Министерство образования Республики Беларусь
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

методическое пособие

“ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ НА БАЗЕ БИБЛИОТЕКИ MFC”, часть 2

БРЕСТ 2009

УДК 681.3 (075.8)
ББК с57

Учреждение образования “Брестский государственный технический университет”
Кафедра интеллектуальных информационных технологий

Рекомендовано к изданию редакционно-издательским советом
Брестского государственного технического университета

Рецензент: доцент кафедры математического моделирования Брестского государственного университета им. А.С. Пушкина, к.т.н. Пролиско Е.Е.

Авторы-составители: Г.Л. Муравьев, доцент, к.т.н., В.Ю. Савицкий, доцент, к.т.н.,
В.И. Хвещук, доцент, к.т.н.,

ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ НА БАЗЕ БИБЛИОТЕКИ MFC. Г.Л. Муравьев, В.Ю. Савицкий, В.И. Хвещук. – Брест: БрГТУ, 2009. Часть 2. - 62 с.

Целью пособия является знакомство студентов с базовыми понятиями оконных приложений и каркасного программирования в системе Microsoft Visual Studio C++ .

УДК 681.3 (075.8)
ББК с57

© Учреждение образования
“Брестский государственный технический университет”, 2009

1. ВВЕДЕНИЕ

1.1. Графический интерфейс приложения

Графический интерфейс приложения предназначен обеспечить интерактивный доступ к приложению со стороны пользователей. 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. Также применяются стандартные диалоговые окна Windows (смотрите заголовочный файл commdlg.h). Это часто используемые окна типа “Сохранить как”, “Открыть”, “Печать”, “Параметры страницы” и др.

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

Графические объекты, формирующие визуальное представление интерфейса приложения, построены в соответствии со стандартом GDI. Указанные компоненты интерфейса могут быть описаны в отдельном файле. Они называются графическими ресурсами. С информационной точки зрения ресурсы - это описания графических объектов, достаточные для генерации 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.

Последние используются для визуального проектирования ресурсов. Это редакторы меню, диалоговых окон, инструменты для работы с пиктограммами, растровыми изображениями и т.п. Доступ к встроенным редакторам ресурсов осуществляется из пункта главного меню Resource и предполагает следующие действия.

1. Для создания нового файла ресурсов следует, открыв главное окно среды разработки Visual Studio, выполнить команду добавления в готовый проект соответствующего файла описания ресурсов: пункт меню Project, подпункт Add to Project, New, вкладка Files, тип файла - ResourceScript.

2. Для добавления нового ресурса с использованием соответствующего редактора ресурсов следует, открыв главное окно среды разработки Visual Studio, выбрать пункт меню Insert, подпункт Resource. На экран будет выведено окно с перечнем доступных ресурсов. При выборе типа ресурсов автоматически будет вызван соответствующий редактор ресурсов.

Как при разработке Windows-приложений, так и их графических пользовательских интерфейсов широко используются классы библиотеки MFC, что ускоряет и упрощает разработку. Фрагмент иерархии классов библиотеки MFC, используемых для создания приложений с графическим интерфейсом, представлен на рисунке ниже.

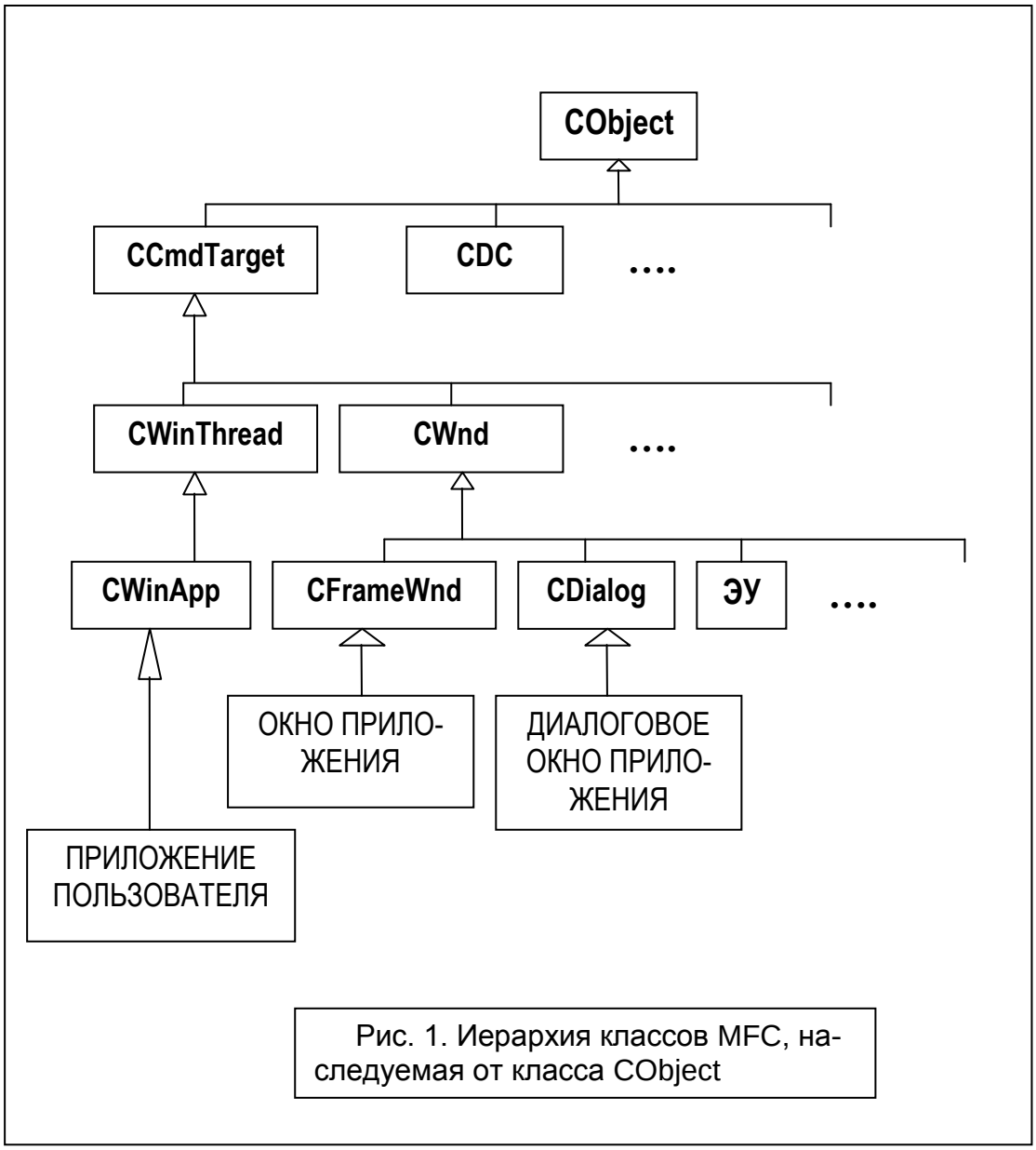


Рис. 1. Иерархия классов MFC, наследуемая от класса CObject

Основные типы ресурсов Visual Studio перечислены далее.

Это акселератор (Accelerator) для настройки комбинаций "горячих" клавиш.

Битовый образ (Bitmap) - цветной графический объект в виде растрового описания, отображающий окно, часть окна, объекты типа “стрелка”, “кисть”, “курсор” и т.п. используемый для быстрого вывода соответствующего изображения на экран. Пиктограмма (Icon) – графический объект.

Курсор (Cursor), в том числе текстовый для отображения позиции ввода и курсор - указатель “мыши”.

Диалоговые окна (Dialog) для описания соответствующих окон и расположенных на них элементах управления. Вызываемый при этом редактор диалоговых окон — это средство разработки графических объектов, позволяющее быстро создавать сложные диалоговые окна с возможностью комбинировать, изменять и настраивать в соответствии с собственными требованиями элементы окна, элементы управления.

Меню (Menu) - для создания иерархических пользовательских меню.

Таблица (String Table) - для хранения выводимой текстовой информации. Например, здесь могут храниться сообщения, отображаемые в строке состояния. Таблица упрощает изменение языка интерфейса программы, т.к. достаточно перевести на

другой язык строки таблицы, не затрагивая код программы. Панель инструментов (Toolbar). Информация о версии проекта (Version).

3. После того как ресурс создан в редакторе ресурсов или “вручную”, компилятор ресурсов считывает ASCII-файл описания ресурсов (*.rc) и создает для компоновщика приложения его двоичный аналог в виде res-файла.

Другой подход к реализации интерфейса – программное описание и генерация элементов интерфейса. Это описания (например, окна), представленные настроечными данными в специальных структурах данных, достаточных для генерации соответствующего интерфейсного объекта.

1.2. Классы MFC для работы с диалоговыми окнами

С типовым MFC-приложением связывается определяющий его на верхнем уровне объект, принадлежащий классу, производному от класса CWinApp. Фрагменты иерархии классов библиотеки MFC, используемых для управления диалоговыми окнами и элементами управления представлены на рисунках ниже. 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.

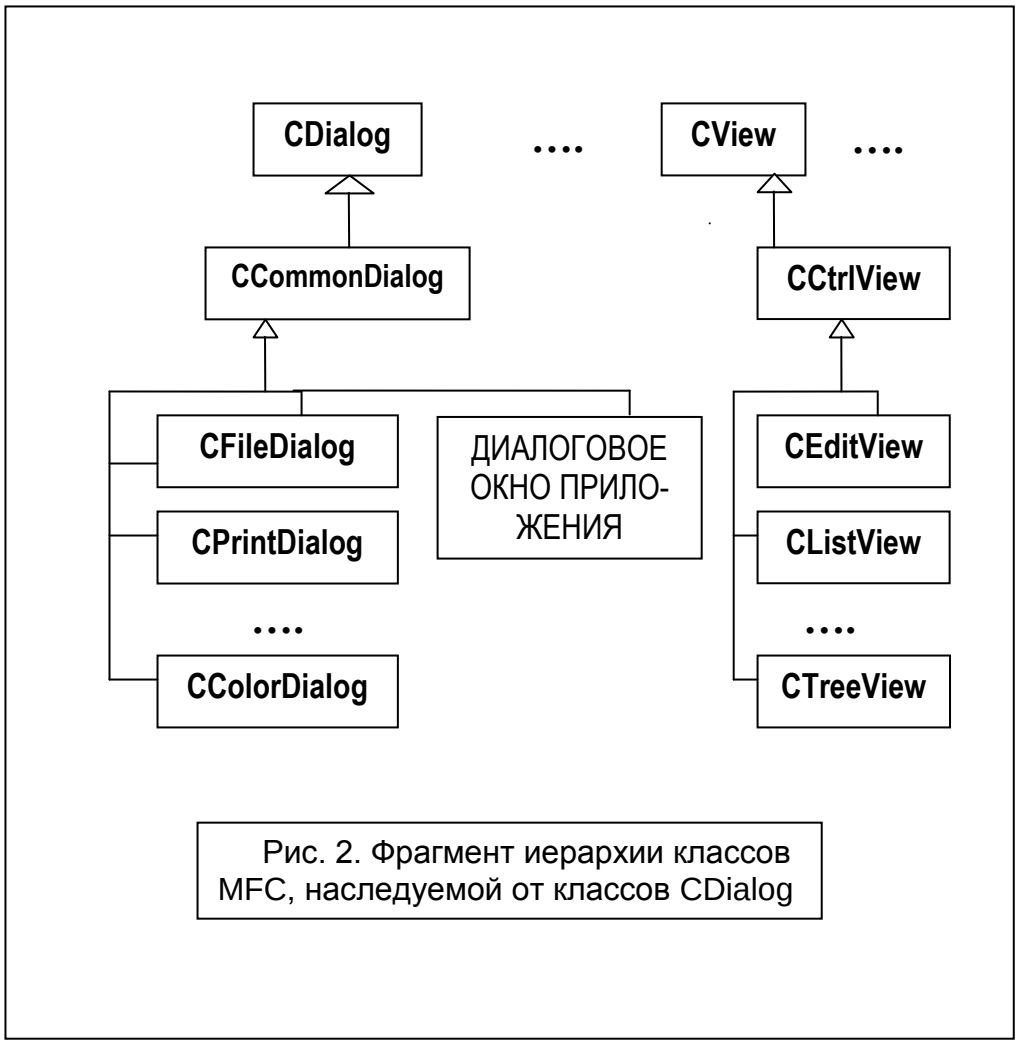


Рис. 2. Фрагмент иерархии классов MFC, наследуемой от классов CDialog

Каждый ЭУ, представляющий собой специфическое окно, производится от соответствующего класса семейства классов элементов управления.

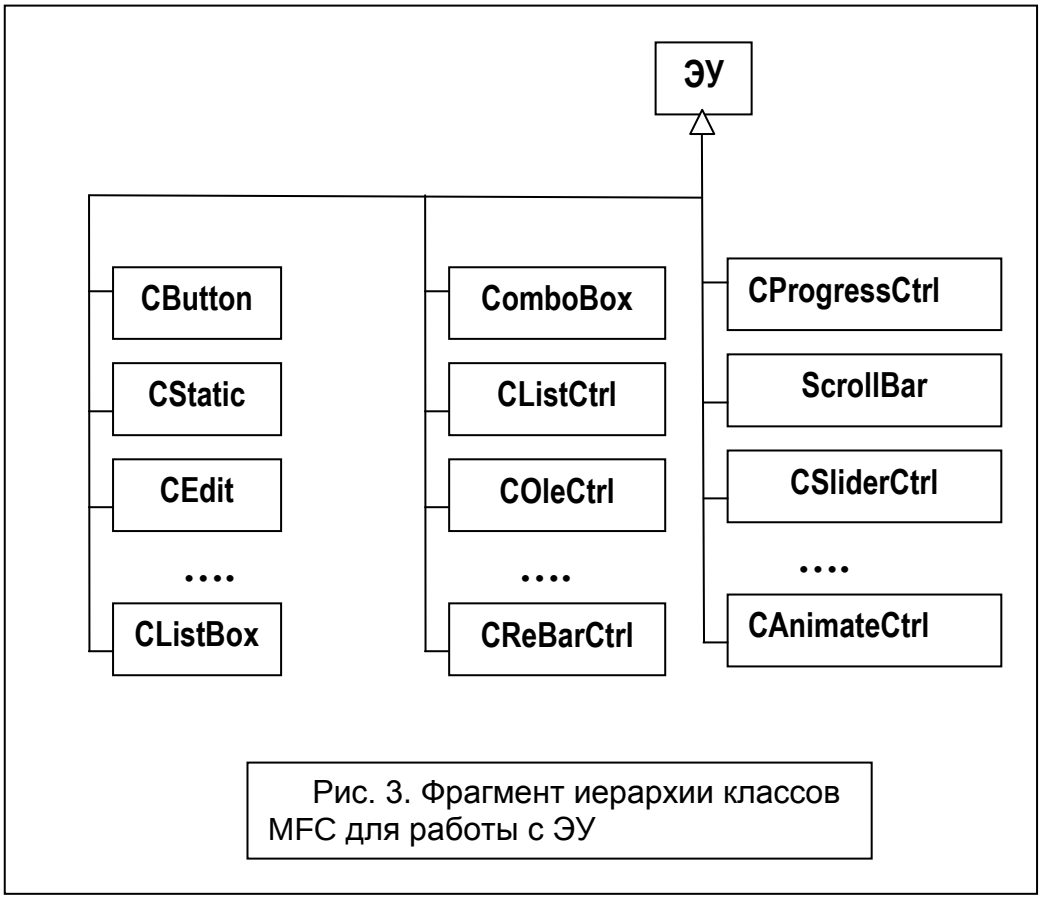


Рис. 3. Фрагмент иерархии классов MFC для работы с ЭУ

1.3. Диалоговые окна. Элементы управления диалоговых окон. Сообщения

Диалоговые окна. Основу графических интерфейсов образуют диалоговые окна, называемые так именно потому, что они позволяют пользователям “общаться” с приложением. Производится это общение посредством использования элементов управления, размещенных в диалоговых окнах. Каждое окно, созданное программно в памяти компьютера либо на базе шаблона – ресурса (визуального или текстового описания), кроме внешнего облика, который видит пользователь, является объектом соответствующего библиотечного класса. При этом объект инициализируется параметрами окна (координатами, размером, стилем и т.д.), заданными при его описании, создании. Методы соответствующего класса используются программистом для управления поведением окна.

Элементы управления. Описание элемента управления (ЭУ) включает следующий набор параметров: а) назначение, реализуемые функции, пользовательский интерфейс. Описание диаграммы прецедентов; б) типы (виды) ЭУ. Стили ЭУ. Режимы использования; в) представление ЭУ на языке описания ресурсов (ресурсное описание); г) базовый класс (классы), используемые методы и данные. Описание диаграммы классов; д) собственный класс, используемые методы и данные. Описание диаграммы классов; е) методы, команды, меняющие состояние ЭУ; ж) генерируемые, посылаемые сообщения (типы, причины возникновения – события, параметры, обработчики); з) создание и разрушение ЭУ; и) описание состояний, диаграммы состояний ЭУ; к) особенности программной реализации, управления ЭУ.

Видимые ЭУ сами, как правило, представляют собой окна, специализированные по назначению, поддерживаемым функциям. Например, кнопки предназначены для фиксации события – нажатие, окна редактирования предназначены как для ввода так и отображения информации и функционально представляют собой упрощенный вариант текстового редактора и т.д. Соответственно ЭУ являются объектами библиотечных классов либо пользовательских классов, производных от библиотечных.

Пользователь может выполнять действия над ЭУ (например, вводить число в окне редактирования, выбирать “мышью” кнопку, одинарным или двойным щелчком выбирать строку в списке и т.д.). Все это события (ввода), приводящие к посылке с помощью ОС соответствующих сообщений.

Приложения, обработчики сообщений приложений, могут, используя методы соответствующих классов, поддерживающих ЭУ, менять их состояния, тем самым выводить информацию пользователю. Например, добавлять новую строку в окно-список, выводить число в окне редактирования и т.д.

Справочная информация. Для получения справочной информации по классам MFC, информации о типах и особенностях сообщений, а также о прототипах их обработчиков можно использовать справочную систему MSDN.

Например, для получения информации о классе CDialog можно выделить текст – “Cdialog” в листинге приложения (либо набрать его в качестве шаблона поиска в справочнике) и нажать клавишу F1. Будет выведено окно с перечнем найденных разделов, содержащих запрошенную информацию. Например, для CDialog, как показано на рисунке ниже, это следующие разделы: CDialog - общие сведения о базовых классах, назначении, поведении, способах создания диалогового окна; раздел CDialog Class Members – общие сведения о методах класса; раздел CDialog::CDialog – общие сведения о конструкторах класса и т. д.

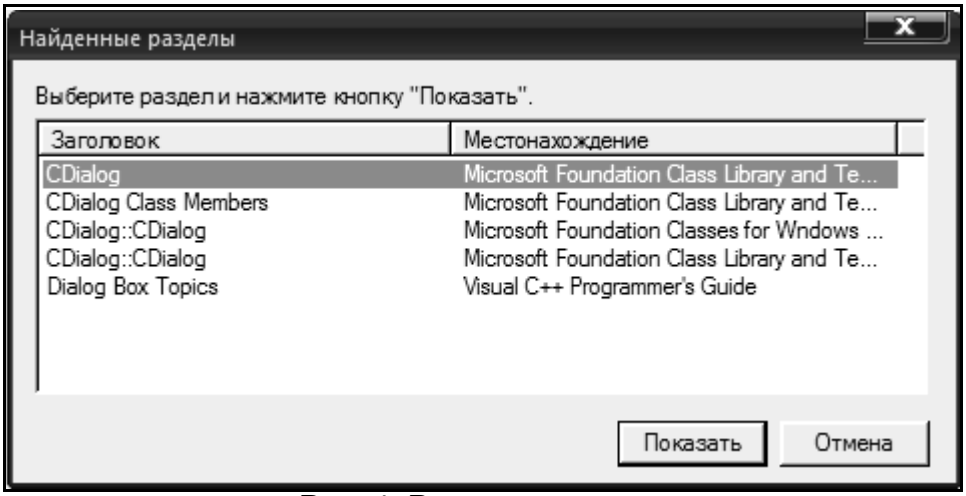


Рис. 4. Разделы справки

Для поиска информации можно использовать указатель MSDN. Например, можно набрать слово “CEdit” в качестве ключевого слова для поиска информации. Результаты поиска представлены на рисунке ниже.

