFFECTS | CF_SCREENFONTS,
CDC **pdcPrinter* = NULL, CWnd **pParentWnd*=NULL) ;

- метод, отвечающий за вывод на экран модального диалогового окна (код завершения операции настройки шрифта IDOK означает, что настройка шрифта завершена, а IDCANCEL – настройка шрифтов отменена),

virtual int CFontDialog::**DoModal** ( ) ;

- метод, заполняющий поля структуры LOGFONT характеристиками текущего выбранного шрифта,

void CFontDialog::**GetCurrentFont** (LPLOGFONT *lplf*) ;

- метод, позволяющий получить имя выбранного шрифта

CString CFontDialog::**GetFaceName** ( ) ;

- метод, позволяющий получить стиль выбранного шрифта

CString CFontDialog::**GetStyleName** ( ) ;

- метод, позволяющий получить размер (в десятых долях пункта) выбранного шрифта

int CFontDialog::**GetSize** ( ) ;

- метод, позволяющий получить цвет выбранного шрифта

COLORREF CFontDialog::**GetColor** ( ) ;

- метод, позволяющий получить вес выбранного шрифта

int CFontDialog::**GetWeight** ( ) ;

- метод, устанавливающий, будет ли выбранный шрифт зачеркнутым

BOOL CFontDialog::**IsStrikeOut** ( ) ;

- метод, устанавливающий, будет ли выбранный шрифт подчеркнутым

BOOL CFontDialog::**IsUnderLine** ( ) ;

- метод, устанавливающий, будет ли выбранный шрифт наклонным

BOOL CFontDialog::**IsItalic** ( ) .

Настройка диалогового окна. Работать с диалоговым окном можно, в простейшем случае, просто создав объект описанного класса CFontDialog. Однако для использования всех имеющихся возможностей надо заполнить поля структуры m_cf типа CHOOSEFONT.

Ниже приведен фрагмент настройки диалогового окна выбора шрифтов и заполнения полей структуры m_cf.

```
#include "afxdlgs.h"
CFontDialog FontDlg;
//Флаги
FontDlg.m_cf.Flags |= CF_APPLY;
FontDlg.m_cf.Flags |= CF_BOTH;
FontDlg.m_cf.Flags |= CF_EFFECTS;
```

```
FontDlg.m_cf.Flags |= CF_INITTOLOGFONTSTRUCT;
FontDlg.m_cf.Flags |= CF_LIMITSIZE;
FontDlg.m_cf.Flags |= CF_USESTYLE;
//Ограничения на размер шрифта
FontDlg.m_cf.nSizeMin = 12;
FontDlg.m_cf.nSizeMax = 18;
//Начальный цвет шрифта
FontDlg.m_cf.rgbColors = 0; .
```

Управление диалоговым окном. После создания объекта класса CFontDialog и необязательной предварительной настройки диалог запускается путем вызова метода DoModal. Возвращаемое методом значение IDOK указывает на тот факт, что пользователь осуществил выбор шрифта. Обработка результата осуществляется считыванием полей структуры m_cf типа CHOOSEFONT, где после выбора пользователя будет сохранена информация о выбранном файле, а также при помощи вызова методов самого класса CFontDialog.

2.6. Окно настройки параметров страницы

При работе с окном средствами MFC используется класс CPageSetupDialog. Он отвечает за создание и функционирование стандартного модального диалогового окна PageSetup Dialog (Параметры страницы) с дополнительной поддержкой установки и модификации полей печати. Настройка объекта указанного класса осуществляется с помощью структуры PAGESETUPDLG.

Внешний вид окна представлен на рисунке ниже.

