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 1

БРЕСТ 2008

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 информатики и прикладной математики Брестского государственного университета им. А.С. Пушкина, к.т.н., доцент Козинский А..А.

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

ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ НА БАЗЕ БИБЛИОТЕКИ MFC. Г.Л. Муравьев, В.И. Хвещук, В.Ю. Савицкий, С.В. Мухов. – Брест.: БрГТУ, 2008. Часть 1. - 45 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

1. БИБЛИОТЕКА MFC. СРЕДА РАЗРАБОТКИ ПРИЛОЖЕНИЙ

1.1. Особенности применения технологий программирования в операционной системе Windows

Операционная система Windows базируется на понятии виртуальная машина, что обеспечивает независимость пользователя от специфики аппаратных средств системы и деталей реализации программных компонентов предоставляемого программного обеспечения. Доступ к системным ресурсам осуществляется через системные функции, образующие прикладной программный интерфейс – API (Application Programming Interface). Поддержку аппаратно-независимой графики обеспечивает подмножество функций API – интерфейс графического устройства GDI (Graphics Device Interface). Функции GDI дают программисту средства создания и реализации оконных интерфейсов. В настоящее время преимущественно используется 32-разрядная версия API, называемая Win32 и являющаяся надмножеством версии Win16. Существенный момент - Win32 поддерживает 32-разрядную линейную адресацию памяти взамен 16-разрядной сегментированной модели. Соответственно управление памятью системы производится в защищенном режиме на базе виртуальной – линейной, плоской модели памяти в 4 Гб с использованием механизмов страничного скроллинга. Windows обеспечивает мультипрограммность работы системы, ее многозадачность. Поддерживается два типа многозадачности: многозадачность, основанная на процессах и основанная на потоках. Соответственно под его управлением одновременно может выполняться несколько приложений (процессов), а в рамках приложений могут быть определены параллельно выполняемые и при необходимости синхронизируемые участки (потоки, “направления протекания процесса”). Снижение объемов приложений, их живучесть и мобильность также обеспечиваются использованием библиотек динамической загрузки - DLL (Dynamic Link Libraries, или), которые загружаются в память только в момент, когда к ним происходит обращение, т.е. при выполнении программы.

Оконные приложения функционируют под управлением операционной системы Windows и взаимодействуют с ресурсами системы, с внешним миром посредством механизма пересылки сообщений через Windows о происходящих событиях. Приложение может реагировать на полученное сообщение – обрабатывать его, выполняя определенные в приложении программистом действия. При этом в промежутке между обработкой сообщений приложение бездействует. Оконные приложения используют привычный пользователю, типизированный Windows-интерфейс на базе системы окон. Используются окна с клиентской областью, диалоговые окна различных типов, включая стандартные. Как только окно активизируется (получает фокус), то именно с ним ассоциируются приходящие сообщения. Соответственно их обработкой занимаются специальные функции-обработчики сообщений, ассоциированные с окном.

При разработке приложений, исполняемых в операционной системе Windows, используют два основных подхода. *Первый подход* - процедурная разработка, базирующаяся на алгоритмической декомпозиции предметной области и принципах структурной разработки программ. В Windows это стиль низкоуровневого программирования с непосредственным использованием функций операционной системы. *Второй подход* – объектно-ориентированная разработка (проектирование, программирование, тестирование), базирующаяся на объектной декомпозиции предметной области и принципах построения объектно-ориентированной модели ПО. В Windows это стиль высокоуровневого программирования с использованием библиотек готовых классов. Например, библиотеки MFC (Microsoft Foundation Class Library), применяемой для создания Windows-приложений с графическим интерфейсом. Результатом применения этого стиля в сис-

теме программирования Visual Studio являются оконные MFC-приложения как “статичного” так и “динамического” типов. Интерфейсные функции приложений реализуются программированием, основанным на ресурсах. Типовые интерфейсные ресурсы - растровые изображении, пиктограммы, описания меню, диалоговых окон и т.п. Описания ресурсов в процессе реализации интерфейса приложения редактируются специальными инструментальными средствами – редакторами ресурсов, обеспечивающими режим WYSIWYG. Описания ресурсов хранятся в файлах ресурсов.

1.2. Характеристика библиотеки MFC

Чтобы облегчить программирование, особенно выполнение стандартных задач, к которым относится и создание многооконных интерфейсов, системы программирования, компиляторы используют специальные библиотеки. Библиотека MFC ориентирована на существенное упрощение работы с прикладным программным интерфейсом (API) Windows, который включает сотни разрозненных API функций, за счет их структуризации. В MFC функции API объединены (путем инкапсуляции, в виде методов) в логически организованное множество классов, что позволяет использовать при программировании средства более высокого уровня, чем обычные вызовы функций. Допускается и смешанное использование. API функции можно использовать в MFC-приложении, написанном на языке C++. Часть классов MFC (классы общего назначения - файлы, строки, время, исключительные ситуации и т.д.) можно использовать как в DOS- так и в Windows-приложениях. Состав папок системы программирования Visual Studio C++ и место библиотеки MFC в составе Visual Studio C++ иллюстрируется на рисунке 1.