Поиск информации об ЭУ (например, CEdit, CListBox и т.д.) может быть произведен также по слову “controls”. Среди найденных разделов следует выбрать – “Controls”. При этом в соответствующем подразделе можно найти информацию о сообщениях, посылаемых от элемента управления к операционной системе в результате воздействий пользователя и макрокомандах включения чувствительности к ним.

Может быть получена информация по конкретному сообщению - поиском по названию сообщения (например, WM_COMMAND или WM и т.д.).

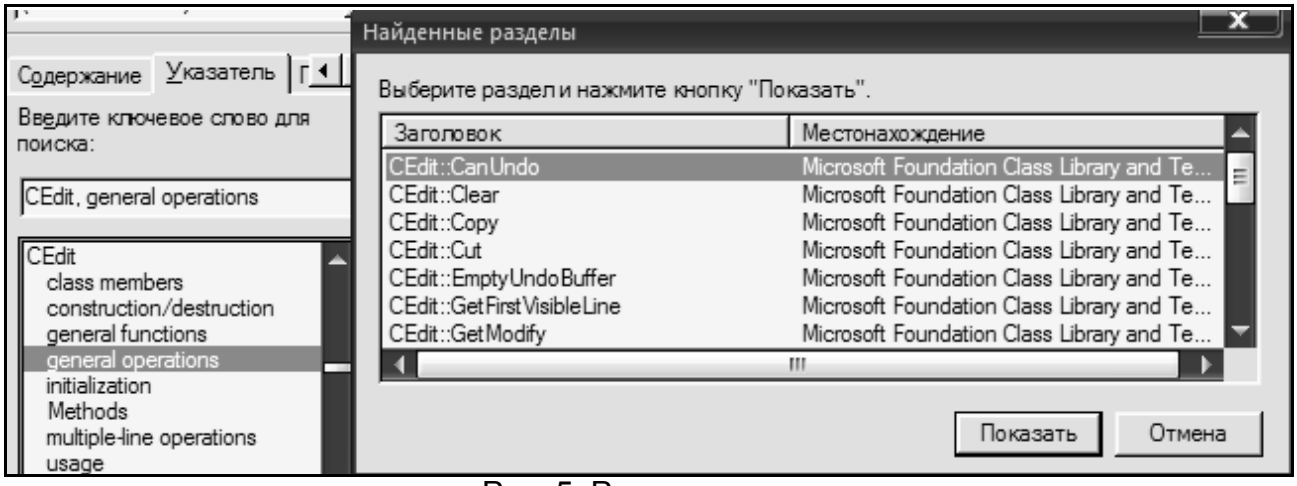


Рис. 5. Разделы справки

Сообщения. Необходимость ввода-вывода информации предполагает организацию в приложении канала связи ПРИЛОЖЕНИЕ-ОКНО. При этом приложение может как посылать сообщения, требуя определенной реакции от внешней среды (ОС, других приложений), так и принимать сообщения, реагируя соответствующим образом путем запуска функций-обработчиков сообщений. Для этого могут использоваться как готовые, типовые обработчики так и “пустые” обработчики сообщений со стандартными интерфейсами, функционирование которых доопределяется программистом.

Во взаимодействии пользователя с приложением участвуют: 1) сами пользователи, которые воздействуют на элементы управления, совершают события и тем самым инициируют сообщения; 2) элементы управления, идентифицируемые, например, дескриптором ресурса, указателем соответствующего объекта, которые имеют графический интерфейс, воспринимающий действия пользователя. ЭУ характеризуются набором посылаемых сообщений; 3) окно-родитетель, подложка, где располагаются ЭУ и куда направляются сообщения на обработку. Окна должны быть чувствительно к сообщениям и должны иметь соответствующие обработчики.

При работе с ЭУ следует учитывать: 1) сообщения, генерируемые самим ЭУ, которые посылаются в ОС через параметры сообщения WM_COMMAND; 2) сообщения, адресуемые ЭУ от функций приложения, которые посылаются принудительно, например, с помощью функции SendMessage; 3) сообщения ОС. Например, WM_CUT и др. Они могут посылаться функцией SendMessage.

1. Сообщения первой группы (notification message) вызываются пользователем, который выполнил действия, могущие привести к изменению состояния ЭУ (например, текста в окне редактирования). Сообщение (например, EN_CHANGE для списка) генерируется ЭУ и посылается в ОС. При этом родительское окно получает его (код и параметры сообщения) через сообщение WM_COMMAND. Например, в обработчик поступает следующая информация

LRESULT CALLBACK *Обработчик* **(**
HWND *ДескрипторОкнаРодителя***,**
UINT *КодПолученногоСообщения***,**
WPARAM *АтрибутыПолученногоСообщения***,**
LPARAM *ДескрипторОкнаИсточникаСообщения*

); .

Здесь *КодПолученногоСообщения* - WM_COMMAND, младшее слово параметра *АтрибутыПолученногоСообщения* содержит идентификатор окна, а старшее слово специфицирует сообщение (например, как EN_CHANGE).

2. Сообщения второй группы позволяют программно влиять на состояние ЭУ, используя его методы. Для этого применяется функция SendMessage. Например, программист может послать сообщение EM_GETLINE окну редактирования для копирования указанной строки текста из этого окна в буфер

SendMessage (
(HWND) *ДескрипторОкнаНазначения***,**
*EM_GETLINE***,**
(WPARAM) *НомерЧитаемойСтроки***,**
(LPARAM) *УказательПриемникаСтроки* // тип LPCTSTR
); .

Для того, чтобы приложение реагировало на сообщение о происшедшем событии, необходимо: 1. Включить чувствительность приложения к сообщению, связав источник сообщения и функцию-обработчик сообщения. Указанное выполняется макрокомандой включения вида ON_*Сообщение* (*ДескрипторИсточникаСообщения*, *ИмяОбработчика*). Макрокоманда включается в карту сообщений того окна, которое содержит обработчик для его обслуживания. 2. Создать обработчик сообщения с прототипом afx_msg void *ИмяОбработчика*( ). Большинство обработчиков имеют стандартные прототипы, описанные, например, в классе CWnd.

Например, нажатие кнопки ОК (с дескриптором IDOK) диалогового окна (класс MY_DIALOG) приводит к посылке сообщения типа WM_COMMAND. В качестве обработчика может использоваться функция вида afx_msg void *MY_DIALOG*::*OnButtonOK*( ) {…} . А макрокоманда включения выглядит как ON_COMMAND(*IDOK*, *OnButtonOK*).

2. ИСПОЛЬЗОВАНИЕ ДИАЛОГОВЫХ ОКОН

2.1. Общие сведения о классе CWnd

Общее описание, назначение. Класс CWnd обеспечивает основу функциональности специфических окон всех других, производных от него классов библиотеки MFC. Это CFrameWnd, CDialog, CView, CMDIFrameWnd и др. Как правило, эти классы в свою очередь используются программистами для наследования и описания производных классов, необходимых для создания пользовательских окон. Некоторые из этих классов, например, классы ЭУ (CButton, CEdit и др.) могут использоваться непосредственно для получения объектов или для наследования. CWnd-объект отличается от Windows-окон, хотя и тесно связан с ними. Подключается макрокомандой

#include <afxwin.h> .

Методы. Класс включает несколько десятков типов методов. В том числе:

1. Конструктор CWnd() для создания CWnd-объекта и деструктор DestroyWindow(), который шлет окну сообщение WM_DESTROY о том, что оно разрушается, сразу же после стирания окна с экрана. Однако сам CWnd-объект не разрушает. Посланное сообщение

перехватывается обработчиком afx_msg void CWnd::OnDestroy(), что позволяет осуществить необходимые действия.

2. Методы инициализации Create(), CreateEx() и др. Так Create() создает дочернее Windows-окно и прикрепляет его к CWnd-объекту.

3. Методы управления состоянием окна типа GetFocus(), SetFocus(), IsWindowEnabled( ) и др.

4. Методы управления размером и положением окна.

5. Методы управления доступом к окну, включая FindWindow(), методы получения указателей на заданные окна, ЭУ, например CWnd* CWnd::GetDlgItem(int nID) const; void CWnd::GetDlgItem(int nID, HWND* phWnd) const.

6. Методы рисования-обновления содержимого окна, в том числе BeginPaint(); EndPaint(); метод void UpdateWindow(), вызывающий перерисовку окна посылкой сообщения WM_PAINT; BOOL ShowWindow(int nCmdShow); различные вариации методов void Invalidate(BOOL bErase = TRUE) и void ValidateRect(LPCRECT lpRect).

7. Методы работы с текстом окна: void SetWindowText(LPCTSTR lpszString), который вызывает сообщение WM_SETTEXT; void GetWindowText(CString& rString) const, который вызывает сообщение WM_GETTEXT.

8. Методы работы с диалоговыми окнами: методы чтения и установки текста в окне (заголовка окна или содержимого окна) типа GetDlgItemText(), SetDlgItemText(), который вызывает сообщение WM_SETTEXT и др.; метод посылки сообщений ЭУ LRESULT SendDlgItemMessage(int ДескрипторЭУ, UINT Сообщение, WPARAM wParam = 0, LPARAM lParam = 0).

9. Методы посылки сообщений: SendMessage() посылает сообщение CWnd-объекту и не возвращает управление, пока сообщение не будет обработано; PostMessage() размещает сообщение в очереди сообщений приложения и возвращает управление, не ожидая, когда сообщение будет обработано.

10. Большая группа методов – подменяемые обработчики сообщений. Это обработчики системных сообщений; обработчики универсальных событий типа OnCommand(), OnClose(), OnMove(), OnSize() и др.; обработчики сообщений ввода типа OnChar(), OnLButtonDown(), OnTimer() и др.

Посылаемые сообщения. Сообщения Windows: это сообщения, начинающиеся с префикса WM_, за исключением WM_COMMAND. Многие из них имеют параметры, определяющие режим обработки сообщения. Сообщения обрабатываются окнами.

Это извещающие сообщения ЭУ: WM_COMMAND сообщения от ЭУ, от дочерних окон своим родительским окнам. Они несут в своих параметрах код события, учитываемый при обработке сообщения.

Командные сообщения: это WM_COMMAND сообщения от таких интерфейсных объектов как меню, панелей инструментов, клавиш. Система обрабатывает команды поразному в зависимости от сообщения.

Методы для обеспечения интерактивности приложений. Это методы, которые обеспечивают:

1. Получение доступа к ЭУ - метод **ПОЛУЧИТЬ_ЭУ** (ДескрипторЭУ, Указатель) или метод **УКАЗАТЕЛЬ** (ДескрипторЭУ)

void CWnd::GetDlgItem (int *ДескрипторЭУ***, HWND*** *Указатель***) const;**

CWnd* CWnd::GetDlgItem (int *ДескрипторЭУ* **) const; .**

2. Вывод данных в окно: а) метод **УСТАНОВИТЬ_ТЕКСТ_В_ОКНЕ** (*СтрокаВывода*) устанавливает заголовок окна заданной строкой, а если окно – ЭУ, то устанавливает текст внутри окна и вызывает посылку сообщения WM_SETTEXT

void CWnd::SetWindowText (LPCTSTR *СтрокаВывода***)** ;

б) методы

CWnd::SetDlgItemText (…),
CWnd::SetDlgItemInt (…) .

3. Ввод данных из окна: а) методами **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ОКНА** (*СтрокаВвода*) или **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ОКНА** (*СтрокаВвода*), которые копируют заголовок окна (класс CWnd) в заданную строку, а если окно – ЭУ, то копируется содержимое окна и посылается сообщение WM_GETTEXT

void CWnd::GetWindowText (CString& *СтрокаВвода***) const;**

int CWnd::GetWindowText (LPCTSTR *СтрокаВвода***, int** *ЧислоЧитаемыхСимволов* **) const;** ;

б) методы для получения текста названия (заголовка title) или текста, ассоциируемого с ЭУ диалогового окна. Они копируют текст в заданную строку. Это **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ЭУ** (*ДескрипторЭУ*, *БуферПриемникСтроки*, *ЧислоЧитаемыхСимволов*) или **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ЭУ** (*ДескрипторЭУ*, *БуферПриемникСтроки*)

int CWnd::GetDlgItemText(int *ДескрипторЭУ***, LPTSTR** *БуферПриемникСтроки***, int** *ЧислоЧитаемыхСимволов* **) const;**

int CWnd::GetDlgItemText(int *ДескрипторЭУ***, CString&** *БуферПриемникСтроки***) const;** ;

в) для ввода данных целого типа может использоваться метод **ВВЕСТИ_ЦЕЛОЕ_ЧИСЛО_ИЗ_ЭУ** (*ДескрипторЭУ*, = NULL, *НаличиеЗнака*)

UINT CWnd::GetDlgItemInt (int *ДескрипторЭУ***, BOOL*** *lpTrans* **= NULL, BOOL** *НаличиеЗнака* **= TRUE) const;**

Если признак *НаличиеЗнака* = 1, то результат ввода преобразуется в целое со знаком. А если 0, то результат ввода преобразуется в целое без знака.

2.2. Общие сведения о классе CDialog

Общее описание, назначение. Диалоговые окна функционально базируются на классе CDialog, производном от CWnd.

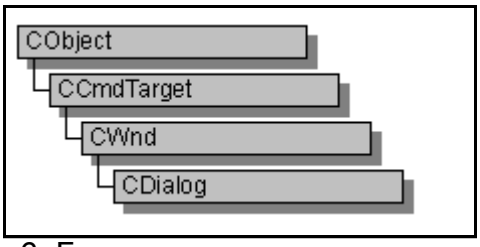


Рис. 6. Базовые и производные классы

По поведению бывают двух типов: модальные, которые перехватывают фокус и требуют завершения для дальнейшей работы приложения, и немодальные, позволяющие приложению работать без закрытия самого окна.

Диалоговое окно получает сообщения от Windows, включая сообщения от элементов управления (ЭУ) диалогового окна. Таким образом, пользователи осуществляют интерактивное взаимодействие с приложением.

Подключается #include <afxwin.h> .

По функциональному назначению диалоговые окна бывают пользовательские и стандартные, производные от класса CDialog и представленные ниже.

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

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

Члены класса. Это 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(). Диаграмма класса с учетом основных открытых членов представлена на рисунке ниже.

Создание и удаление диалоговых окон. Объект CDialog есть комбинация шаблона окна и CDialog-производного класса. Шаблон может быть выполнен а) в виде ресурса - диалоговое окно визуально в редакторе ресурсов; б) в виде ресурса - диалоговое окно текстовым описанием; в) шаблон может быть создан программно в памяти.

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

Ниже дается сводка наиболее часто используемых для этого методов.

1. Конструкторы CDialog-объектов

CDialog(LPCTSTR *ПоименованныйДескрипторОкна*, **CWnd*** *ОкноРодитель* **= NULL);**

CDialog(UINT *ДескрипторОкна*, **CWnd*** *ОкноРодитель* **= NULL);**

CDialog( );

Здесь *ОкноРодитель* указывает на родительское окно, к которому относится данное. Если NULL, то это главное окно приложения.

2. Метод

virtual int DoModal ();

вызывает модальное диалоговое окно. Метод

void EndDialog(int *ПризнакЗавершения* **);**

закрывает диалоговое окно и возвращает *ПризнакЗавершения* в точку вызова окна по DoModal ().

3. Метод инициализации окна, который требует подмены (overriding) и вызывается сообщением WM_INITDIALOG, которое посылается окну во время вызова Create, CreateIndirect или DoModal немедленно до вывода окна на экран

virtual BOOL OnInitDialog( ); .

CDialog
CDialog (LPCTSTR lpszDialName, CWnd* Owner = NULL); CDialog (UINT nID, CWnd* Owner = NULL); CDialog ( ); virtual int DoModal ( ); void EndDialog (int Status); virtual BOOL OnInitDialog ( ); virtual void OnCancel ( ); virtual void OnOK ( ); void CDialog::GotoDlgCtrl (CWnd* УказательЭУ_ПолучателяФокуса); CWnd* CWnd::GetDlgItem (int ДескрипторЭУ) const; void CWnd::GetDlgItem (int ДескрипторЭУ, HWND* Указатель) const; void CWnd::SetWindowText (LPCTSTR СтрокаВывода) void CWnd::GetWindowText (CString& СтрокаВвода) const; int CWnd::GetDlgItemText(int ДескрипторЭУ, LPTSTR БуферПриемникСтроки, int ЧислоЧитаемыхСимволов) const; CWnd::DestroyWindow ( );
Класс MFC для создания производных классов пользовательских диалоговых объектов

Рис. 7. Диаграмма класса CDialog

При подмене первым вызывается метод CDialog::OnInitDialog(), а затем программируются действия по специфичной инициализации окна. Примерный тест представлен ниже:

```
BOOL МОЙ_ДИАЛОГ::OnInitDialog()
{
  CDialog::OnInitDialog();
  ……..
  // Действия по инициализации
  return TRUE;
}  .
```

4. Стандартный обработчик закрытия окна (с кодом возврата DoModal - IDOK)

virtual void OnOK(); ,

и обработчик закрытия окна (с кодом возврата DoModal – IDCANCEL)

virtual void OnCancel(); .

Пользовательский класс может быть создан: а) с помощью мастера ClassWizard, который позволяет просматривать списки сообщений, генерируемых ЭУ окна, выбирать те, которые представляют интерес (при этом ClassWizard автоматически генерирует макрокоманды для карты сообщений окна и прототипы обработчиков, а программист дописывает реализации обработчиков), добавлять члены-данные для поддержки ЭУ; б) вручную.

Для конструирования модального окна на базе ресурса (шаблона) можно использовать любой public конструктор, а затем активизировать его DoModal(). Соответственно для создания модального окна надо: 1) создать пользовательский класс, производный от CDialog; 2) создать, используя конструктор, объект; 3) активизировать окно с ЭУ методом DoModal.

Для конструирования немодального окна надо использовать protected конструктор. Т.е. надо создать производный пользовательский класс диалогового окна, перегрузить конструктор. Далее создать объект, вызвав конструктор, создать на базе ресурса диалоговое окно методом Create. Соответственно для создания немодального окна надо: 1) создать пользовательский класс, производный от CDialog; 2) в его конструкторе вызвать Create; 3) создать окно, используя конструктор.

Если шаблон окна был создан в памяти, то используется структура данных типа DLGTEMPLATE. Она определяет координаты x, y, ширину и высоту окна cx, cy, стиль окна style (как комбинацию стилей окон и диалоговых окон), расширенный стиль dwExtendedStyle, число ЭУ в составе диалогового окна cdit и, следовательно, число структур типа DLGITEMTEMPLATE для их описания:

```
typedef struct
{
 DWORD style;
 DWORD dwExtendedStyle;
 WORD  cdit;
```

short *x*;
short *y*;
short *cx*;
short *cy*;
} DLGTEMPLATE, *LPDLGTEMPLATE; .