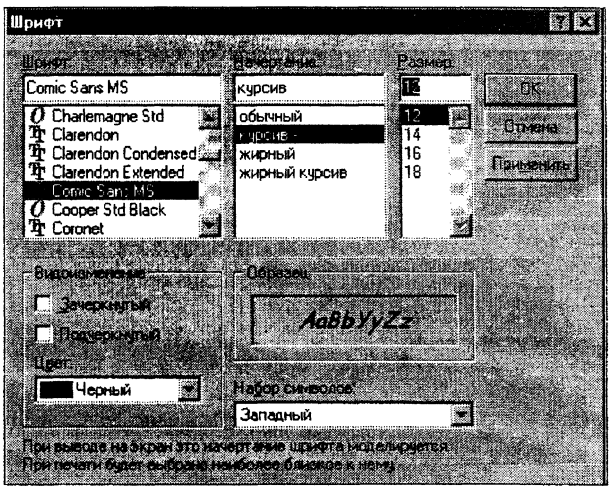
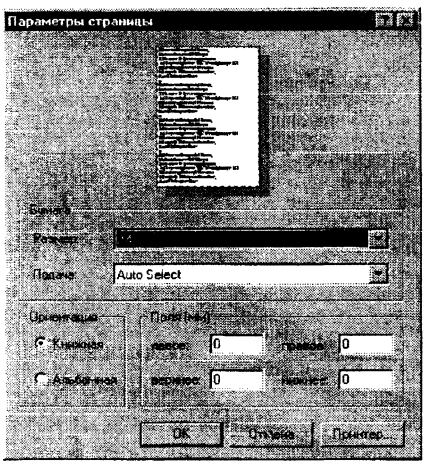


Рисунок 8 – Вид типового диалогового окна настройки шрифта

Ниже представлено описание класса CPageSetupDialog, поддерживающего функциональность рассматриваемого окна

```
class CPageSetupDialog : public CCommonDialog
{
    DECLARE_DYNAMIC(CPageSetupDialog)
public:
// Attributes
    PAGESETUPDLG m_psd;
// Constructors
    CPageSetupDialog(DWORD dwFlags = PSD_MARGINS
                    | PSD_INWININIINTLMEASURE,  CWnd* pParentWnd = NULL);
// Attributes
    LPDEVMODE GetDevMode() const;
    CString GetDriverName() const;
    CString GetDeviceName() const;
    CString GetPortName() const;
    HDC CreatePrinterDC();
    CSize GetPaperSize() const;
    void GetMargins(LPRECT lpRectMargins, LPRECT lpRectMinMargins) const;
// Operations
    virtual int DoModal();
// Overridables
    virtual UINT PreDrawPage(WORD wPaper, WORD wFlags,
                             LPPAGESETUPDLG pPSD);
    virtual UINT OnDrawPage(CDC* pDC, UINT nMessage, LPRECT lpRect);
// Implementation
protected:
    static UINT CALLBACK PaintHookProc(HWND hWnd, UINT message, WPARAM
                                       wParam, LPARAM lParam);
};
```



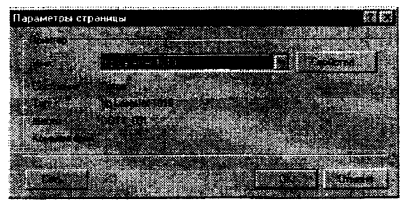


Рисунок 9 – Вид типового диалогового окна настройки параметров страницы

Член-данное класса PAGESETUPDLG **m_psd** – структура, в которой хранится описание текущего состояния настроек диалогового окна. Описание структуры PAGESETUPDLG приведено ниже

```
typedef struct tagPSDA
{
    DWORD        lStructSize;
    HWND         hwndOwner;
    HGLOBAL      hDevMode;
    HGLOBAL      hDevNames;
    DWORD        Flags;
    POINT        ptPaperSize;
    RECT         rtMinMargin;
    RECT         rtMargin;
    HINSTANCE    hInstance;
    LPARAM       lCustData;
    LPPAGESETUPHOOK lpfnPageSetupHook;
    LPPAGEPAINTHOOK lpfnPagePaintHook;
    LPCSTR       lpPageSetupTemplateName;
    HGLOBAL      hPageSetupTemplate;
} PAGESETUPDLG;
```