Здесь после создания CDialog-объекта используется метод CreateIndirect для создания немодального окна или InitModalIndirect и DoModal для создания модального окна.

Для немодальных окон следует также перегрузить обработчик OnCancel, вызвав из него DestroyWindow. Нельзя использовать CDialog::OnCancel без перегрузки, так там вызывается метод EndDialog, который делает окно невидимым, но не разрущает его.

Обрабатываемые сообщения. Это, в первую очередь, сообщения, обрабатываемые в объектах CWnd. Информацию по сообщениям можно получить в MDDN, как показано на рисунке ниже.

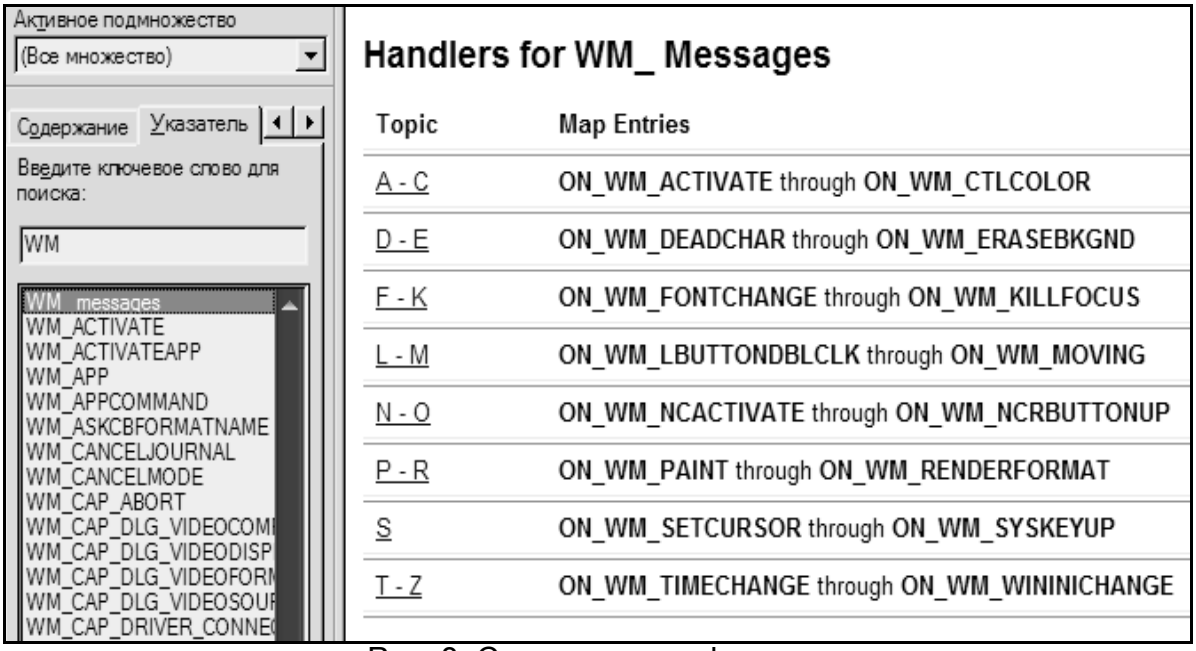


Рис. 8. Справочная информация

Специфические сообщения объектов CDialog: а) сообщение WM_INITDIALOG, которое посылается от ОС окну во время вызова методов Create, CreateIndirect или DoModal немедленно до вывода окна на экран. Прототип обработчика

virtual BOOL CDialog::OnInitDialog();

б) сообщение от кнопок типа WM_COMMAND. Прототипы обработчиков

virtual void OnOK();
virtual void OnCancel(); .

2.3. Общие сведения об использовании диалоговых окон и ЭУ для ввода-вывода данных

Общая технология организации интерактивных графических интерфейсов на базе диалоговых окон и ЭУ предполагает набор следующих типовых действий.

А. РАЗРАБОТКА ОКОННОГО ИНТЕРФЕЙСА и в том числе:

а) проектирование оконного интерфейса, в т.ч. системы окон, состава их элементов (включая элементы управления – ЭУ), порядка их взаимодействия и использования;

б) для каждого диалогового окна проектирование элементов управления (например, кнопок, окон редактирования и т.д.), определение их вида, размещения, возлагаемых функций;

в) для каждого ЭУ добавление, например, в редакторе ресурсов, к шаблону (ресурсу) ОКНО_ПОДЛОЖКИ (основному или диалоговому окну), настройка свойств, получение идентификатора (дескриптора ID);

г) для каждого ЭУ внесение изменений в класс ОКНО_ПОДЛОЖКИ – например, описание прототипов функций-обработчиков сообщений;

д) внесение изменений в функции инициализации класса ОКНО_ПОДЛОЖКИ, например, при необходимости конкретной инициализации содержимого и свойств ЭУ в момент запуска экземпляра класса ОКНО_ПОДЛОЖКИ;

е) для каждого ЭУ описание функциональности - алгоритмов использования ЭУ для ввода-вывода информации в соответствующих функциях-обработчиках сообщений и т.д.

Б. ОРГАНИЗАЦИЯ ОБРАБОТКИ СООБЩЕНИЙ, посылаемых окну (здесь объекту класса ОКНО_ПОДЛОЖКИ - CDialog). Предполагает:

а) включение чувствительности окна (объекта соответствующего класса) к сообщениям с помощью макрокоманд в карте сообщений окна

ON_*СООБЩЕНИЕ* **(***ID_ЭУ_ИсточникаСообщения*,
ИмяОбработчикаОкнаПриемникаСообщения **)** ;

б) описание обработчиков сообщений с прототипами

afx_msg void *ИмяОбработчикаОкнаПриемникаСообщения* **();** .

В. РЕАЛИЗАЦИЯ ВВОДА-ВЫВОДА. Специфика выполнения – алгоритмы реализации ввода-вывода данных существенно зависит от особенностей функционирования ЭУ разных типов. Тем не менее, предполагает выполнение некоторых типовых действий.

1. ПОЛУЧЕНИЕ ДОСТУПА К ЭУ. Выполняется, например, путем получения указателя на ЭУ (например, окно редактирования класса CEdit или на дочернее окно диалогового окна или окна) с использованием метода класса CWnd **ПОЛУЧИТЬ_ЭУ** (ДескрипторЭУ, Указатель)

void CWnd::GetDlgItem (int *ДескрипторЭУ***, HWND*** *Указатель***) const;**

или метода класса CWnd **УКАЗАТЕЛЬ** (ДескрипторЭУ)

CWnd* CWnd::GetDlgItem (int *ДескрипторЭУ* **) const; .**

Особенность метода – тип указателя следует привести к типу соответствующего ЭУ; сам указатель временный и не может быть сохранен. При отрицательном результате указатель равен NULL.

ПРИМЕР использования для ЭУ – окно редактирования:

```
CEdit*   УказательЭУ;
УказательЭУ = ( CEdit* ) GetDlgItem ( ДескрипторЭУ );
GotoDlgCtrl ( УказательЭУ );   .
```

2. ПЕРЕДАЧА ФОКУСА ЭУ с использованием метода класса CDialog **ПЕРЕЙТИ_К_ЭУ** (*УказательЭУ_ПолучателяФокуса*)

void CDialog::GotoDlgCtrl (CWnd* *УказательЭУ_ПолучателяФокуса* **)**; .

3. ПРЕОБРАЗОВАНИЕ ДАННЫХ В СТРОКОВЫЙ ТИП. Вывод данных, как правило, производится в строковом формате, что может потребовать их предварительного переформатирования. Так для преобразования в строку данных целого типа можно использовать функцию

wsprintf (*СтроковоеПредставлениеЧисла***, "%d",** *ИсходноеЧисло***)**; .

Для преобразования вещественного значения можно использовать аналогичную функцию sprintf() либо функцию **ПРЕОБРАЗОВАТЬ_В_СТРОКУ** (*ИсходноеЧисло*, *ДлинаСтроки*, *СтроковоеПредставлениеЧисла*)

char* _gcvt (double, int, char) .

4. ВЫВОД ДАННЫХ:

а) ЭУ в значительной мере используют функциональность базового класса CWnd. Для вывода данных в окно (например, при работе с однострочным окном редактирования, а также для вывода полного содержимого многострочного окна) можно использовать метод **УСТАНОВИТЬ_ТЕКСТ_В_ОКНЕ** (*СтрокаВывода*)

void CWnd::SetWindowText (LPCTSTR *СтрокаВывода***)** .

Метод устанавливает заголовок окна заданной строкой, а если окно – ЭУ, то устанавливает текст внутри окна. СтрокаВывода – Cstring-строка или С-строка. Метод вызывает посылку этому окну сообщения WM_SETTEXT.

ПРИМЕР вывода информации (числовых и не числовых данных) в виде строк в стандартном элементе управления – в окне редактирования (класс CEdit) приведен ниже. Пусть дескриптор окна редактирования - IDC_EDIT2. Тогда вывод данного в виде строки

```
CEdit  *pEditBox = ( CEdit * )  CDialog:: GetDlgItem ( IDC_EDIT2 );
pEditBox -> SetWindowText ( OutputStr );  .
```

Вывод данного целого типа

```
wsprintf ( OutputStr, "%d", IntNumber );
CEdit  *pEditBox = ( CEdit * ) CDialog:: GetDlgItem ( IDC_EDIT2 );
pEditBox -> SetWindowText( OutputStr );
```

Вывод данного вещественного типа (фрагмент текста программы)

```
#include <stdio.h>
……
float FloatNumber = 12.345;
char OutputStr [100];
CEdit  *pEditBox = ( CEdit * )  CDialog:: GetDlgItem ( IDC_EDIT2 );
pEditBox -> SetWindowText ( OutputStr );  .
……..
sprintf ( OutputStr, "%f" , FloatNumber );
pEditBox -> SetWindowText ( OutputStr );  .
……
_gcvt (FloatNumber, strlen(OutputStr), OutputStr );
pEditBox -> SetWindowText ( OutputStr );
……
```

б) кроме этого для вывода могут использоваться методы

CWnd::SetDlgItemText (…),
CWnd::SetDlgItemInt (…) .

в) для работы с ЭУ конкретных типов можно использовать специфические методы. Например, для вывода в многострочное окно редактирования следует применять методы CEdit типа:

CEdit::SetLine(…) ,
CEdit::ReplaceSel(…) .

5. ПРЕОБРАЗОВАНИЕ ДАННЫХ СТРОКОВОГО ТИПА. Ввод данных, как правило, производится в строковом формате, что может потребовать их последующего переформатирования. Так для получения из строки данного вещественного типа можно использовать функцию

Число = **atof** (*СтроковоеПредставлениеЧисла*).

6. ВВОД ДАННЫХ.

а) для ввода данных из окна (например, при работе с однострочным окном редактирования, а также для считывания полного содержимого многострочного окна) можно использовать метод CWnd **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ОКНА** (*СтрокаВвода*)

void CWnd::GetWindowText (CString& *СтрокаВвода***) const;**

или метод **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ОКНА** (*СтрокаВвода*)

int CWnd::GetWindowText (LPCTSTR *СтрокаВвода***, int** *ЧислоЧитаемыхСимволов* **) const;** .

Методы копируют заголовок окна (класс CWnd) в заданную строку, а если окно – ЭУ, то копируется содержимое окна. *СтрокаВывода* здесь Cstring-строка или С-строка. Метод вызывает посылку этому окну сообщения WM_GETTEXT.

ПРИМЕР (фрагмент) ввода данных строкового, целого и вещественного типов в окне редактирования с идентификатором IDC_EDIT1 представлен ниже.

```
char szMyString [80] = " ";
int IntNumber ;
float FloatNumber ;
……..
CEdit  *pEditBox = (CEdit *) CDialog:: GetDlgItem ( IDC_EDIT1 );
pEditBox -> GetWindowText  ( szMyString, sizeof  szMyString - 1);
……..
CEdit  *pEditBox = ( CEdit * ) CDialog:: GetDlgItem ( IDC_EDIT1 );
pEditBox -> GetWindowText  ( szMyString, sizeof szMyString - 1);
IntNumber = atof (szMyString);
……..
CEdit  *pEditBox = ( CEdit * ) CDialog:: GetDlgItem ( IDC_EDIT1 );
pEditBox -> GetWindowText  ( szMyString, sizeof  szMyString - 1);
FloatNumber = atof (szMyString);
……..
```

б) ниже представлены другие методы CWnd для получения текста названия (заголовка title) или текста, ассоциируемого с ЭУ в диалоговом окне. Они копируют текст в заданную строку **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ЭУ** (*ДескрипторЭУ*, *БуферПриемникСтроки*, *ЧислоЧитаемыхСимволов*)

int CWnd::GetDlgItemText(int *ДескрипторЭУ***, LPTSTR** *БуферПриемникСтроки***, int** *ЧислоЧитаемыхСимволов* **) const;**

или **ПОЛУЧИТЬ_ТЕКСТ_ИЗ_ЭУ** (*ДескрипторЭУ*, *БуферПриемникСтроки*)

int CWnd::GetDlgItemText(int *ДескрипторЭУ***, CString&** *БуферПриемникСтроки***) const;** .

ПРИМЕР использования

```
char MyString[20];
float MyFloat;
……..
GetDlgItemText ( IDC_EDIT1, MyString,15);
MyFloat = atof ( MyString );  .
```

в) для ввода данных целого типа может использоваться метод **ВВЕСТИ_ЦЕЛОЕ_ЧИСЛО_ИЗ_ЭУ** (*ДескрипторЭУ*, = NULL, *НаличиеЗнака*)

UINT CWnd::GetDlgItemInt (int *ДескрипторЭУ***, BOOL*** *lpTrans* **= NULL, BOOL** *НаличиеЗнака* **= TRUE) const;**

Если признак *НаличиеЗнака* = 1, то результат ввода преобразуется в целое со знаком. А если 0, то результат ввода преобразуется в целое без знака.

г) для работы с ЭУ конкретных типов можно использовать специфические методы. Например, для ввода из многострочного окна редактирования следует применять методы:

int CEdit::GetLine(int *НомерСтрокиОкнаРедактирования*, **LPTSTR** *СтрокаПриемник* **) const;**
int CEdit::GetLine(int *НомерСтрокиОкнаРедактирования*, **LPTSTR** *СтрокаПриемник*, **int** *МаксимальнаяДлинаСтрокиПриемника* **) const; .**

ПРИМЕР использования:

```
CEdit*      МоеОкно;
int         НомерСтроки;
int         ЧислоСтрок = МоеОкно -> GetLineCount ( );
CString     strText, strLine;

for (НомерСтроки = 0; НомерСтроки <  ЧислоСтрок;  НомерСтроки ++)
{
    МоеОкно -> GetLine(НомерСтроки, strText.GetBuffer ( МоеОкно -> LineLength ( НомерСтроки ) ) );
    strText.ReleaseBuffer ( );
    …
}  .
```

2.4. Общие сведения о классе CButton

Общее описание, назначение. Элемент управления типа “кнопка” (button) – небольшое прямоугольное (квадратное) производное окно. Кнопку, путем щелчка, можно нажать или отжать. При этом она, как правило, меняется внешне. Может использоваться автономно либо в составе группы кнопок. Подключается #include <afxwin.h> .

Функциональность кнопок поддерживается классом CButton, производным от CWnd.

Стили. Стиль кнопки зависит от параметров стиля, задаваемых, например, при инициализации соответствующего объекта класса CButton методом Create. Либо задается установкой флажков в окнах свойств ресурса в редакторе ресурсов. Разновидности кнопок (класс CButton): переключатели (check box - стиль BS_CHECKBOX), радио кнопки (radio button - стиль BS_RADIOBUTTON) командные кнопки (pushbutton - стиль BS_PUSHBUTTON). Еще один вид – графическая кнопка (класс CBitmapButton производный от CButton). Это кнопка, маркируемая изображением (типа bitmap) вместо текста. Может маркироваться разными изображениями для разных состояний кнопки: нажата, отжата, выбрана (focused), недоступна (disabled) и т.д. Ниже представлены основные стили:

- BS_PUSHBUTTON - командная кнопка, которая посылает сообщения типа WM_COMMAND своему окну-владельцу (owner window) каждый раз, когда пользователь выбирает кнопку;
- BS_CHECKBOX - переключатель. Это кнопка в виде небольшого прямоугольника с поясняющим текстом справа (по умолчанию) либо слева;
- BS_RADIOBUTTON - радио-кнопка. Это кнопка в виде небольшой окружности с поясняющим текстом справа (по умолчанию) либо слева. Используются, как правило, группами связанных и взаимоисключающих по выбору кнопок;

- BS_LEFTTEXT - в комбинации с переключателями или радио-кнопками позволяет выводить поясняющий текст слева от кнопки;
- BS_AUTOCHECKBOX - разновидность переключателя с отметкой, которая появляется в его поле при выборе пользователем;
- BS_AUTORADIOBUTTON - разновидность радио-кнопки, которая выделяется пользователем и автоматически снимает выделение со всех других кнопок этого стиля, объединенных в группу;
- BS_GROUPBOX - прямоугольник, в котором группируются другие кнопки;
- BS_DEFPUSHBUTTON - кнопка с темной рамкой по периметру. Пользователь может выбирать ее нажатием клавиши ENTER.

Члены класса. Конструктор CButton, метод инициализации Create, методы GetState, SetState для получения информации о текущем состоянии кнопки либо для программной установки состояния и др.

Создание и удаление. Кнопки могут создаваться на основе соответствующего оконного шаблона (dialog template) либо непосредственно программным путем. В первом случае кнопка создается визуально в редакторе ресурсов либо описывается в текстовом редакторе. В обоих случаях далее создается объект класса CButton соответствующим конструктором

CButton *МояКнопка*;

При этом при ресурсном описании кнопки ее параметры автоматически инициализируют создаваемый объект. При программном создании для этого используется метод Create. Он создает кнопку и “прикрепляет” ее к CButton-объекту. Прототип метода

BOOL CButton::Create (LPCTSTR *ИмяКнопки***, DWORD** *СтильКнопки***, const RECT&** *РазмерКоординаты***, CWnd*** *РодительскоеОкно***, UINT** *ДескрипторКнопки* **);**

Здесь возвращаемое значение – признак завершения (нормальное завершение – ненулевой код); *РодительскоеОкно,* как правило, класса CDialog. Используемые в методе стили: WS_CHILD (всегда), WS_VISIBLE (как правило), WS_DISABLED (иногда), WS_GROUP (для групп кнопок) и др. Стиль WS_VISIBLE определяет, что ОС Windows посылает кнопке все сообщения, требуемые для ее активизации, визуализации.

Пример использования метода

CButton *МояКомандная Кнопка*, *МояРадиоКнопка*;

МояКомандная Кнопка.**Create** ("Командная Кнопка"), WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, CRect(5,5,100,30), *Указатель*, ID_PushButton);

МояРадиоКнопка.**Create** (("РадиоКнопка"), WS_CHILD | WS_VISIBLE | BS_RADIOBUTTON, CRect(10,40,100,70), *Указатель*, ID_RadioButton); .

Кнопка (объект класса **CButton)**, созданная первым способом (на основе диалогового, оконного ресурса), разрушается автоматически, когда пользователь закрывает диалоговое окно. Если кнопка создавалась программно и к тому же в динамической памяти (heap) с помощью оператора new, то необходимо удалить объект оператором delete. Если же объект создан в стеке (stack) или был внедрен в родительский диалоговый объект, то он разрушается автоматически.

Можно создать кнопка независимо от диалогового или любого другого окна, например,

ДескрипторКнопки = CreateWindow(
"BUTTON", "OK", WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
10, 10, 100, 100,
ДескрипторОРодительскогоОкна, NULL,
(HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE), NULL); .

Посылаемые сообщения. Список сообщений: BM_CLICK, BM_GETCHECK, BM_GETIMAGE, BM_GETSTATE, BM_SETCHECK, BM_SETIMAGE, BM_SETSTATE, BM_SETSTYLE, BN_CLICKED, BN_DBLCLK, BN_DOUBLECLICKED (аналог BN_DBLCLK), BN_KILLFOCUS, BN_SETFOCUS, WM_CTLCOLORBTN и др. Список сообщений можно получить в MSDN (см. Buttons Overview, Buttons Messages и т.д.), а информацию о сообщении путем поиска, например, по образцу BN_CLICKED.

Сообщение BN_CLICKED посылается, когда пользователь “щелкает” кнопку. Родительское окно кнопки получает его (код сообщения) через сообщение WM_COMMAND. При этом в обработчик окна поступает следующая информация:

LRESULT CALLBACK *WindowProc*(
HWND *ДескрипторДиалоговогоОкнаРодителя*,
UINT *КодПолученногоСообщения*,
WPARAM *АтрибутыПолученногоСообщения*,
LPARAM *ДескрипторКнопкиИсточникаСообщения*
) .

Здесь *КодПолученногоСообщения* - WM_COMMAND; младшее слово параметра *АтрибутыПолученногоСообщения* содержит идентификатор кнопки, а старшее слово специфицирует сообщение как BN_CLICKED. Недоступная кнопка (стиль disabled) такое сообщение окну-родителю не посылает.

Сообщение BN_DOUBLECLICKED автоматически посылается, когда пользователь дважды “щелкает” кнопку, если включен стиль BS_USERBUTTON, BS_RADIOBUTTON, BS_OWNERDRAW. Другие кнопки посылают это сообщение только при включенном стиле BS_NOTIFY. При этом родительское окно кнопки получает его (код сообщения) через сообщение WM_COMMAND: здесь *КодПолученногоСообщения* - WM_COMMAND; младшее слово параметра *АтрибутыПолученногоСообщения* содержит идентификатор кнопки, а старшее слово специфицирует сообщение как BN_CLICKED. Недоступная кнопка (стиль disabled) такое сообщение окну-родителю не посылает.

Обработка сообщений. Необходимо связать ЭУ-источник сообщения и обработчик сообщения: описать обработчик сообщения; включить чувствительность родителя к сообщению – задать макрокоманду карты сообщений. Например,

ON_BN_CLICKED (ID_ЭУ_ИсточникаСообщения,
ИмяОбработчикаОкнаПриемникаСообщения).

2.5. Общие сведения о классе CEdit

Назначение, общее описание. Элемент управления типа “окно редактирования” (edit) – небольшое прямоугольное (квадратное) дочернее (child) окно, в котором пользователь может вводить текст с клавиатуры. Фокус передается окну “мышью”, клавишей Tab. Ра-

ботает как редактор текста, позволяя вводить, корректировать, копировать, вставлять, вырезать текст (взаимодействуя с clipboard). Подключается #include <afxwin.h> .

Функциональность кнопок поддерживается классом CEdit, производным от CWnd.

Стили. Основные разновидности окна редактирования (класс CEdit): однострочный редактор (по умолчанию); многострочный редактор (стиль ES_MULTILINE); окно только для отображения текста (стиль ES_READONLY); окно для ввода скрытой информации (стиль ES_PASSWORD). Режим использования окна зависит от параметров стиля, задаваемых, например, при инициализации соответствующего объекта класса CEdit методом Create либо задается установкой флажков в окнах свойств ресурса в редакторе ресурсов. В многострочном режиме работа окна зависит также от включения-выключения возможности горизонтального и вертикального скроллирования текста.

Члены класса. Это конструктор CEdit, метод инициализации Create. Методы для получения значений атрибутов окна редактирования в одно и многострочном вариантах использования. Например, GetSel для получения начальной и конечной позиций текущего выделения в окне

CEdit* *МоеОкно*;
...
DWORD *ПозицииВыделения* = *МоеОкно* -> GetSel (); .

Методы для установки значений атрибутов окна редактирования, работы с окном в одно и многострочном вариантах использования. Например, SetSel для выделения фрагмента текста по заданным начальной и конечной позициям выделения, например

МоеОкно -> SetSel (*НачальнаяПозиция*, *КонечнаяПозиция*); .

Методы Clear, Cut, Undo и др., поддерживающие работу с clipboard.

Методы для чтения-записи строк. Объекты класса CEdit в значительной мере используют функциональность базового класса CWnd. Так при работе с однострочным окном следует использовать методы (например, для установки текста в окне редактирования, а также для считывания ранее введенного текста):

CWnd::GetWindowText (…) ,
CWnd::SetWindowText (…) .

Они же могут применяться для вывода-чтения полного содержимого многострочного окна. Для работы с частями текста многострочного окна можно применять методы CEdit:

int CEdit::GetLine(int *НомерСтрокиОкнаРедактирования***, LPTSTR** *СтрокаПриемник* **) const;**
int CEdit::GetLine(int *НомерСтрокиОкнаРедактирования***, LPTSTR** *СтрокаПриемник***, int** *МаксимальнаяДлинаСтрокиПриемника* **) const;**
CEdit::ReplaceSel(…) .

Создание и удаление. Окна редактирования могут создаваться на основе соответствующего оконного шаблона (dialog template) либо непосредственно программным путем. В первом случае окно создается визуально в редакторе ресурсов либо описывается в текстовом редакторе. В обоих случаях далее создается объект класса CEdit соответствующим конструктором CEdit

CEdit *МоеОкноРедактирования*; .

При этом при ресурсном описании окна его параметры автоматически инициализируют создаваемый объект. При программном создании используется метод Create. Он создает окно и “прикрепляет” его к CEdit-объекту. Для этого следует вставить вызов Create в соответствующий конструктор. Прототип Create

BOOL CEdit::Create (DWORD *СтильОкна***, const RECT&** *РазмерКоординаты***, CWnd*** *РодительскоеОкно***, UINT** *ДескрипторОкна* **);** .

Здесь: возвращаемое значение – признак завершения (нормальному завершению соответствует ненулевой код); *РодительскоеОкно*, как правило, класса CDialog. Используемые в методе стили: WS_CHILD (всегда), WS_VISIBLE (как правило), WS_DISABLED (иногда), WS_GROUP (для групп ЭУ) и др. Стиль WS_VISIBLE определяет, что ОС Windows посылает окну все сообщения, требуемые для его активизации, визуализации.

Пример использования метода

CEdit *МоеОкно***;**
или
CEdit* *УказательМоеОкно* = **new CEdit;**
УказательМоеОкно -> **Create (** ES_MULTILINE | WS_CHILD | WS_VISIBLE | WS_TABSTOP | WS_BORDER, CRect (5, 5, 100, 300), this, 1); .

Окно редактирования (объект класса CEdit), созданное в составе диалогового, оконного ресурса, разрушается автоматически, когда пользователь закрывает диалоговое окно. Если окно создавалась программно и к тому же в динамической памяти (heap) с помощью оператора new, то необходимо удалить объект оператором delete. Если же объект создан в стеке (stack) или был внедрен в родительский диалоговый объект, то он разрушается автоматически.

Посылаемые сообщения. Список сообщений можно получить в MSDN (см. Edit Controls Overview, Edit ControlMessages и т.д.), а информацию о сообщении путем поиска, например, по образцу EM_GETLINE. Ниже дана общая характеристика групп сообщений.

1. Сообщения (notification message), генерируемые самим окном редактирования и посылаемые ОС через сообщение WM_COMMAND: типа EN_CHANGE, EN_KILLFOCUS, EN_SETFOCUS, EN_UPDATE, EN_VSCROLL и др. Например, сообщение EN_CHANGE генерируется и посылается окном, если пользователь выполнил действия, которые могут привести к событию “изменение текста в окне”. В отличие от EN_UPDATE, это сообщение посылается после того как система обновляет экран. При этом в обработчик окна поступает следующая информация

```
LRESULT CALLBACK WindowProc(
  HWND ДескрипторДиалоговогоОкнаРодителя,
  UINT КодПолученногоСообщения,
  WPARAM АтрибутыПолученногоСообщения,
  LPARAM ДескрипторОкошкаИсточникаСообщения
); .
```

Здесь *КодПолученногоСообщения* - WM_COMMAND; младшее слово параметра *АтрибутыПолученногоСообщения* содержит идентификатор окна, а старшее слово специфицирует сообщение как EN_CHANGE.

Сообщение EN_UPDATE посылается, когда окно нуждается в перерисовке. Например, посылается после форматирования текста, но до вывода его в окне. Это делает возможным изменить размеры окна при необходимости.

2. Сообщения, адресуемые окну редактирования от программ приложения. Они посылаются принудительно, например, с помощью SendMessage и позволяют программно влиять на состояние окна редактирования. Это сообщения типа EM_GETLINE, EM_SETSEL и др. Например, сообщение EM_GETLINE посылается для вызова копирования строки текста из окна в буфер. Вид функции представлен ниже

```
SendMessage(
  (HWND) ДескрипторОкнаНазначения,
  EM_GETLINE,
  (WPARAM) НомерЧитаемойСтроки,
  (LPARAM) УказательПриемникаСтроки          // line buffer (LPCTSTR)
);
```

В однострочном окне *НомерЧитаемойСтроки* игнорируется.

Сообщение EM_SETSEL посылается для выделения диапазона символов. Вид функции представлен ниже

```
SendMessage(
  (HWND) ДескрипторОкнаНазначения,
  EM_SETSEL,
  (WPARAM) НачальнаяПозицияВыделения,
  (LPARAM) КонечнаяПозицияВыделения
);   .
```

3. После инициализации окна редактирования с использованием метода Create ОС посылает окну редактирования сообщения типа WM_CREATE, WM_NCCREATE и др. Они воспринимаются обработчиками-методами класса CWnd типа OnCreate(), OnNcCreate() и др. При необходимости методы можно подменить в пользовательском классе окна редактирования, производном от CEdit. Например, OnCreate() можно подменить для начальной инициализации пользовательского класса.

Примеры макрокоманд включения представлены ниже

```
ON_EN_CHANGE(ID_ЭУ_ИсточникаСообщения,
                    ИмяОбработчикаОкнаПриемникаСообщения),
ON_EN_ UPDATE(ID_ЭУ_ИсточникаСообщения,
                    ИмяОбработчикаОкнаПриемникаСообщения).
```

2.6. Диалоговое окно в составе главного окна

ЗАДАНИЕ № 1. *Создать MFC-приложение на базе ТКП с оконным интерфейсом из главного окна и диалогового окна с двумя кнопками OK, CANCEL.*

При запуске приложения должно выводиться главное окно и на фоне главного окна приложения - диалоговое окно, которому и передается управление. Диалоговое окно

функционирует в модальном режиме и после нажатия любой из кнопок OK или CANCEL исчезает, возвращая управление главному окну. Диалоговое окно появляется каждый раз при событии "перерисовки" главного окна, т.е. при попытке изменения размеров главного окна.

Из описания задания следует:

1) начальная визуализация диалогового окна должна происходить автоматически при запуске приложения, как реакция приложения на сообщение WM_PAINT, посылаемое системой главному окну приложения при его запуске;

2) визуализация диалогового окна должна происходить автоматически каждый раз, как реакция приложения на сообщение WM_PAINT, посылаемое системой главному окну приложения при событии его "перерисовки";

3) для обработки сообщений, связанных с нажатием указанных кнопок диалогового окна можно использовать стандартные обработчики сообщений.

Приложение создается на базе типового каркаса MFC-приложения. Для создания диалогового окна используется встроенный редактор ресурсов как альтернатива описанию ресурса - диалоговое окно текстом в файле ресурсов. ПОРЯДОК (схема) выполнения задачи представлен ниже.

1. Спроектировать приложение.

1.1. Спроектировать интерфейс приложения (здесь диалоговое окно на фоне главного).

1.1.1. Определить состав элементов управления окна и их вид. При необходимости описать окна и ЭУ на соответствующем языке описания ресурсов. Здесь интерфейсные формы включают, как показано ниже, главное окно и диалоговое окно, выводимое на фоне главного, с двумя командными кнопками OK и CANCEL.

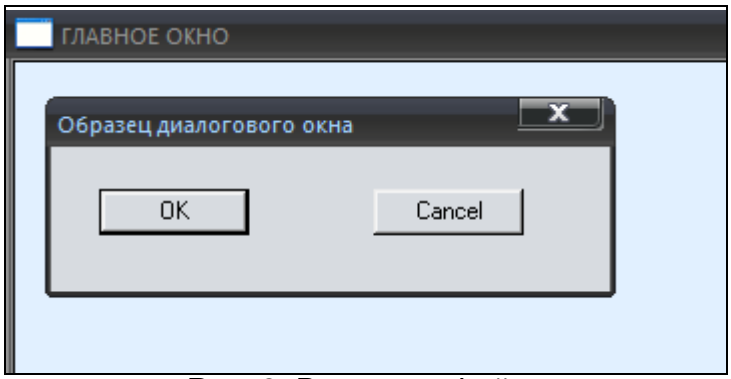


Рис. 9. Вид интерфейса

Примерный формат описания окна на языке описания ресурсов представлен ниже

ДескрипторОкна **DIALOG** DISCARDABLE *КоординатыОкна*
STYLE *СтильОкна1* | *СтильОкна2* | …
CAPTION *ЗаголовокОкна*
FONT *ИспользуемыйШрифт*
BEGIN
ОписаниеЭлементаУправления1
ОписаниеЭлементаУправления2
……..
END

Стили диалоговых окон представлены в Приложении 1.

1.1.2. Специфицировать состав событий-сообщений при работе с диалоговым окном и реакций приложения на эти сообщения:

- событие “нажатие кнопки ОК” вызывает сообщение WM_COMMAND. Реакция стандартная – закрытие окна. Макрокоманда включения чувствительности окна к сообщениям этого типа

ON_COMMAND (IDOK, <*ИмяПользовательскогоОбработчика*>),

например ON_COMMAND (IDOK, OnButtonOK). Поскольку здесь используется стандартный обработчик, то не обязательно указывать явно этот тип сообщений в очереди обрабатываемых сообщений. Здесь IDOK – стандартный дескриптор кнопки ОК;

- событие “нажатие кнопки CANCEL”, вызывает сообщение WM_COMMAND. Реакция стандартная – закрытие окна. Макрокоманда включения чувствительности окна к сообщениям этого типа

ON_COMMAND (IDCANCEL, <*ИмяПользовательскогоОбработчика*>).

Например, ON_COMMAND (IDCANCEL, OnButtonCANCEL). Поскольку используется стандартный обработчик, то не обязательно указывать явно этот тип сообщений в очереди обрабатываемых сообщений. Здесь IDCANCEL – стандартный дескриптор кнопки CANCEL.

1.1.3. Специфицировать состав событий-сообщений при работе с главным окном и реакций приложения на эти сообщения:

- события “запуск приложения с перерисовкой окна”, “изменение размеров окна”, “перекрытие окна” и др., приводят к посылке сообщений типа WM_PAINT. Реакция пользовательская – открытие (визуализация) диалогового окна с передачей ему управления. Макрокоманда включения чувствительности окна к сообщениям этого типа ON_WM_PAINT ( ). Поскольку здесь используется пользовательский обработчик

afx_msg void OnPaint() { … }; ,

то в очереди обрабатываемых сообщений главного окна необходимо явно указать этот тип сообщений.

1.2. Спроектировать классы приложения. Здесь используются три пользовательских класса. Это класс

class APPLICATION: public CWinApp

для создания экземпляра приложения, класс

class WINDOW: public CFrameWnd

для создания объекта – главное окно приложения, класс

class MY_DIALOG: public CDialog

для создания объекта – диалоговое окно. Состав и диаграммы классов, отображающие порядок наследования классов, используемых при создании приложения, представлены на рисунках ниже.

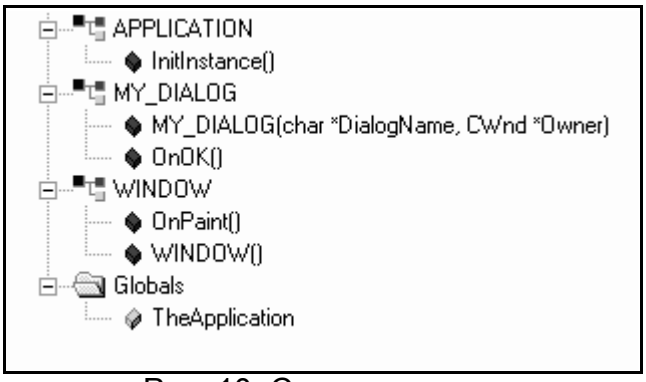


Рис. 10. Состав классов

Рис. 11. Диаграмма классов

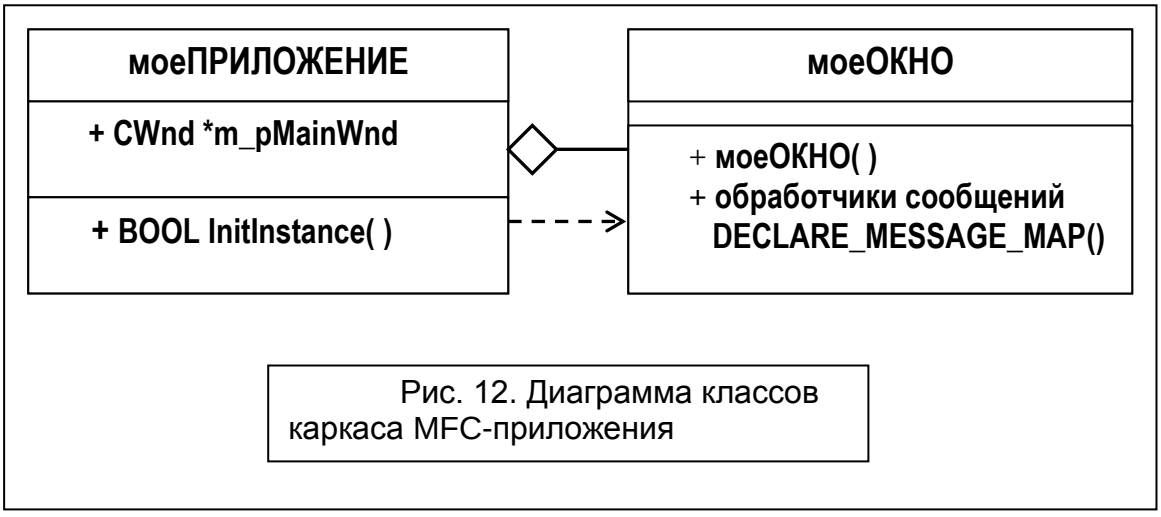


Рис. 12. Диаграмма классов каркаса MFC-приложения

1.3. Спроектировать модульную структуру приложения. Здесь для представления программной части (за исключением ресурсов) может использоваться один модуль.

2. Подготовить (реализовать) каркас MFC-приложения с файлом ресурсов.

2.1. Создать типовой каркас приложения. Выполнить и убедиться в его работоспособности.

2.2. Создать в составе приложения файл описания ресурсов Resource Script (или просто файл ресурсов) с расширением - rc. Для этого выполнить команду ГМ-Project-AddToProject-New-Files, в качестве типа добавляемого файла указать – Resource Script, а в качестве имени файла задать любое имя, например, <ИмяПриложения>. Автомати-

чески к проекту будет добавлен Resource Script файл с именем <ИмяПриложения>.rc>. Здесь ИмяПриложения – main. Соответственно файловый (модульный) состав приложения представлен ниже и включает два файла.