Здесь:

lStructSize – определяет размер структуры в байтах;

hwndOwner – дескриптор окна, которое владеет окном (или NULL);

hDevMode – задает идентификатор глобального блока памяти, который содержит структуру типа DEVMODE, используемую для инициализации параметров принтера (если значение NULL, то функции библиотеки сами запросят память под структуру и, после завершения диалога, запишут туда выбранные параметры принтера);

hDevNames – определяет идентификатор глобального блока памяти, который содержит структуру типа DEVNAMES, определяющую имя драйвера принтера, его имя и порт вывода, к которому подключен принтер (если задано значение NULL, то функции библиотеки сами запросят память под структуру и, после завершения диалога, запишут туда строки, соответствующие выбранному принтеру);

ptPaperSize – определяет размеры страницы, заданные пользователем (здесь флаги PSD_INTHOUSANDSOFINCHES и PSD_INHUNDREDTHSOFMILLIMETERS определяют единицу измерения);

rtMinMargin определяет минимально допустимую ширину левого, верхнего правого и нижнего полей страницы (при этом должен быть установлен флаг PSD_MINMARGINS, а значения полей не должны превышать значения, указанные в *rtMargin);*

rtMargin определяет ширину левого, верхнего, правого и нижнего полей страницы (при этом должен быть установлен флаг PSD_MARGIN), после завершения работы с диалоговым окном в этом поле сохраняются настройки пользователя;

Flags – набор флагов инициализации, которые можно использовать при инициализации окна в виде комбинации следующих значений:

- PSD_DEFAULTMINMARGINS фиксирует минимальные значения, установленные по умолчанию, которые пользователь может применять для задания полей страницы при

печати (игнорируется, если установлен один из флагов PSD_MARGINS, PSD_MINMARGINS);

- PSD_DISABLEMARGINS блокирует использование элементов управления в окне, ответственных за установку полей страницы;

- PSD_DISABLEORIENTATION блокирует использование элементов управления в окне, ответственных за ориентацию страницы;

- PSD_DISABLEPAGEPAINTING блокирует возможность вывода образца страницы в окне;

- PSD_DISABLEPAPER блокирует использование элементов управления в окне, ответственных за установку параметров страницы (размер и источник);

- PSD_DISABLEPRINTER блокирует использование кнопки Printer (Принтер) в диалоговом окне;

- PSD_INHUNDEREDTHSOFMILLIMETERS указывает, что единицей измерения размеров бумаги и полей являются сотые доли миллиметра;

- PSD_INTHOUSANDTHSOFINCHES указывает, что единицей измерения размеров бумаги и полей являются тысячные доли дюйма;

- PSD_MARGINS предписывает использовать значения, указанные в поле *rtMargin*, в качестве начальных для настройки полей страницы (по умолчанию поля устанавливаются равными одному дюйму);

- PSD_MINMARGINS предписывает использовать в качестве минимально допустимых значения для полей страницы, определенные в поле *rtMinMargins* (по умолчанию поля устанавливаются параметрам принтера);

- PSD_NOWARNING отменяет вывод сообщения о том, что в системе не установлен принтер по умолчанию;

- PSD_RETURNDEFAULT предписывает инициализировать структуры полей *hDevNames* и *hDevMode* параметрами принтера, установленного по умолчанию (диалоговое окно не выводится);

- PSD_SHOWHELP предписывает отображать кнопку Help (?).

Базовые методы класса:

- конструктор диалога

CPageSetupDialog::**CPageSetupDialog**(DWORD *dwFlags* = PSD_MARGINS | PS_INWININIINTLMEASURE, CWnd **pParentWnd* = NULL) ;

- метод класса CWinApp, который необходимо вызывать для фиксации текущих установок пользователя. Метод заменяет при значении *bFreeOld* = TRUE текущий принтер на новый, указанный в параметрах *hDevNames* и *hDevMode*. Если оба параметра равны NULL, то используется принтер по умолчанию. После вызова этого метода приложение получает новый контекст устройства для принтера с соответствующими атрибутами

void CWinApp::**SelectPrinter** (HANDLE *hDevNames*, HANDLE *hDevMode*, BOOL *bFreeOld* = TRUE) ;