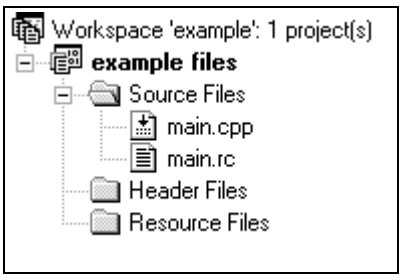


Рис. 13. Файловый состав приложения

А в папке проекта соответствующего приложения появятся файлы с названиями main.rc и заголовочный файл resource.h.

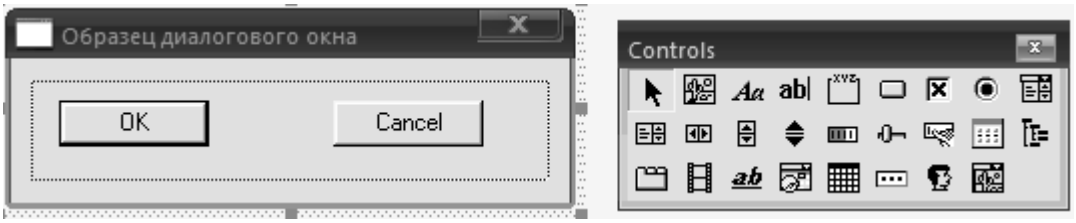
Рис. 14. Файловый состав приложения

2.3. Подключить файл описания ресурсов к приложению посредством команды #include "resource.h".

2.4. Выполнить приложение (диалоговое окно не появится!).

3. Создать новый ресурс - диалоговое окно.

3.1. Добавить ресурс к проекту приложения командой главного меню Insert-Resource-Dialog. Появится шаблон окна, показанный ниже. Окну автоматически будет присвоен дескриптор (идентификационным номером), например ID - IDD_DIALOG1. Дескриптор можно посмотреть в свойствах окна и при желании его можно изменить. Для этого достаточно вызвать контекстное меню для шаблона окна, выполнить пункт properties и считать данные из окна свойств “Dialog Properties”. Кроме этого в поле редактирования должна появиться и панель инструментов (Controls).



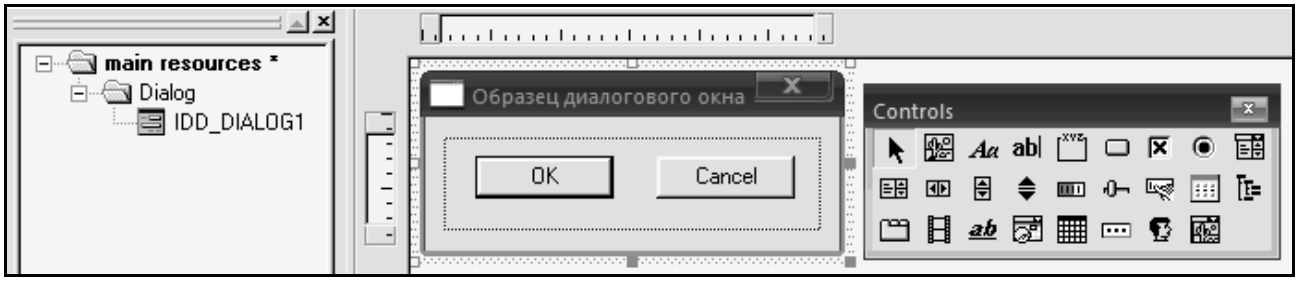


Рис. 15. Редактор ресурсов

В случае отсутствия панель инструментов можно активизировать через главное меню, выполнив пункты Tools-Customize… . В появившемся окне Customize на вкладке Toolbars следует включить Controls.

3.2. Отредактировать внешний вид (облик) окна. Например, изменить размер и положение окна, размеры и расположение кнопок, перемещая их “мышью”. Настроить свойства (атрибуты) окна, используя вкладки окна свойств “Dialog Properties”. Например, задать стиль (Styles) окна как Popup и (Border) как окно с рамкой Dialog Frame (или Resizing!). Задать название заголовка окна Caption, например “Образец диалогового окна”, как показано на вкладках ниже. Аналогично настроить свойства (атрибуты) кнопок (специализированных окон), используя вкладки окна свойств “Push Button Properties”.

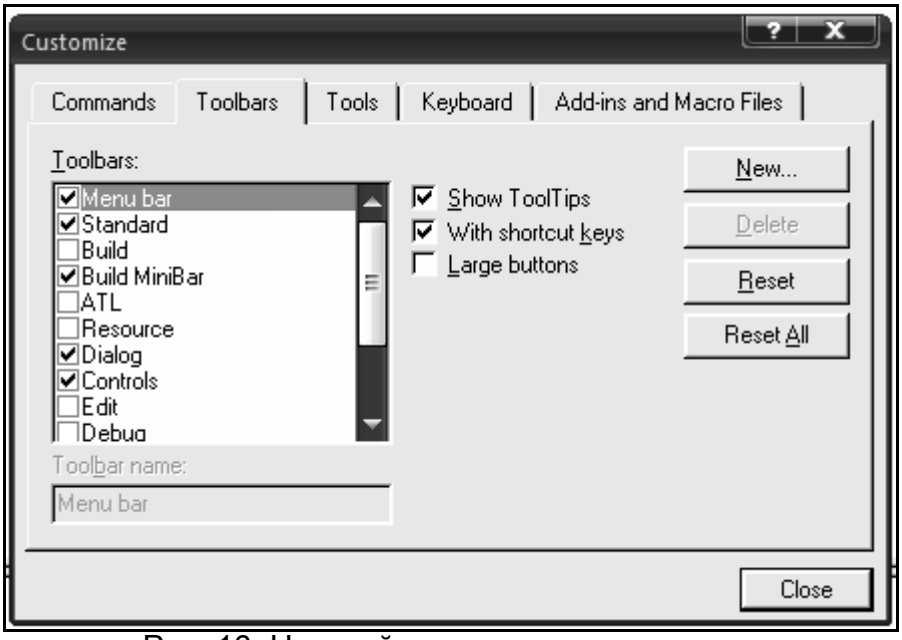


Рис. 16. Настройка панели инструментов

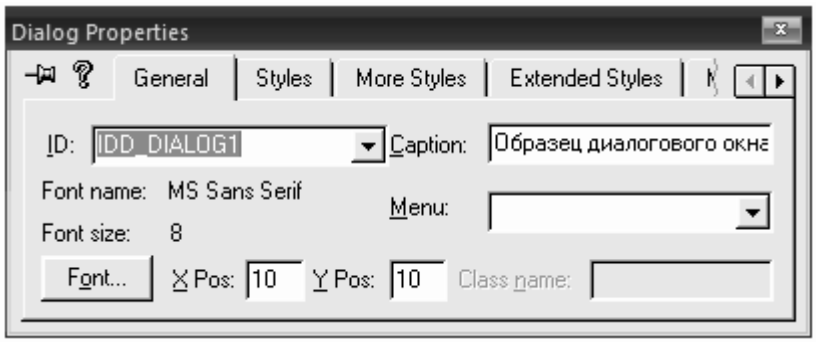


Рис. 17. Окно свойств диалогового окна

Рис. 18. Окно свойств диалогового окна

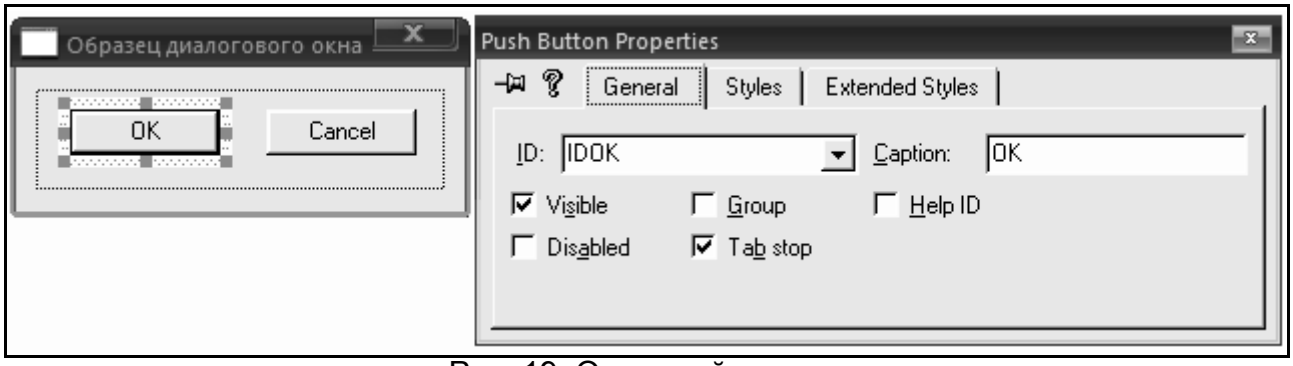


Рис. 19. Окно свойств кнопки

Заметьте, что для системных кнопок ID установлены как IDOK, IDCANCEL. Это обеспечит автоматическую стандартную обработку событий их выбора – завершение работы с модальным диалоговым окном, стирание его с экрана. Проверьте работу окна – пункт главного меню Layout-Test.

3.3. Откомпилировать приложение и запустить на выполнение (диалоговое окно не появится!).

Теперь можно просмотреть в текстовом редакторе (например, в Word) содержимое ресурсного файла (здесь main.rc), находящегося в папке проекта. Фрагменты содержимого приведены ниже.

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
 ……..
////////////////////////////////////////////////////////////////////////////
// Dialog
IDD_DIALOG1        DIALOG                    DISCARDABLE          0, 0, 186, 44
STYLE              DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION            "Образец диалогового окна"
FONT               8,  "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON     "OK",         IDOK,          16, 13, 50, 14
    PUSHBUTTON        "Cancel",     IDCANCEL,      107, 13, 50, 14
END
////////////////////////////////////////////////////////////////////////////
……..
```

Соответственно содержимое файла resource.h представлено ниже. В нем можно получить числовые эквиваленты дескрипторов. Например, видно, что поименованному дескриптору (ID) диалогового окна IDD_DIALOG1 соответствует целое число 101. В дальнейшем ссылаться на соответствующий объект можно, используя любое представление ID.

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by main.rc
//
#define IDD_DIALOG1                 101
// Next default values for new objects
//
```

```
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE      103
#define _APS_NEXT_COMMAND_VALUE       40001
#define _APS_NEXT_CONTROL_VALUE       1000
#define _APS_NEXT_SYMED_VALUE         101
#endif
#endif
```

Состав ресурсов можно увидеть в соответствующем окне ResourceView системы Visual Studio как показано на рисунке ниже.

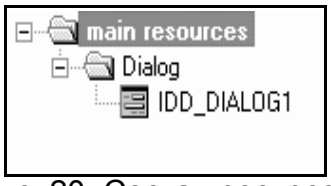


Рис. 20. Состав ресурсов

4. Описать пользовательский класс для создания экземпляров пользовательских диалоговых окон.

Здесь в тексте каркаса приложения необходимо описать класс MY_DIALOG, обслуживающий диалоговое окно IDD_DIALOG1, как производный от библиотечного класса MFC CDialog:

```
class  MY_DIALOG: public CDialog
{
public:
     MY_DIALOG ( char  * DialogName, CWnd  *Owner ): CDialog ( DialogName, Owner)
                    { };
     //afx_msg прототипы обработчиков сообщений
     DECLARE_MESSAGE_MAP( )
};
```

карту сообщений

```
BEGIN_MESSAGE_MAP (MY_DIALOG, CDialog)
    //Макрокоманды указания типов обрабатываемых сообщений и их обработчиков
END_MESSAGE_MAP ( )  .
```

В составе класса MY_DIALOG необходимо описать конструктор MY_DIALOG (char *DialogName, CWnd *Owner), где *DialogName* – дескриптор диалогового окна в строковом формате, *Owner* - указатель окна - владельца, родителя диалогового окна (например, текущее окно. Тогда в качестве Owner можно использовать указатель this).

При необходимости описываются пользовательские обработчики сообщений (здесь не используются).

Компилировать и выполнить (диалоговое окно не появится!).

5. Подключить диалоговое окно к приложению – оно будет запускаться сообщением WM_PAINT.

5.1. Включить в описание класса WINDOW главного окна обработчик сообщения WM_PAINT, т.е. описать функцию afx_msg void OnPaint() как член класса WINDOW:

```
class  WINDOW : public CFrameWnd
{
public:
     WINDOW( );
     afx_msg void OnPaint( );
     DECLARE_MESSAGE_MAP( )
};                    .
```

5.2. Включить в карте сообщений (очереди сообщений) главного окна (класса WINDOW) чувствительность к сообщениям типа WM_PAINT:

```
BEGIN_MESSAGE_MAP (WINDOW, CFrameWnd)
   ON_WM_PAINT( )
END_MESSAGE_MAP( )   .
```

5.3. Создать каркас функции-обработчика сообщения afx_msg void OnPaint()

```
afx_msg void WINDOW::OnPaint ( )
{
     CPaintDC dc ( this );
     // <ФРАГМЕНТ ПОЛЬЗОВАТЕЛЯ>
};
```

5.4. Описать функцию-обработчик сообщения afx_msg void OnPaint(). Создать в ней экземпляр класса MY_DIALOG с именем TheDialog, инициализировать его параметрами (обликом) ресурса типа “диалоговое окно” с дескриптором ID = IDD_DIALOG1. Визуализировать и активизировать окно методом DoModal ( ) :

```
afx_msg void WINDOW::OnPaint()
{
    CPaintDC  dc ( this );
    MY_DIALOG  TheDialog ( ( LPTSTR ) IDD_DIALOG1 , this );
    TheDialog.DoModal ( );
};    .
```

6. Откомпилировать и выполнить приложение. Тестировать работу приложения.

2.7. Задания для самостоятельного выполнения

Для каждого из указанных ниже заданий до реализации приложения разработать и описать:

а) вид интерфейса приложения;

б) диаграмму робастности интерфейса приложения;

в) диаграмму состояний;

г) диаграмму классов.

1*. Создать аналогичное приложение с переопределением обработчика сообщения по нажатию кнопки OK в виде подтверждения нажатия этой кнопки.

2. Создать аналогичное приложение с переопределением обработчика сообщения по нажатию кнопки CANCEL в виде подтверждения нажатия этой кнопки и завершения работы диалогового окна (для завершения работы окна использовать метод EndDialog()).

3. Создать приложение аналогичное п.1 и 2 с запуском диалогового окна каждый раз только по нажатию левой клавиши мыши (сообщение WM_LBUTTONDOWN, прототип обработчика - **afx_msg void OnLButtonDown(UINT** *ОписаниеСитуации***, CPoint** *Координаты Курсора* **)**).

4*. Создать приложение аналогичное п.3 с запуском диалогового окна каждый раз по нажатию левой клавиши мыши (сообщение WM_LBUTTONDOWN) или по набору заранее определенного символа (сообщение WM_CHAR, прототип обработчика - **afx_msg void OnChar(UINT** *ASCII_КодКлавиши***, UINT** *ЧислоПовторов***, UINT** *ОписаниеСитуации***)**).

5. Создать приложение аналогичное п.3 с завершением работы с приложением по нажатию кнопки CANCEL.

6. Создать приложение с выводом диалогового окна (с одной кнопкой ОК) до главного окна с сообщением о запуске приложения. По нажатию кнопки ОК закрывать диалоговое окно и активизировать главное.

7. Создать приложение с выводом диалогового окна до главного окна аналогично п.6, а затем и в главном окне аналогично п.3.

8. Создать аналогичное п.7 приложение. При запуске приложения до появления главного окна должно выводиться диалоговое окно с одной кнопкой ОК и справкой о приложении. После нажатия кнопки ОК диалоговое окно закрывается, передавая управление главному окну приложения. По нажатию левой клавиши мыши аналогично п.3 должно выводиться диалоговое окна с кнопками ОК и CANCEL. После нажатия CANCEL диалоговое окно должно закрываться, передавая управление главному окну приложения.

9. Создать приложение с выводом диалогового окна после главного окна каждый раз по нажатию левой клавиши мыши. Окно должно предлагать вывод справки о приложении при нажатии кнопки ОК, а по нажатию кнопки CANCEL закрывать диалог. После нажатия ОК должно выводить следующее - справочное окно с информацией о приложении и кнопкой ОК, после нажатия которой окно закрывается, возвращая управление главному окну приложения (или предыдущему диалоговому окну).

10*. Повторить п.9, инициализируя запрос на справку мышью.

11. Создать приложение с диалоговым окном в качестве интерфейса приложения вместо главного окна. По кнопке ОК выполнять пользовательское действие, по CANCEL завершать работу приложения.

12. Поменять атрибуты, стиль диалогового окна на базе любого из предыдущих заданий (например, для п. 3): - выполнить смещение диалогового окна в левый верхний угол главного окна; - разместить окно примерно по центру экрана и т.д.

13*. Создать немодальное диалоговое окно.

ПРИМЕЧАНИЕ: пункты, помеченные *, выполняются по указанию преподавателя

2.8. Диалоговое окно в составе главного окна. Переопределение обработчиков сообщений

ЗАДАНИЕ № 2. *Создать MFC-приложение на базе ТКП с оконным интерфейсом из главного окна и диалогового окна с двумя кнопками OK, CANCEL.*

При запуске приложения должно выводиться главное окно.

При щелчке левой клавишей мыши в главном окне должно активизироваться диалоговое окно и выводиться на фоне главного окна приложения. Диалоговое окно

функционирует в модальном режиме и после нажатия любой из кнопок (OK или CANCEL) исчезает, возвращая управление главному окну.

При нажатии кнопки OK должно также выводиться сообщение, подтверждающее факт нажатия именно этой кнопки.

Из описания задания следует:

1) визуализация диалогового окна должна происходить каждый раз как реакция на сообщение главному окну типа WM_LBUTTONDOWN;

2) для обработки сообщений, связанных с нажатием кнопки CANCEL диалогового окна можно использовать стандартный обработчик сообщений;

3) для обработки сообщений, связанных с нажатием кнопки OK, используется пользовательский обработчик сообщений, подтверждающий с помощью функции MessageBox факт нажатия именно этой кнопки и завершающий работу диалогового окна.

Приложение создается на базе типового каркаса MFC-приложения. Для создания диалогового окна используется встроенный редактор ресурсов как альтернатива описанию ресурса - диалоговое окно текстом в файле ресурсов. ПОРЯДОК (схема) выполнения задачи представлен ниже.

1. Спроектировать приложение

1.1. Спроектировать интерфейс.

1.1.1. Определить состав элементов управления окна и их вид. При необходимости описать окна и ЭУ на соответствующем языке описания ресурсов. Здесь интерфейсные формы включают, как показано ниже, главное окно и диалоговое окно, выводимое на фоне главного, с двумя командными кнопками OK и CANCEL.

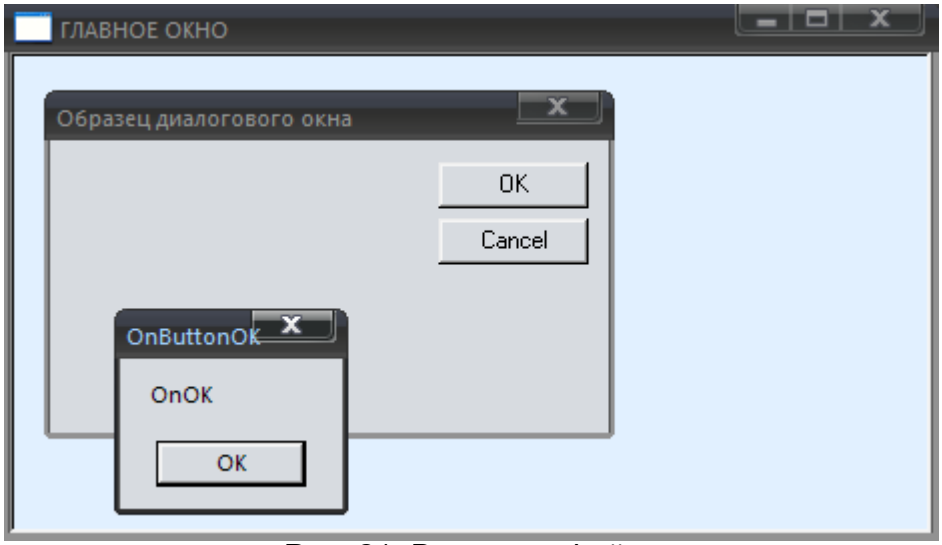


Рис. 21. Вид интерфейса

1.1.2. Специфицировать состав событий-сообщений при работе с диалоговым окном и реакций приложения на эти сообщения:

- событие “нажатие кнопки OK” вызывает сообщение WM_COMMAND. Реакция пользовательская – вывод сообщения и закрытие окна. Поскольку используется пользовательский обработчик, то в очереди обрабатываемых сообщений нужно указать этот тип сообщений. Для этого используется макрокоманда ON_COMMAND (IDOK, <ИмяПользовательскогоОбработчика>).

Пусть здесь используется обработчик с прототипом afx_msg void OnButtonOK()

```
afx_msg void MY_DIALOG::OnButtonOK( )
```

```
{
    MessageBox( "OnOK","OnButtonOK" );
    EndDialog ( 0 );
};  .
```

Макрокоманда включения имеет вид ON_COMMAND (IDOK, OnButtonOK);
- событие “нажатие кнопки CANCEL” вызывает сообщение WM_COMMAND. Реакция стандартная – закрытие окна. Используется стандартный обработчик и не обязательно указывать явно этот тип сообщений в очереди обрабатываемых сообщений.

1.1.3. Специфицировать состав событий-сообщений при работе с главным окном и реакций приложения на эти сообщения:

- события “запуск приложения с перерисовкой окна”, “изменение размеров окна”, “перекрытие окна” и др., приводят к посылке сообщений типа WM_PAINT. Реакция пользовательская – здесь никаких действий не производится, но они могут быть добавлены в дальнейшем. Поскольку здесь используется пользовательский обработчик, то в очереди обрабатываемых сообщений главного окна необходимо явно указать этот тип сообщений макрокомандой включения ON_WM_PAINT();
- событие “нажатие левой клавиши мыши” вызывает сообщение WM_LBUTTONDOWN. Реакция на сообщения этого типа пользовательская – открытие (визуализация) диалогового окна с передачей ему управления. Макрокоманда включения чувствительности окна к сообщениям этого типа ON_WM_LBUTTONDOWN ( ). Макрокоманде ON_WM_LBUTTONDOWN() соответствует обработчик с прототипом afx_msg void OnLButtonDown(UINT flags, CPoint loc)

```
afx_msg void WINDOW::OnLButtonDown(UINT flags, CPoint loc)
{
    CClientDC dc(this);
    MY_DIALOG TheDialog( (LPTSTR) IDD_DIALOG1, this)  ;
    TheDialog.DoModal();
};   .
```

1.2. Спроектировать классы приложения. Здесь используется три пользовательских класса. Это класс APPLICATION для создания экземпляра приложения; класс WINDOW для создания объекта – главное окно приложения; класс MY_DIALOG для создания объекта – диалоговое окно. Диаграмма классов, отображающая порядок наследования основных классов, используемых при создании приложения, аналогична диаграмме классов ЗАДАНИЯ № 1.

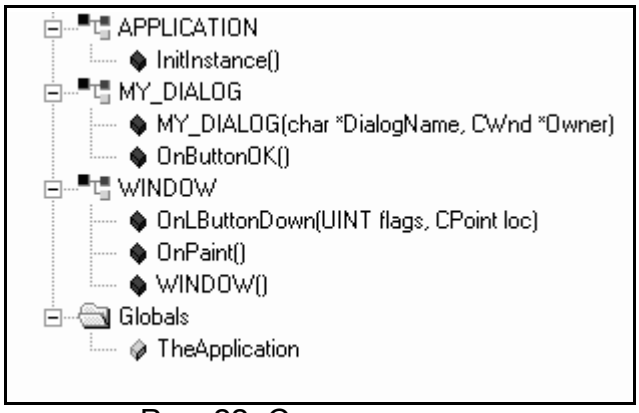


Рис. 22. Состав классов

1.3. Спроектировать модульную структуру приложения. Здесь для представления программной части (за исключением ресурсов) может использоваться один модуль, например main.cpp.

2. Подготовить (реализовать) каркас MFC-приложения с файлом ресурсов.

2.1. Создать типовой каркас приложения. Выполнить и убедиться в его работоспособности.

2.2. Создать в составе приложения файл описания ресурсов Resource Script (или просто файл ресурсов) с расширением - rc.

Для этого выполнить команду ГМ-Project-AddToProject-New-Files, в качестве типа добавляемого файла указать – Resource Script, а в качестве имени файла задать любое имя, например, <ИмяПриложения>.

Автоматически к проекту будет добавлен Resource Script файл с именем <ИмяПриложения>.rc>. Здесь ИмяПриложения – main. Соответственно файловый (модульный) состав приложения представлен ниже и включает два файла.

2.3. Подключить файл описания ресурсов к приложению посредством команды

#include "resource.h".

2.4. Выполнить приложение (диалоговое окно не появится!).

3. Создать новый ресурс - диалоговое окно.

3.1. Добавить ресурс к проекту приложения командой главного меню Insert-Resource-Dialog. Появится шаблон окна. Окну автоматически будет присвоен дескриптор (идентификационным номером), например ID - IDD_DIALOG1.

3.2. Отредактировать внешний вид (облик) окна.

Например, изменить размер и положение окна, размеры и расположение кнопок, перемещая их “мышью”.

Настроить свойства (атрибуты) окна, используя вкладки окна свойств “Dialog Properties”. Например, задать стиль (Styles) окна как Popup и (Border) как окно с рамкой Dialog Frame (или Resizing!). Задать название заголовка окна Caption.

Аналогично настроить свойства (атрибуты) кнопок (специализированных окон), используя вкладки окна свойств “Push Button Properties”.

Заметьте, что для системных кнопок ID установлены как IDOK, IDCANCEL. Это обеспечит автоматическую стандартную обработку событий их выбора – завершение работы с модальным диалоговым окном, стирание его с экрана. Однако для кнопки ОК обработчик в дальнейшем будет переопределен на пользовательский, а для Cancel будет использован стандартный.

Проверьте работу окна – пункт главного меню Layout-Test.

3.3. Откомпилировать приложение и запустить на выполнение (диалоговое окно не появится!).

4. Описать пользовательский класс для создания экземпляров пользовательских диалоговых окон.

Здесь в тексте каркаса приложения необходимо описать класс MY_DIALOG, обслуживающий диалоговое окно IDD_DIALOG1, как производный от библиотечного класса MFC - CDialog:

```
class MY_DIALOG: public CDialog
{
public:
    MY_DIALOG ( char * DialogName, CWnd *Owner ): CDialog ( DialogName, Owner)
                { };
```

```
    afx_msg void OnButtonOK ( );
    DECLARE_MESSAGE_MAP( )
};
```

карту сообщений

```
BEGIN_MESSAGE_MAP (MY_DIALOG, CDialog)
      ON_COMMAND(IDOK, OnButtonOK)
END_MESSAGE_MAP ( )  .
```

В составе класса MY_DIALOG необходимо описать конструктор MY_DIALOG (char *DialogName, CWnd *Owner), где DialogName – дескриптор диалогового окна в строковом формате, Owner - указатель окна - владельца, родителя диалогового окна (например, текущее окно. Тогда в качестве Owner можно использовать указатель this).

Переопределить обработчик

```
afx_msg void MY_DIALOG::OnButtonOK()
{
    MessageBox("OnOK","OnButtonOK");
    EndDialog(0);
};
```

Откомпилировать и запустить на выполнение (диалоговое окно не появится!).

5. Подключить диалоговое окно к приложению – оно будет запускаться сообщением ON_WM_LBUTTONDOWN.

5.1. Включить в описание класса WINDOW главного окна обработчик сообщения ON_WM_LBUTTONDOWN, т.е. описать функцию afx_msg void OnLButtonDown(UINT flags, CPoint loc) как член класса WINDOW:

```
class WINDOW : public CFrameWnd
{
public:
    WINDOW( );
    afx_msg void OnPaint();
    afx_msg void OnLButtonDown(UINT flags, CPoint loc);
    DECLARE_MESSAGE_MAP()
};
```

5.2. Включить в карте сообщений (очереди сообщений) главного окна (класса WINDOW) чувствительность к сообщениям типа WM_PAINT и WM_LBUTTONDOWN:

```
BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
  ON_WM_PAINT()
  ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()  .
```

5.3. Создать каркас функции-обработчика сообщения afx_msg void OnPaint()

```
afx_msg void WINDOW::OnPaint()
{
    CPaintDC dc(this);
    // < ФРАГМЕНТ_ПОЛЬЗОВАТЕЛЯ >
};
```

5.4. Создать каркас функции-обработчика сообщения afx_msg void OnLButton-Down(UINT flags, CPoint loc)

```
afx_msg void WINDOW::OnLButtonDown(UINT flags, CPoint loc)
{
    CClientDC dc(this);
    // < ФРАГМЕНТ_ПОЛЬЗОВАТЕЛЯ >
};
```

5.5. Описать функцию-обработчик сообщения WM_LBUTTONDOWN - afx_msg void OnLButtonDown(UINT flags, CPoint loc). Создать в ней экземпляр класса MY_DIALOG с именем TheDialog, инициализировать его параметрами (обликом) ресурса типа “диалоговое окно” с дескриптором ID = IDD_DIALOG1. Визуализировать и активизировать окно методом DoModal ( ):

```
afx_msg void WINDOW:: OnLButtonDown(UINT flags, CPoint loc)
{
   MY_DIALOG TheDialog( (LPTSTR) IDD_DIALOG1 , this);
   TheDialog.DoModal ( );
};
```

6. Откомпилировать и выполнить приложение. Тестировать работу приложения.

2.9. Задания для самостоятельного выполнения

1*. Создать аналогичное приложение с выводом сообщения “Нажата левая клавиша”.

2. Создать аналогичное приложение с выводом сообщения “Нажата левая клавиша”, начиная с позиции, где был выполнен щелчок.

3. Создать аналогичное приложение с выводом координат позиции, где был выполнен щелчок, начиная с позиции, где был выполнен щелчок.

4*. Создать приложение, аналогичное п. 3, с выводом координат позиции, где был выполнен щелчок, в верхнем левом углу главного окна.

5. Создать приложение, аналогичное п. 4, решив проблему перерисовки: а) дублировать вывод в обработчике сообщения WM_PAINT; б) весь вывод реализовать в обработчике сообщения WM_PAINT, вызов обновления-перерисовки инициировать посылкой сообщения методом InvalidateRect ( ).

6*. Создать приложение, аналогичное п. 5, но с запуском диалогового окна щелчком правой клавишей мыши.

7. Создать аналогичное приложение с запуском диалогового окна щелчком как левой так и правой клавишей мыши.

8*. Создать аналогичное приложение с запуском диалогового окна вводом заданного символа.

ПРИМЕЧАНИЕ: пункты, помеченные *, выполняются по указанию преподавателя.

2.10. Диалоговое окно в качестве главного окна

ЗАДАНИЕ № 3. *Создать MFC-приложение на базе ТКП с оконным интерфейсом из диалогового окна в качестве главного. Диалоговое окно содержит системное меню и кнопку "Справка", при нажатии на которую выводится справочная информация о приложении. Завершение работы приложения производится соответствующей кнопкой системного меню диалогового окна.*

Приложение создается на базе типового каркаса MFC-приложения в Visual Studio C++. Для создания самого ресурса – диалогового окна используется встроенный редактор ресурсов. ПОРЯДОК (схема) выполнения задачи представлен ниже.

1. Спроектировать приложение.

1.1. Спроектировать интерфейс приложения.

1.1.1. Определить состав элементов управления окна и их вид. При необходимости описать окна и ЭУ на соответствующем языке описания ресурсов. Здесь интерфейсные формы включают, как показано ниже, диалоговое окно, выводимое при запуске приложения в роли главного. Окно содержит кнопку "Справка", при нажатии которой выводится сообщение о приложении. При закрытии окна сообщения управление возвращается диалоговому окну.

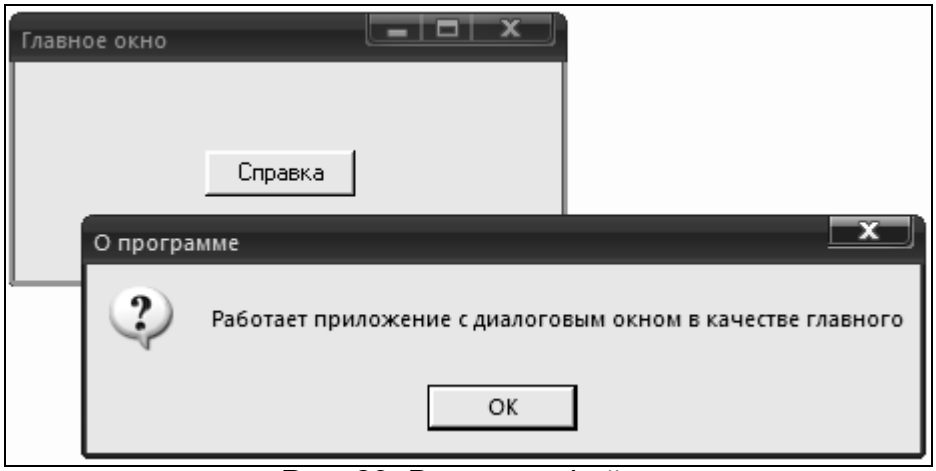


Рис. 23. Вид интерфейса

1.1.2. Специфицировать состав событий-сообщений при работе с диалоговым (главным) окном и реакций приложения на эти сообщения:

- событие "нажатие кнопки Справка" вызывает сообщение WM_COMMAND. Реакция пользовательская – вывод сообщения на фоне окна. Поскольку используется пользовательский обработчик, то в очереди обрабатываемых сообщений нужно указать этот тип сообщений - в очередь сообщений вставляется макрокоманда, например ON_COMMAND (IDOK, OnOK). Это означает, что сообщению WM_COMMAND соответствует обработчик с прототипом afx_msg void OnOK( ).

1.2. Спроектировать классы приложения. Здесь используются классы: класс class APPLICATION: public CWinApp для создания экземпляра приложения, класс class MY_DIALOG: public CDialog для создания объекта – диалоговое окно.

1.3. Спроектировать модульную структуру приложения. Здесь для представления программной части (за исключением ресурсов) используется один модуль, например main.cpp

2. Подготовить каркас MFC-приложения с файлом ресурсов. Создать типовой каркас приложения. Создать файл описания ресурсов и подключить его к приложению.

3. Создать новый ресурс - диалоговое окно. Добавить ресурс к проекту. Отредактировать внешний вид окна, настроить свойства окна.

Структура приложения представлена на рисунке ниже.

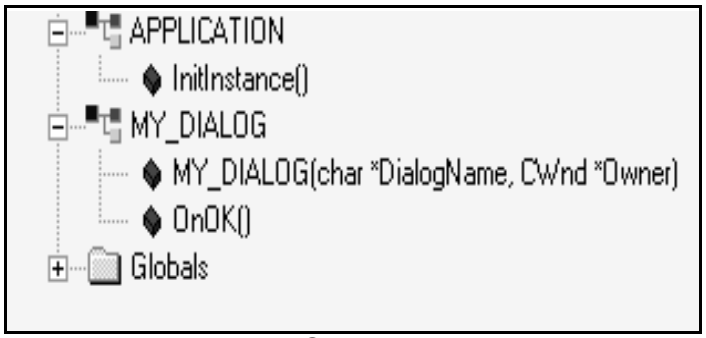


Рис. 24. Состав классов

4. Описать пользовательский класс MY_DIALOG для создания экземпляра главного окна

```
class MY_DIALOG : public CDialog
{
public:
   MY_DIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName, Owner) { };
     afx_msg void OnOK ( );
     DECLARE_MESSAGE_MAP()
};
```

карту сообщений

```
BEGIN_MESSAGE_MAP(MY_DIALOG, CDialog)
   ON_COMMAND(IDOK, OnOK)
END_MESSAGE_MAP()
```

Переопределить обработчик

```
afx_msg void MY_DIALOG::OnOK()
{
     MessageBox("Информация о приложении","О программе");
};
```

5. Подключить диалоговое окно к приложению – оно будет запускаться автоматически при запуске приложения. Для этого можно использовать конструктор приложения BOOL InitInstance(). Необходимо создать в нем экземпляр класса MY_DIALOG с именем TheDialog, инициализировать его параметрами (обликом) ресурса типа “диалоговое окно” с идентификатором ID = IDD_DIALOG1. Визуализировать и активизировать окно методом DoModal() как показано ниже

```
class APPLICATION : public CWinApp
{
```

```
public:
    BOOL InitInstance();
};

BOOL APPLICATION::InitInstance()
{
    MY_DIALOG TheDialog((LPTSTR) IDD_DIALOG1 , NULL);
    TheDialog.DoModal();
    return TRUE;
}
```

6. Откомпилировать и выполнить приложение.

2.11. Диалоговое окно с окном редактирования в качестве главного окна

ЗАДАНИЕ № 4. *Создать MFC-приложение на базе ТКП для многократного ввода числовых значений и их вывода с противоположным знаком.*

ПОРЯДОК (схема) выполнения задачи представлен ниже.

1. Спроектировать приложение.

1.1. Спроектировать интерфейс приложения.

1.1.1. Определить сценарий работы приложения, состав элементов управления окна и их вид.

Здесь интерфейсные формы включают, как показано ниже,

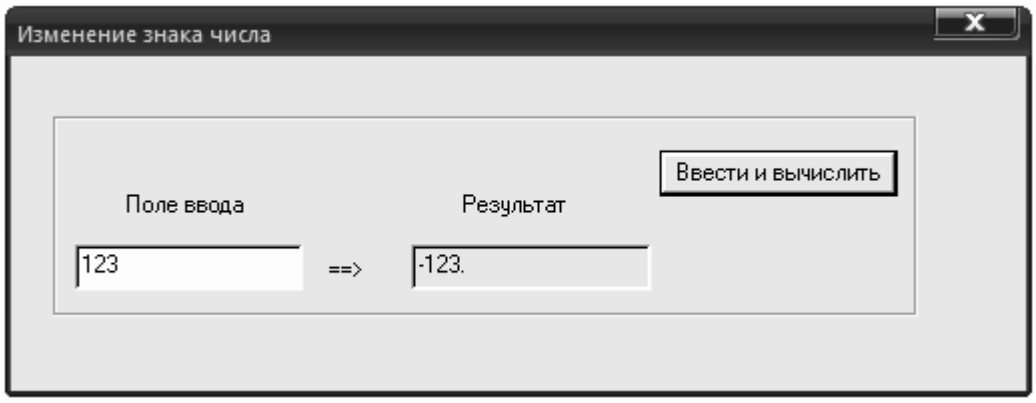


Рис. 25. Вид интерфейса

диалоговое окно, выводимое при запуске приложения в роли главного.