- метод, отвечающий за создание и доступ к текущему контексту устройства принтера на базе структур DEVMODE и DEVNAMES (окно диалога не выводится)

HDC CPageSetupDialog::**CreatePrinterDC** ( ) ;

- метод, позволяющий получить имя текущего выбранного принтера (вызывается после DoModal)

CString CPageSetupDialog::**GetDeviceName** ( ) ;

- метод, позволяющий получить имя драйвера текущего выбранного принтера (вызывается после DoModal)

CString CPageSetupDialog::**GetDriverName** ( ) ;

- метод, позволяющий получить указатель на структуру типа DEVMODE, которая содержит информацию о конфигурации и параметрах принтера (вызывается после DoModal)

LPDEVMODE CPageSetupDialog::**GetDevMode** ( ) ;

- метод, позволяющий получить имя порта вывода текущего выбранного принтера (вызывается после DoModal)

CString CPageSetupDialog::**GetPortName** ( ) ;

- метод, позволяющий получить значения размеров полей печати текущего выбранного принтера (если какой-либо из параметров не используется, то передается значение NULL, а сами размеры возвращаются в сотых долях миллиметра или тысячных долях дюйма, в зависимости от установленных единиц измерения)

void CPageSetupDialog::**GetMargins** (LPRECT *lpRectMargins*, LPRECT *lpRectMinMargins*) ;

- метод, позволяющий получить значения размеров бумаги для текущего принтера (размерность зависит от установленной единицы измерения)

CSize CPageSetupDialog::**GetPaperSize** ( ) ;

– метод для отображения экранного образа страницы печати в диалоговом окне (по умолчанию выводится изображение страницы текста)

virtual UINT CPageSetupDialog::**OnDrawPage** (CDC **pDC*, UINT Message, LPRECT *lpRect*) ,

где *pDC* – указатель на контекст устройства принтера, *lpRect* – указатель на объект *CRect* или RECT, содержащий координаты перерисовываемой области, *nMessage* – сообщение, определяющее область выводимой страницы. Параметр *nMessage* может принимать следующее значения:

- WM_PSD_FULLPAGERECT для отображения страницы целиком;
- WM_PSD_MINMARGINRECT для отображения области, ограниченной текущими минимальными полями;
- WM_PSD_MARGINRECT для отображения области, ограниченной текущими полями;
- WM_PSD_GREEKTEXTRECT для отображения содержания страницы;
- WM_PSD_ENVSTAMPRECT определяет область, зарезервированную для штампа;
- WM_PSD_YAFULLPAGERECT определяет область до границ образца страницы.

Указанный метод необходимо переопределить, если нужно организовать собственное рисование изображения страницы;

– метод для отображения экранного образа страницы печати в диалоговом окне в режиме предварительного просмотра

virtual UINT CPageSetupDialog::**PreDrawPage** (WORD *wPaper*, WORD *wFlags*, LPPAGESETUPDLG *pPSD*) .

Здесь *wPaper* размер бумаги, *pPSD* указатель на структуру PAGESETUPDLG, *wFlags* – определяет ориентацию бумаги и тип принтера (матричный или Hewlett Packard – HPPCL) и может принимать одно из следующих значений:

0x001 – матричный принтер, горизонтальная ориентация бумаги;
0x003 – принтер HPPCL, горизонтальная ориентация бумаги;
0x005 – матричный принтер, вертикальная ориентация бумаги;
0x007 – принтер HPPCL, вертикальная ориентация бумаги;
0x00d – матричный принтер, горизонтальная ориентация конверта;
0x00b – принтер HPPCL, горизонтальная ориентация конверта;
0x01f – матричный принтер, вертикальная ориентация конверта;
0x019 – принтер HPPCL, вертикальная ориентация конверта.

Настройка диалогового окна. Работать с диалогом можно, в простейшем случае, и просто создав объект описанного класса CPageSetupDialog. Однако воспользоваться всеми предоставляемыми возможностями можно, только заполнив поля структуры m_psd типа PAGESETUPDLG.