Окно содержит кнопку “Ввести и вычислить”, поле для ввода числа и поле для вывода результата. Закрытие приложения выполняется кнопкой системного меню.

Сценарий работы приложения описан ниже.

В поле ввода диалогового окна задается число, а ввод его в приложение осуществляется по нажатию клавиши Enter или кнопки окна “Ввести и вычислить” с одновременным отображением результата преобразования введенного значения в поле Результат.

Выход из приложения осуществляется по нажатию клавиши Esc или кнопки завершения системного меню диалогового окна.

Соответственно для обработки сообщений, связанных с кнопкой “Ввести и вычислить”, используется пользовательский обработчики сообщений, выполняющий ввод зна-

чения из поля ввода в виде строки, преобразование строки в число, выполнение необходимых действий над числом, преобразование числа в строку и вывод строки как результата.

1.1.2. Специфицировать состав событий-сообщений при работе с диалоговым (главным) окном и реакций приложения на эти сообщения:

- событие нажатие кнопки “Ввести и вычислить” вызывает сообщение WM_COMMAND.

Реакция пользовательская – считывание данного из окна ввода и его вывод после преобразования в окне вывода.

Поскольку используется пользовательский обработчик, то в очередь сообщений вставляется макрокоманда, например,

ON_COMMAND (IDOK, OnOK) .

Здесь обработчик с прототипом afx_msg void OnOK() обрабатывает сообщения нажатия кнопки “Ввести и вычислить”;

- событие запуска и активизации диалогового окна вызывает сообщение инициализации WM_INITDIALOG.

Реакция на это сообщение – вывод в окне ввода текста приглашения: "Введите число". Прототип обработчика:

BOOL OnInitDialog().

1.2. Спроектировать классы приложения.

Здесь используются классы: класс

class APPLICATION: public CWinApp

для создания экземпляра приложения; класс

class MY_DIALOG: public CDialog

для создания объекта – диалоговое окно.

Состав классов представлен на рисунке ниже.

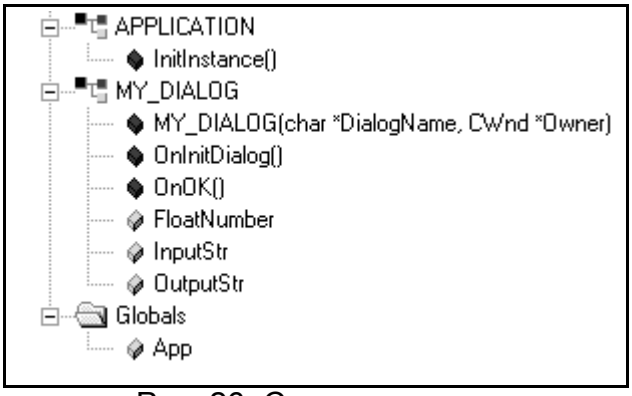


Рис. 26. Состав классов

Диаграммы указанных классов, основные члены, методы классов представлены на рисунке ниже.

Кроме этого неявно используются классы для создания экземпляров объектов элементов управления типа: – окно редактирования, статичное окно, рамка (группирующий элемент управления), командная кнопка.

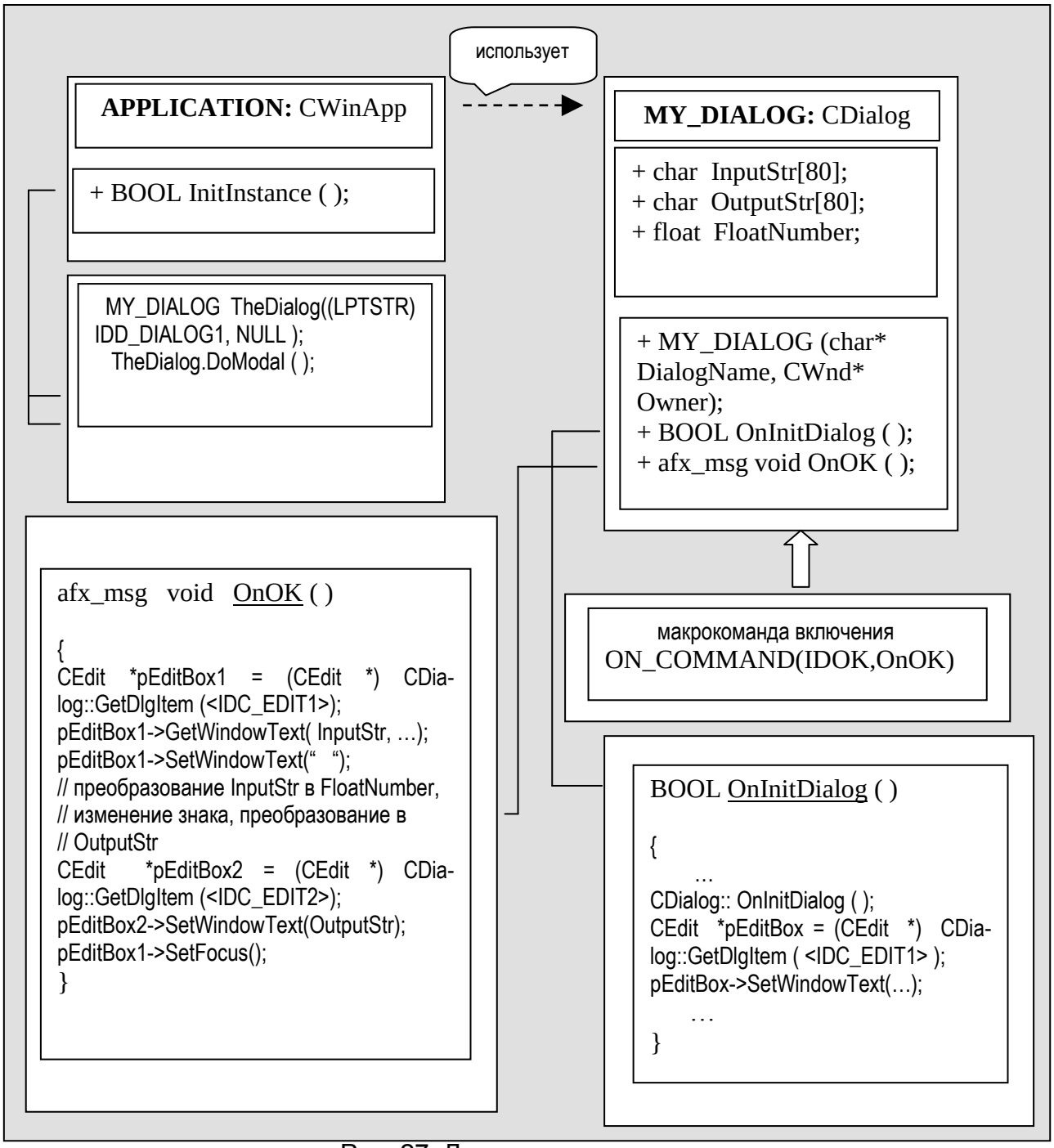


Рис. 27. Диаграмма классов

1.3. Спроектировать модульную структуру приложения. Здесь для представления программной части (за исключением ресурсов) используется один модуль.

2. Создать типовой каркас приложения, файл описания ресурсов и подключить его.

3. Создать новый ресурс - диалоговое окно. Добавить ресурс к проекту. Отредактировать внешний вид окна, настроить свойства окна.

Шаблон диалогового окна и контекстное меню для окошка редактирования (тип edit box, класс MFC – CEdit) представлены на рисунках ниже.

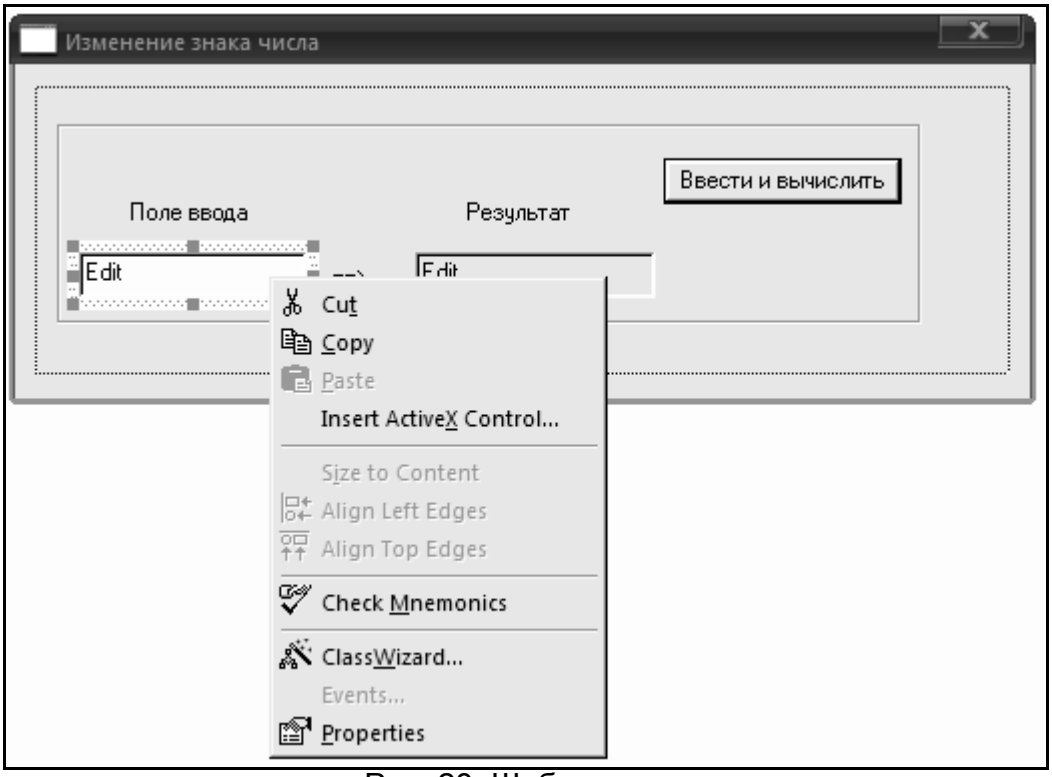


Рис. 28. Шаблон окна

Окно свойств ЭУ, используемого для ввода чисел, представлено ниже.

Рис. 29. Окно свойств диалогового окна

Для поля эхо-вывода также используется окошко (поле) редактирования с ID – IDC_EDIT2. Настройки окошка (вкладки General и Styles) выполнена с учетом использования этого элемента управления только в режиме вывода - отображения (read-only) данных и представлена на рисунках ниже.

Рис. 30. Окно свойств окна редактирования

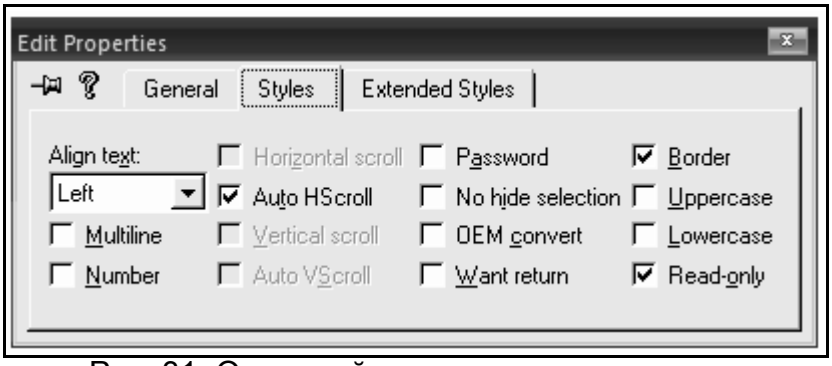


Рис. 31. Окно свойств окна редактирования

Окно свойств кнопки приведено ниже.

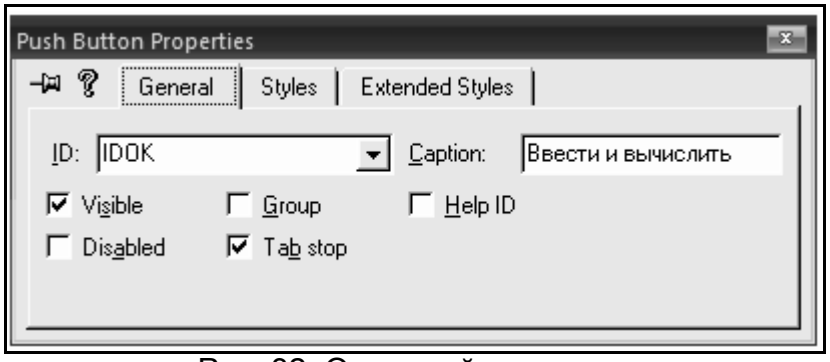


Рис. 32. Окно свойств кнопки

Настройка свойств группирующего элемента (рамки Group Box) и статичного поля для отображения стрелки показаны ниже

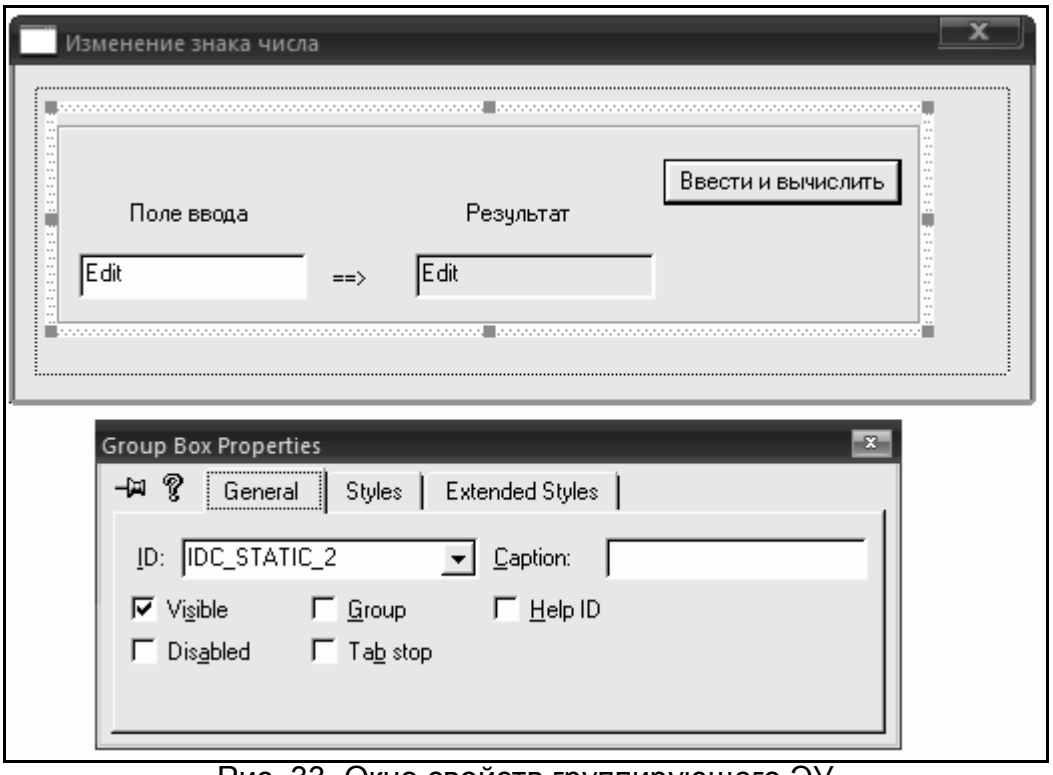


Рис. 33. Окно свойств группирующего ЭУ

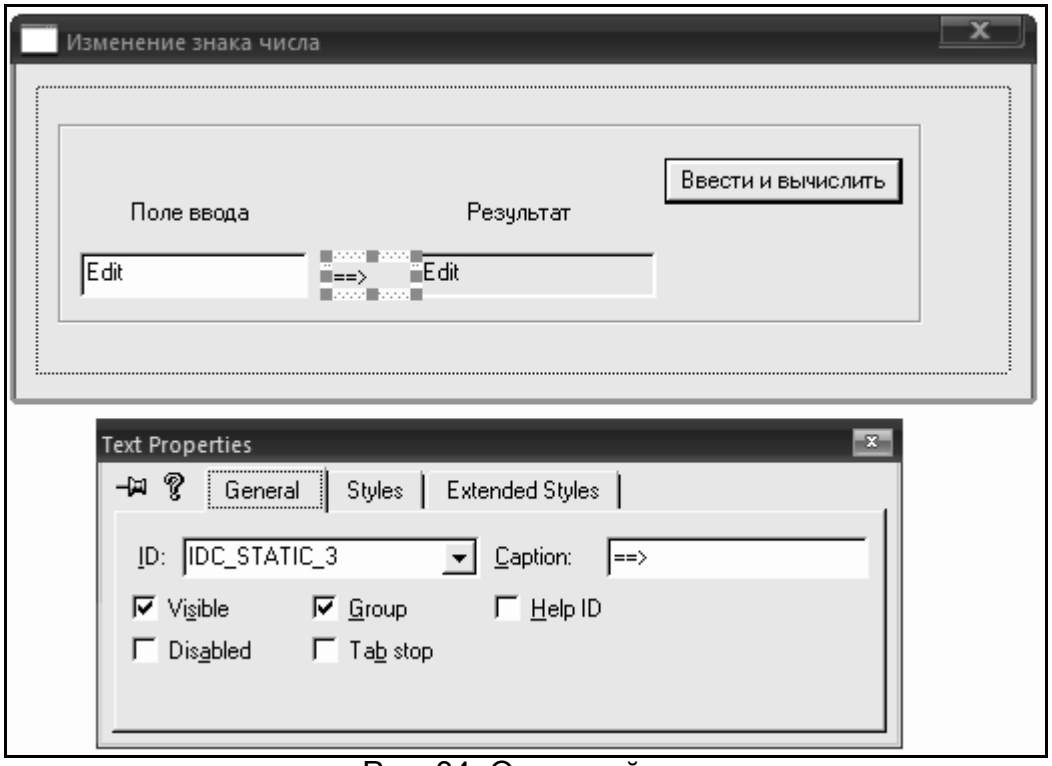


Рис. 34. Окно свойств

Фрагмент содержимого ресурсного файла main.rc приведено ниже

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
    ……..
//////////////////////////////////////////////////////////////////////////////
// Dialog
IDD_DIALOG1     DIALOG          DISCARDABLE  0, 0, 338, 103
STYLE           DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION         "Изменение знака числа"
FONT            8, "MS Sans Serif"
BEGIN
   DEFPUSHBUTTON   "Ввести и вычислить", IDOK,216,29,79,14
   EDITTEXT        IDC_EDIT1,21,58,76,14,ES_AUTOHSCROLL
   EDITTEXT        IDC_EDIT2,133,58,80,14,ES_AUTOHSCROLL | ES_READONLY
   CTEXT           "Поле ввода",IDC_STATIC_1,21,41,73,8
   CTEXT           "Результат",IDC_STATIC_2,129,41,76,8
   LTEXT           "==>",IDC_STATIC_3,105,61,26,8
   GROUPBOX        "",IDC_STATIC_2,14,15,288,65
END
//////////////////////////////////////////////////////////////////////////////
    ……..
```

4. Описать пользовательский класс MY_DIALOG для создания экземпляра главного окна как производный от библиотечного класса MFC CDialog и обслуживающий диалоговое окно IDD_DIALOG1.

Все пользовательские переменные, используемые для организации ввода-вывода значений, можно описать как глобальные переменные, но т.к. они используются только активным окном (class MY_DIALOG), то инкапсулируем их именно в класс окна в его открытую секцию. Например, используем переменные char InputStr[80], float FloatNumber,

char OutputStr[80] соответственно для хранения введенного значения в строковом и числовом форматах и выводимого результата в строковом формате.

В отличие от предыдущих примеров добавить в описание метод OnInitDialog( ), а также перегрузить обработчик afx_msg void OnOK( )