Ниже приведен фрагмент настройки диалогового окна настройки параметров страницы и заполнения полей структуры m_psd

```
#include "afxdlgs.h"
CPageSetupDialog PgSetDlg;
PgSetDlg.m_psd.Flags |= PSD_DISABLEMARGINS;
PgSetDlg.m_psd.Flags |= PSD_DISABLEPAPER;
PgSetDlg.m_psd.Flags |= PSD_DISABLEPRINTER;  .
```

Управление диалоговым окном выбора цвета. После создания объекта класса CPageSetupDialog и необязательной предварительной настройки диалог запускается путем вызова метода DoModal. Значение IDOK указывает на тот факт, что пользователь осуществил настройку параметров страницы. Обработка результата осуществляется считыванием полей структуры m_psd типа PAGESETUPDLG, где после выбора пользователя будет сохранена информация о выбранных настройках страницы и принтера, а также при помощи вызова методов самого класса CPageSetupDialog.

Однако в отличие от рассмотренных ранее типовых диалоговых окон, при работе с рассматриваемым окном после завершения работы пользователя с диалогом все изменения, которые были сделаны, не сохраняются. Программист должен позаботиться о сохранении настроек пользователя самостоятельно либо в классе документа, либо в классе приложения.

2.7. Окно настройки печати

При работе с окном средствами MFC используется класс CPrintDialog. Он отвечает за создание двух диалогов: Печать (Print) и Настройка печати (Print setup), с помощью которых пользователь может напечатать документ, выбрать нужный принтер или изменить его настройки. Настройка объекта этого класса осуществляется с помощью структуры типа PRINTDLG.

Внешний вид окна представлен на рисунке ниже.

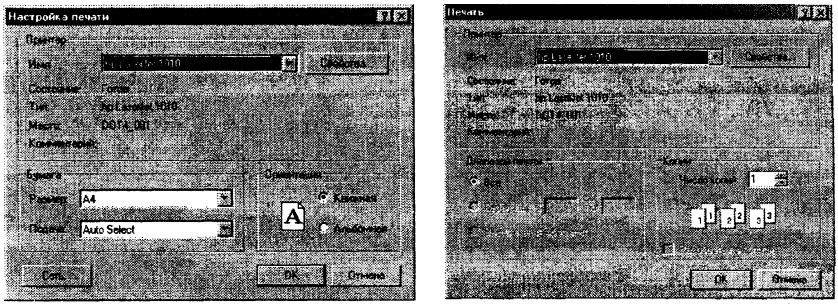


Рисунок 10 – Вид типового диалогового окна настройки печати

Ниже представлено описание класса CPrintDialog, поддерживающего функциональность рассматриваемого окна

```
class CPrintDialog : public CCommonDialog
{
    DECLARE_DYNAMIC(CPrintDialog)
public:
// Attributes
    PRINTDLG& m_pd;
// Constructors
    CPrintDialog(BOOL bPrintSetupOnly, DWORD dwFlags = PD_ALLPAGES
        | PD_USEDEVMODECOPIES | PD_NOPAGENUMS
        | PD_HIDEPRINTTOFILE | PD_NOSELECTION,
        CWnd* pParentWnd = NULL);
// Operations
    virtual int DoModal();
    BOOL GetDefaults();
    int GetCopies() const;
    BOOL PrintCollate() const;
    BOOL PrintSelection() const;
    BOOL PrintAll() const;
    BOOL PrintRange() const;
    int GetFromPage() const;
    int GetToPage() const;
    LPDEVMODE GetDevMode() const;
    CString GetDriverName() const;
    CString GetDeviceName() const;
    CString GetPortName() const;
    HDC GetPrinterDC() const;
    HDC CreatePrinterDC();
// Implementation
private:
    PRINTDLG m_pdActual;
```

```
protected:
        CPrintDialog(PRINTDLG& pdInit);
        virtual CPrintDialog* AttachOnSetup();
        //{{AFX_MSG(CPrintDialog)
        afx_msg void OnPrintSetup();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

Член-данное PRINTDLG **m_pd** – структура, в которой хранится описание текущего состояния настроек диалогового окна. Описание структуры PRINTDLG приведено ниже