```
class MY_DIALOG : public CDialog
{
public:
      char  InputStr[80];
      char  OutputStr[80];
      float  FloatNumber;
       MY_DIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName, Owner) { };
      BOOL OnInitDialog ( );
      afx_msg void OnOK ( );
      DECLARE_MESSAGE_MAP()
};          .
```

Описать очередь сообщений

```
BEGIN_MESSAGE_MAP (MY_DIALOG,  CDialog)
    ON_COMMAND ( IDOK, OnOK)
END_MESSAGE_MAP( )          .
```

Описать метод OnInitDialog(), предназначенный для инициализации окошек редактирования – для задания их первоначального содержимого в момент вывода диалогового окна на экран

```
BOOL MY_DIALOG::OnInitDialog ( )
{
      CDialog::OnInitDialog ( );
      CEdit *pEditBox1 = (CEdit *) CDialog::GetDlgItem ( IDC_EDIT1 );
      pEditBox1 -> SetWindowText ( "Введите число" );
      return TRUE;
};                .
```

Описать обработчик сообщений afx_msg void OnOK( ), осуществляющий основные пользовательские действия по вводу значения в строковом формате как char InputStr[80], хранению значения в числовом формате как float FloatNumber и выводе результата в строковом формате как char OutputStr[80] соответственно

```
afx_msg void MY_DIALOG::OnOK ( )
{
      CEdit *pEditBox1 = ( CEdit * ) CDialog::GetDlgItem ( IDC_EDIT1 );
       //== ввод значения в виде строки
       pEditBox1 -> GetWindowText ( InputStr, sizeof InputStr-1 );
       pEditBox1 -> SetWindowText ( "      " );
       //== преобразование значения в тип float
        FloatNumber = atof ( InputStr );
       // == < ФРАГМЕНТ ПО ИСПОЛЬЗОВАНИЮ FloatNumber >
```

```
        FloatNumber = - FloatNumber;
        //== преобразование значения из типа float в строку
        _gcvt (FloatNumber, 15, OutputStr);
       CEdit *pEditBox2 = ( CEdit * ) CDialog::GetDlgItem( IDC_EDIT2 );
       //== вывод значения в виде строки
       pEditBox2 -> SetWindowText ( OutputStr );
       //== передача окошку ввода управления (фокуса)
       pEditBox1 -> SetFocus ( );
};  .
```

5. Подключить диалоговое окно к приложению – оно должно запускаться автоматически при инициализации пользовательского приложения в качестве интерфейсного окна вместо главного окна. Для этого необходимо скорректировать содержимое функции InitInstance ( ). А именно - необходимо создать диалоговое окно, например, как экземпляр TheDialog класса MY_DIALOG, инициализировав его параметрами облика (стиля) ресурса с идентификатором ID = IDD_DIALOG1. А затем необходимо провести визуализацию и активизацию окна как объекта TheDialog:

```
class APPLICATION: public CWinApp
{
public:
     BOOL InitInstance();
};

BOOL APPLICATION::InitInstance()
{
     MY_DIALOG  TheDialog( (LPTSTR) IDD_DIALOG1, NULL);
     TheDialog.DoModal( );
     return TRUE;
}          .
```

6. Откомпилировать и выполнить приложение.

2.12. Задания для самостоятельного выполнения

1*. Создать аналогичное приложение с запуском диалога по сообщению WM_PAINT, по нажатию клавиши клавиатуры и по нажатию клавиши мыши.

2. Создать приложение с диалоговым окном в качестве главного для вычисления квадратов вводимых значений. Результат выводить в отдельном диалоговом окне по нажатию соответствующей кнопки (например, Результат). После просмотра результата передавать управление исходному окну.

3. Создать приложение с диалоговым окном в качестве главного для вычисления кубов вводимых значений. Результат выводить в отдельном окне типа главного по нажатию соответствующей кнопки (например, Результат). После просмотра результата передавать управление исходному окну.

4*. Создать MFC-приложение на базе ТКП для ввода массива вещественных числовых значений и их суммирования.

В поле ввода диалогового окна задается число, а ввод его в приложение осуществляется по нажатии клавиши ОК или Enter. Суммирование и вывод результата в поле ввода

по кнопке СЛОЖИТЬ МАССИВ, а сброс системы для обработки следующего массива по кнопке СЛОЖИТЬ МАССИВ. Выход из приложения осуществляется по нажатии клавиши Esc или кнопки Cancel.

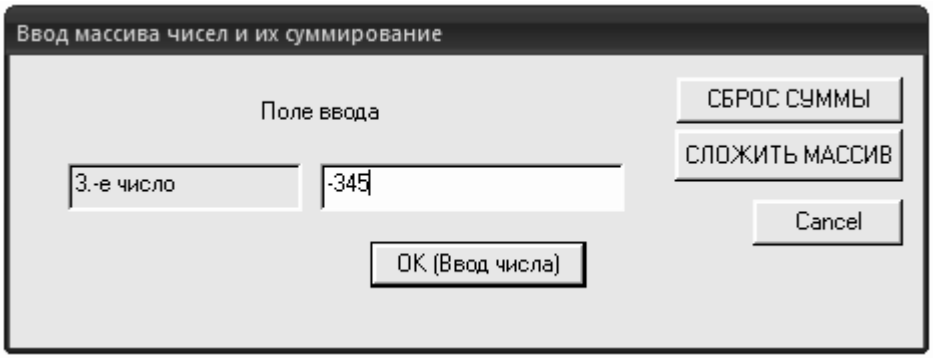


Рис. 35. Вид интерфейса

5*. Создать MFC-приложение на базе ТКП с оконным интерфейсом из диалогового окна в качестве главного. Диалоговое окно содержит кнопку OK и кнопку Cancel. При нажатии кнопки OK пользователю предлагается загрузить следующее диалоговое окно такого же типа (класса) поверх текущего. Это позволяет создавать много диалоговых окон. При этом управление передается каждому следующему окну.

При нажатии кнопки Cancel текущее окно исчезает, а управление автоматически передается его родителю – предыдущему окну.

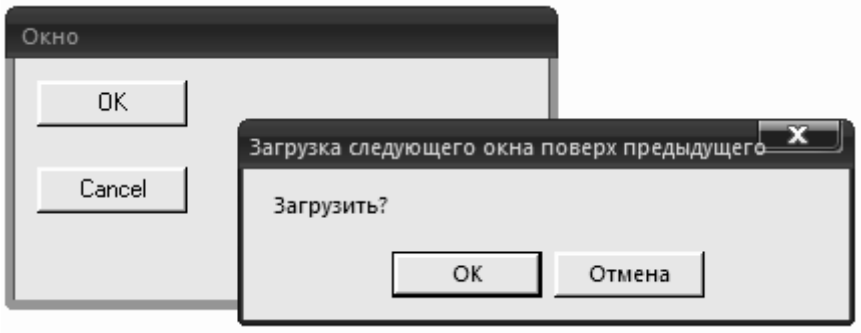


Рис. 36. Вид интерфейса

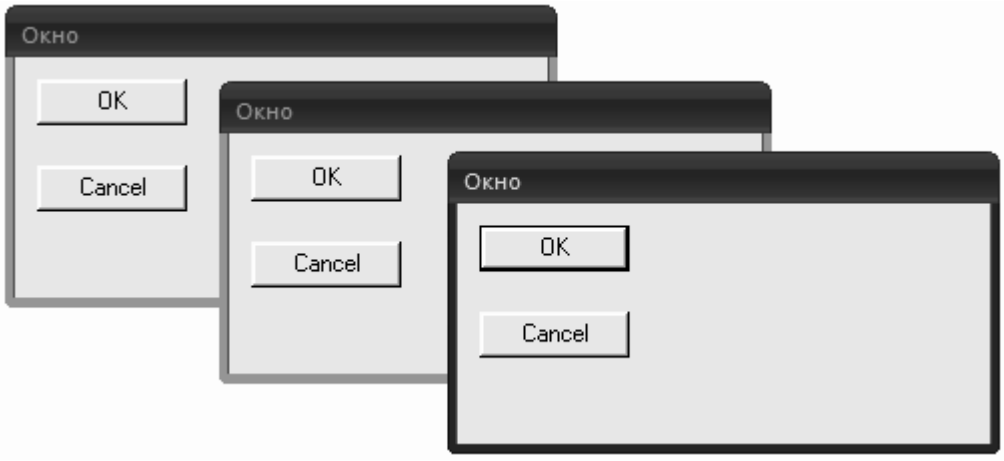


Рис. 37. Вид интерфейса

ПРИМЕЧАНИЕ: пункты, помеченные *, выполняются по указанию преподавателя.

ЛИТЕРАТУРА

1. Паппас К., Мюррей У. Visual C++. Руководство для профессионалов: 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., Мюррей У. Эффективная работа: Visual C++.NET. – СПб.: Питер, 2002. – 816 с.

Дополнительная литература

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

5. Г. Буч. 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., Брусенцев В.А. Методы и алгоритмы компьютерной графики в примерах на Visual C++. - СПб.: БХВ-Петербург, 2003. – 560 с.

9. Мюррей У., Паппас К. Создание переносимых приложений для Windows. – 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., Щупак Ю.А. C++. Объектно-ориентированное программирование: практикум. – СПб.: Питер, 2004. – 265 с.

ПРИЛОЖЕНИЕ 1. Стили окон и диалоговых окон

Таблица 2. Стили окон

Параметр стиля	Описание
WS_BORDER	создание окна с рамкой
WS_CAPTION	добавление к окну с рамкой заголовка
WS_CHILD	создание дочернего окна
WS_CHILDWINDOW	создание дочернего окна стиля WS_CHILD
WS_CLIPCHILDREN	при создании родительского окна запрещает его рисование в области, занятой любым дочерним окном
WS_CLIPSIBLINGS	(только со стилем WS_CHILD) не использовать при получении данным окном сообщения о перерисовке области, занятые другими дочерними окнами. Иначе разрешается рисовать в клиентской области другого дочернего окна
WS_DISABLED	создание первоначально неактивного окна
WS_DLGFRAME	создание окна с двойной рамкой без заголовка
WS_GROUP	(только в диалоговых окнах) стиль указывается для первого элемента управления в группе элементов, пользователь перемещается между ними с помощью клавиш-стрелок
WS_HSCROLL	создание окна с горизонтальной полосой прокрутки
WS_MAXIMIZE WS_MINIMIZE	создание окна максимального (минимального) размера
WS_MAXIMIZEBOX WS_MINIMIZEBOX	создание окна с кнопкой максимизации (минимизации)
WS_OVERLAPPED	создание перекрывающегося окна
WS_OVERLAPPED WINDOW	создание перекрывающегося окна на базе стилей WS_OVERLAPPED, WS_THICKFRAME и WS_SYSMENU
WS_POPUP	(не используется с окнами стиля WS_CHILD) создание временного окна
WS_ POPUPWINDOW	создание временного окна на базе стилей WS_BORDER, WS_POPUP и WS_SYSMENU
WS_SYSMENU	(только для окон с заголовком) создание окна с кнопкой вызова системного меню вместо стандартной кнопки, позволяющей закрыть окно при использовании этого стиля для дочернего окна
WS_TABSTOP	(только в диалоговых окнах) указывает на произвольное количество элементов управления (стиль WS_TABSTOP), между которыми можно перемещаться клавишей <TaЬ>
WS_THICKFRAME	создание окна с толстой рамкой, используемой для изменения размеров окна
WS_VISIBLE	создание окна, видимого сразу после создания
WS_VSCROLL	создание окна с вертикальной полосой прокрутки

Таблица 3. Стили диалоговых окон

Параметр стиля	Описание
DS_3DLOOK	использование трехмерных рамок для изображения элементов управления диалогового окна. Стиль требуется только для Windows NT 3.51
DS_ABSALIGN	в качестве координат диалогового окна вместо клиентских используются экранные
DS_CENTER	центрирование диалогового окна в рабочей области родительского окна. Иначе центрирование производится в рабочей области монитора в соответствии с системными настройками
DS_CENTERMOUSE	центрирование диалогового окна относительно курсора “мыши”
DS_CONTEXTHELP	включает значок вопроса в поле заголовка диалогового окна
DS_FIXEDSYS	вызывает использование в диалоговом окне SYSTEM_FIXED_FONT вместо SYSTEM_FONT (по умолчанию)
DS_MODALFRAME	создает диалоговое окно с модальной рамкой. Можно комбинировать с полем названия и системным меню, задаваемыми стилями WS_CAPTION и WS_SYSMENU

ПРИЛОЖЕНИЕ 2. Стили командных кнопок

Таблица 4. Стили кнопок

Параметр стиля	Описание
BS_BOTTOM	размещает текст вверху прямоугольника кнопки
BS_CENTER	центрирует текст по горизонтали в прямоугольнике кнопки
BS_DEFPUSHBUTTON	кнопка с жирной черной рамкой. Если она в составе диалогового окна, то пользователь может выбирать ее нажатием клавиши ENTER, даже если она не в фокусе ввода. Полезно для быстрого выбора наиболее используемой кнопки (или кнопки по умолчанию)
BS_LEFT	выравнивает текст влево в прямоугольнике кнопки
BS_PUSHBUTTON	командная кнопка, посылающая при “щелчке” пользователя сообщение WM_COMMAND окну-собственнику, где размещена эта кнопка
BS_RIGHT	выравнивает текст вправо в прямоугольнике кнопки
BS_TOP	размещает текст вверху прямоугольника кнопки
BS_VCENTER	центрирует текст по вертикали в прямоугольнике кнопки
BS_RADIOBUTTON	круглая кнопка, посылающая при “щелчке” пользователя сообщение WM_COMMAND окну-собственнику, где размещена эта кнопка. При повторном “щелчке” выбор отменяется
BS_CHECKBOX	прямоугольная кнопка, которую можно выбрать (переключатель)
BS_GROUPBOX	прямоугольная область, охватывающая группу кнопок

ПРИЛОЖЕНИЕ 3. Листинг MFC-приложения “простое диалоговое окно”

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;

//////////////////////////////////////////////////
class MY_DIALOG : public CDialog
{
public:
   MY_DIALOG( char *DialogName, CWnd *Owner ):
              CDialog(DialogName, Owner)  {  };
   DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(MY_DIALOG, CDialog)
END_MESSAGE_MAP()

//////////////////////////////////////////////////
class WINDOW : public CFrameWnd
{
public:
     WINDOW();
     afx_msg void OnPaint ( );
     DECLARE_MESSAGE_MAP()
};

WINDOW::WINDOW ( )
{
     Create( NULL,"ГЛАВНОЕ ОКНО" );
}

afx_msg void WINDOW::OnPaint  ()
{
     CPaintDC dc(this);
     MY_DIALOG  TheDialog ( ( LPTSTR ) IDD_DIALOG1, this);
     TheDialog.DoModal ( );
};

BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
    ON_WM_PAINT()
END_MESSAGE_MAP()
```

```
    ////////////////////////////////////////////////
    class APPLICATION: public CWinApp
    {
    public:
        BOOL InitInstance ( );
    };

    BOOL APPLICATION::InitInstance ( )
    {
        m_pMainWnd =  new WINDOW;
        m_pMainWnd -> ShowWindow ( m_nCmdShow );
        m_pMainWnd -> UpdateWindow  ();
        return TRUE;
    }

    ////////////////////////////////////////////////////////
APPLICATION  TheApplication;
```

ПРИЛОЖЕНИЕ 4. Листинг MFC-приложения с диалоговыми окнами, выводимыми до и в составе главного окна

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;

//////////////////////////////////////////////////////
class MY_DIALOG : public CDialog
{
public:
   MY_DIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName, Owner) { };
   DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(MY_DIALOG, CDialog)
END_MESSAGE_MAP()

//////////////////////////////////////////////////////
class WINDOW : public CFrameWnd
{
public:
        WINDOW();
        afx_msg void OnPaint();
        DECLARE_MESSAGE_MAP()
};

WINDOW::WINDOW()
{
        Create(NULL,"ГЛАВНОЕ ОКНО ");
        //или MY_DIALOG TheDialog((LPTSTR)IDD_DIALOG1 ,this);
        //или TheDialog.DoModal();
}

afx_msg void WINDOW::OnPaint()
{
        CPaintDC dc(this);
        MY_DIALOG TheDialog((LPTSTR)IDD_DIALOG1,this);
        TheDialog.DoModal();
};

BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
```

```
   ON_WM_PAINT()
END_MESSAGE_MAP()

//////////////////////////////////////////////////
class APPLICATION:public CWinApp
{
public:
        BOOL InitInstance();
};

BOOL APPLICATION::InitInstance()
{
        m_pMainWnd =  new WINDOW;
        //или без MY_DIALOG TheDialog((LPTSTR)IDD_DIALOG1 ,NULL);
        //или без TheDialog.DoModal();
        m_pMainWnd -> ShowWindow(m_nCmdShow);
        m_pMainWnd -> UpdateWindow();
        return TRUE;
}

//////////////////////////////////////////////////////////////
APPLICATION  TheApplication;
```

ПРИЛОЖЕНИЕ 5. Стили окон редактирования

Таблица 5. Стили окон

Параметр стиля	Описание
ES_CENTER	центрирует текст
ES _LEFT	выравнивает текст по левому краю
ES _RIGHT	выравнивает текст по правому краю
ES _LOWERCASE	при вводе символы преобразуются в нижний регистр
ES _UPPERCASE	при вводе символы преобразуются в верхний регистр
ES _MULTILINE	обеспечивает работу в режиме многострочного редактора. При комбинации с другими стилями возможна вертикальная и горизонтальная прокрутка текста
ES _PASSWORD	вводимый текст скрывается символами “*”
ES _READONLY	текст отображается без возможности ввода или редактирования

ПРИЛОЖЕНИЕ 6. Листинг MFC-приложения с диалоговым окном в качестве главного окна

```
#include <afxwin.h>
#include <iostream>
#include "resource.h"
using namespace std;

//////////////////////////////////////////////////////
class MY_DIALOG : public CDialog
{
public:
    MY_DIALOG(char *DialogName, CWnd *Owner): CDialog(DialogName, Owner) { };
    DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(MY_DIALOG, CDialog)
END_MESSAGE_MAP()

//////////////////////////////////////////////////////
class APPLICATION:public CWinApp
{
public:
        BOOL InitInstance();
};

BOOL APPLICATION::InitInstance()
{
        MY_DIALOG TheDialog((LPTSTR)IDD_DIALOG1,NULL);
        TheDialog.DoModal();
        return TRUE;
}

//////////////////////////////////////
APPLICATION  App;
```

ОГЛАВЛЕНИЕ

Учебное издание

Муравьев Геннадий Леонидович, Савицкий Юрий Викторович,
Хвещук Владимир Иванович

методическое пособие

“ ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ на базе библиотеки MFC”, Часть 2

Ответственный за выпуск: ***Г.Л.Муравьев***
Редактор ***Т.В. Строкач***
Технический редактор
Компьютерный набор и
верстка: ***Г.Л.Муравьев***

Издательская лицензия
Подписано в печать …. Формат
Бумага писч. Усл. изд. л. …Тираж
Заказ №
Отпечатано на ризографе УО “Брестский государственный технический университет”
224017, Брест, ул. Московская, 267
Полиграфическая лицензия …