```
typedef struct tagPDA {
        DWORD        lStructSize;
        HWND         hwndOwner;
        HGLOBAL      hDevMode;
        HGLOBAL      hDevNames;
        HDC          hDC;
        DWORD        Flags;
        WORD         nFromPage;
        WORD         nToPage;
        WORD         nMinPage;
        WORD         nMaxPage;
        WORD         nCopies;
        HINSTANCE    hInstance;
        LPARAM       lCustData;
        LPPRINTHOOKPROC  lpfnPrintHook;
        LPSETUPHOOKPROC  lpfnSetupHook;
        LPCSTR       lpPrintTemplateName;
        LPCSTR       lpSetupTemplateName;
        HGLOBAL      hPrintTemplate;
        HGLOBAL      hSetupTemplate;
} PRINTDLG; .
```

Здесь:

lStructSize – размер структуры в байтах;

hwndOwner – дескриптор окна-владельца диалогового окна (или NULL);

hDevMode – идентификатор глобального блока памяти, который содержит структуру типа DEVMODE, используемую для инициализации параметров принтера (при этом, если соответствующий указатель NULL, то функции библиотеки автоматически запросят необходимую память под структуру и после завершения работы запишут туда выбранные пользователем параметры принтера);

hDevNames – идентификатор глобального блока памяти, который содержит структуру типа DEVNAMES, определяющую имя драйвера принтера, его имя и порт, к которому подключен принтер. Если соответствующий указатель NULL, то функции библиотеки автоматически запросят память под структуру и после завершения работы запишут туда

строки, соответствующие выбранному пользователем принтеру. Приложение использует их для создания контекста устройства;

hDC – идентификатор контекста устройства или информационного контекста в зависимости от установленного флага (значение PD_RETURNDC используется для печати, PD_RETURNIC используется для получения информации о принтере);

nFromPage хранит номер страницы, с которой должна начинаться печать;

nToPage хранит номер страницы, до которой должна выполняться печать;

nMinPage определяет минимальное количество страниц, которое можно задать при помощи элементов управления From и To;

nMaxPage определяет максимальное количество страниц, которое можно задать при помощи элементов управления From и To;

nCopies хранит число копий, которые нужно напечатать;

Flags хранит флаги инициализации, которые можно использовать при инициализации окна:

- PD_ALLPAGES включает переключатель All (Все);
- PD_COLLATE включает переключатель Collate (Разобрать по копиям);
- PD_DISABLEPRINTTOFILE блокирует флажок Print to file (Печать в файл);
- PD_HIDEPRINTTOFILE предписывает не отображать флажок Print to file (Печать в файл);
- PD_NOPAGENUMS блокирует переключатель Pages (Страницы) и связанный с ним элемент управления;
- PD_NOSELECTION блокирует переключатель Selection (Выделенный фрагмент);
- PD_NOWARNING отменяет вывод сообщения о том, что в системе не установлен принтер по умолчанию;
- PD_PAGENUMS устанавливает переключатель Pages во включенное состояние;
- PD_PRINTSETUP определяет, что на экран будет выведен диалог Print setup вместо Print;
- PD_PRINTTOFILE устанавливает при инициализации включенным флажок Print to file (при завершении работы диалогового окна установленное значение флага сообщает о том, что подсистеме печати необходимо запросить у пользователя имя выходного файла);
- PD_RETURNDC устанавливает, что в поле hDC будет записан идентификатор контекста принтера;
- PD_RETURNIC устанавливает, что в поле hDC будет записан идентификатор информационного контекста, который можно использовать для получения информации о принтере;
- PD_RETURNDEFAULT устанавливает, что в поля *hDevNames* и *hDevMode* записываются идентификаторы DEVNAMES и DEVMODE, которые заполнены параметрами принтера, установленного по умолчанию;
- PD_SELECTION устанавливает переключатель Selection включенным;
- PD_SHOWHELP устанавливает, что необходимо отображать кнопку Help (?);
- PD_USEDEVMODECOPIESANDCOLLATE блокирует поле Copies (Число копий), если драйвер принтера не поддерживает печать нескольких копий, и сбрасывает флажок Collate, если драйвер не поддерживает этот режим.

Базовые методы класса:

- конструктор

CPrintDialog::**CPrintDialog**(BOOL *bPrintSetupOnly*, DWORD *dwFlags* = PD_ALLPAGES | PD_USEDEVMODECOPIES | PD_NOPAGENUM | PD_HIDEPRINTTOFILE | PD_NOSELECTION, CWnd **pParentWnd* = NULL) ,

где параметр *bPrintSetupOnly* = FALSE задает режим Print (Печать), а *bPrintSetupOnly* = TRUE режим Print Setup (Настройка печати) (в первом случае отображается кнопка «Настройки печати» и к заданным значениям флагов автоматически добавляется значение PD_RETURNDC);

- метод, отвечающий за создание и использование текущего контекста устройства принтера на базе структур DEVMODE и DEVNAMES (при этом окно диалога не выводится)

HDC CPrintDialog::**CreatePrinterDC** ( ) ;

- метод, позволяющий получить дескриптор контекста устройства принтера (если при создании объекта конструктор вызывался с параметром *bPrintSetupOnly* = FALSE. После его использования контекст следует удалить, вызвав метод DeleteDC)

HDC CPrintDialog::**GetPrinterDC** ( ) ;

- метод, позволяющий получить параметры установленного по умолчанию принтера без открытия диалогового окна (указанные параметры настройки записываются в структуру m_pd)

BOOL CPrintDialog::**GetDefaults** ( ) ;

- метод, позволяющий получить имя текущего, выбранного принтера

CString CPrintDialog::**GetDeviceName** ( ) ;

- метод, позволяющий получить имя драйвера текущего выбранного принтера

CString CPrintDialog::**GetDriverName** ( ) ;

- метод, позволяющий получить имя порта текущего выбранного принтера

CString CPrintDialog::**GetPortName** ( ) ;

- метод, позволяющий получить указатель на структуру типа DEVMODE, которая содержит информацию о конфигурации и параметрах принтера

LPDEVMODE CPrintDialog::**GetDevMode** ( ) ;

- метод, позволяющий получить установленное число копий, которые нужно напечатать

int CPrintDialog::**GetCopies** ( ) ;

- метод, позволяющий получить установленный номер первой печатаемой страницы

int CPrintDialog::**GetFromPage** ( ) ;

- метод, позволяющий получить установленный номер последней печатаемой страницы

int CPrintDialog::**GetToPage** ( ) ;

- метод, позволяющий получить информацию о том, будут ли печататься все страницы документа (значение TRUE)

BOOL CPrintDialog::**PrintAll** ( ) ;

- метод, позволяющий получить информацию о том, выбрал ли пользователь опцию для печати копий документа (значение TRUE)

BOOL CPrintDialog::**PrintCollate** ( ) ;

- метод, позволяющий получить информацию о том, выбрал ли пользователь опцию для печати опеределенных страниц документа (значение TRUE)

BOOL CPrintDialog::**PrintRange** ( ) ;

- метод, позволяющий получить информацию о том, выбрал ли пользователь опцию для печати только выделенного фрагмента документа (значение TRUE)

BOOL CPrintDialog::**PrintSelection** ( ) .

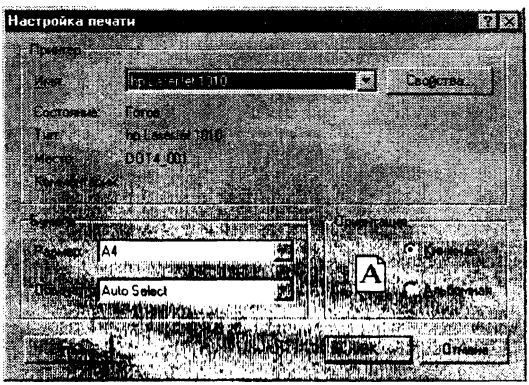


Рисунок 11 – Вид типового диалогового окна настройки печати, полученного при настройке

Настройка диалогового окна. Работать с диалогом можно, в простейшем случае, создав объект описанного класса CPrintDialog.

Ниже приведен фрагмент настройки диалогового окна настройки печати и печати полей структуры m_pd

```
#include "afxdlgs.h"
CPrintDialog PrintDlg(FALSE);
PrintDlg.m_pd.Flags |= PD_PRINTSETUP;
PrintDlg.m_pd.Flags |= PD_PRINTTOFILE;
PrintDlg.m_pd.Flags |= PD_USEDEVMODECOPIESANDCOLLATE;
PrintDlg.m_pd.Flags |= PD_ALLPAGES; .
```

Управление диалоговым окном. После создания объекта класса CPrintDialog и необязательной предварительной настройки диалог запускается путем вызова метода DoModal. Возвращаемое методом значение IDOK указывает на тот факт, что пользователь осуществил печать-настройку печати. Обработка результата осуществляется считыванием полей структуры m_pd типа PRINTDLG, где после выбора пользователя будет сохранена информация о выбранных настройках печати, а также при помощи вызова методов самого класса CPrintDialog.

ЛИТЕРАТУРА

1. Паппас, К. Visual C++. Руководство для профессионалов / К. Паппас, У. Мюррей; пер. с англ. – СПб: BHV – Санкт-Петербург, 1996.

2. Орлов, С.А. Технологии разработки программного обеспечения: учебник для вузов. – СПб.: Питер, 2004. – 527 с.

3. Паппас, К. Эффективная работа: Visual C++.NET / К. Паппас, У. Мюррей. – СПб.: Питер, 2002. – 816 с.

Дополнительная литература

4. Страуструп, Б. Язык программирования СИ++. – М.: Радио и связь, 1991. – 352 с.

5. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд.; пер. с англ. – М.: Издательство БИНОМ, СПб: Невский диалект, 1998 г. – 560 с.

6. Франка, П. C++: учебный курс. – СПб.: Питер, 2005. – 522 с.

7. Шилдт, Г. Самоучитель C++, 3-е изд. – СПб.: БХВ-Петербург, 2003. – 688 с.

8. Поляков, А.Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А.Ю. Поляков, В.А. Брусенцев. – СПб.: БХВ-Петербург, 2003. – 560 с.

9. Мюррей, У. Создание переносимых приложений для Windows / У. Мюррей, К. Паппас. – СПб.: BHV – Санкт-Петербург, 1997. – 816 с.

10. Финогенов, К.Г. Win32. Основы программирования. – М.: ДИАЛОГ-МИФИ, 2002. – 416 с.

11. Шилдт, Г. Справочник программиста по C/C++, 3-е изд. – М.: Изд. Дом Вильямс, 2003. – 432 с.

12. Шмуллер, Дж. Освой самостоятельно UML за 24 часа. – М.: Изд. Дом Вильямс, 2002. – 352 с.

13. Павловская, Т.А. C++. Объектно-ориентированное программирование: практикум / Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2004. – 265 с.

ОГЛАВЛЕНИЕ

Учебное издание

Авторы-составители:
Муравьев Геннадий Леонидович
Мухов Сергей Владимирович

ТИПОВЫЕ ДИАЛОГОВЫЕ ОКНА

Методическое пособие для студентов специальности 1 - 40 02 01, 1 - 40 03 01

Ответственный за выпуск: ***Муравьев Г.Л.***
Редактор: ***Строкач Т.В.***
Компьютерный набор: ***Муравьев Г.Л.***
Компьютерная вёрстка: ***Кармаш Е.Л.***
Корректор: ***Никитчик Е.В.***

Подписано в печать 25.10.2011 г. Формат 60x84 1/16. Гарнитура Arial Narrow.
Усл.-п. л. 2,79. Усл.-изд. л. 3,0. Тираж 50 экз. Заказ № 977.
Отпечатано на ризографе УО "Брестский государственный технический университет".
224017, Брест, ул. Московская, 267.