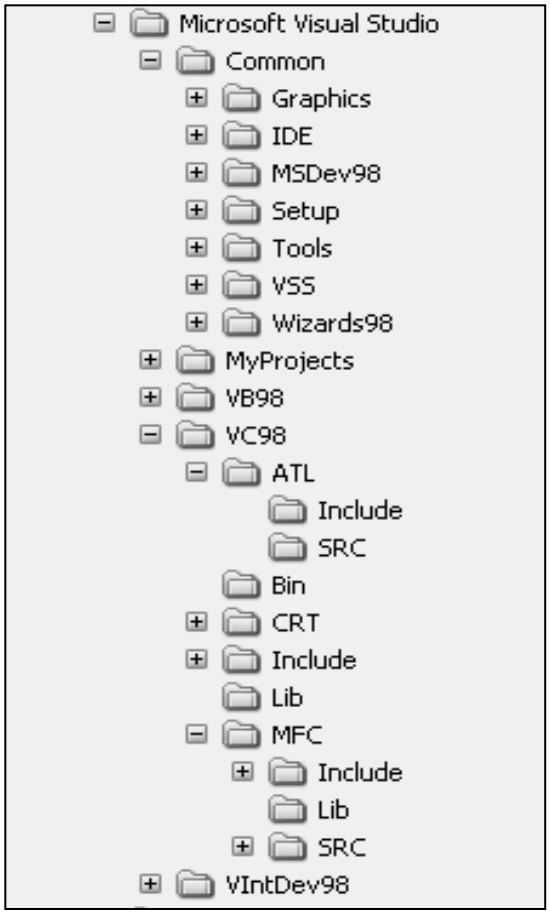


Рис. 1. Состав папок системы Visual Studio

Таким образом, объектно-ориентированная библиотека MFC – это базовый набор (библиотека) классов, написанных на языке C++ и предназначенных для упрощения и ускорения процесса программирования для ОС Windows. Библиотека включает многоуровневую иерархию классов (около 200 членов). Они поддерживают высокоуровневый стиль разработки Windows-приложений на базе объектно-ориентированного подхода. Практически классы MFC представляют собой каркас, на основе которого можно писать программы для ОС Windows.

Библиотека MFC реализует принцип многократного использования одного и того же кода, позволяя просто наследовать типовые составляющие приложений (наследовать классы). Интерфейс, обеспечиваемый библиотекой, практически независим от конкретных деталей реализации, поэтому программы, написанные на основе MFC, легко

адаптируются к новым версиям Windows. Это существенно упрощает программирование, так можно использовать готовые классы MFC для создания объектов в пользовательских приложениях, можно дорабатывать классы MFC, наследуя их и создавая пользовательские классы и иерархии классов.

Технологически процесс программирования в MFC реализуется как “каркасное” программирование. Библиотека MFC, в том числе, обеспечивает технологическую, программную поддержку каркасов архитектуры приложений типа “ОДИНОЧНЫЙ_ДОКУМЕНТ” с интерфейсом SDI (Single Document Interface) и типа “ДОКУМЕНТ-ВИД” с интерфейсом MDI (Multiple Document Interface), образующих ядро программирования в MFC. Первый каркас использует класс ОКНО_С_РАМКОЙ (CFrameWnd) для организации всех функций по обработке данных (документов), включая их хранение, визуализацию. Последний каркас является базовым для построения интерфейсов многодокументных оконных приложений, а соответствующий каркас и архитектура образуют ядро программирования в MFC. Здесь объект обработки разделяется на класс ВИД (CView) и класс ДОКУМЕНТ (CDocument). Соответственно приложение может состоять из одного или нескольких документов - объектов, классы которых являются производными от класса ДОКУМЕНТ. С каждым из документов может быть связан один или несколько обликов - объектов классов, производных от класса ВИД. Класс ВИД обеспечивает, определяет оконный вид, облик, представление документа, а класс ДОКУМЕНТ позволяет представлять более абстрактные объекты, чем документ, понимаемый как объект обработки текстового процессора.

К числу дополняющих библиотек можно отнести: - библиотеку стандартных шаблонов STL (Standard Template Library); - библиотеку шаблонов ActiveX ATL (Microsoft Active Template Library), предназначенную для создания небольших COM-объектов и элементов управления ActiveX; - объектно-ориентированную библиотеку WFC (Windows Foundation Classes) для Windows-приложений на языке Java и др. Библиотека MFC реализована как в виде обычной статично подключаемой библиотеки (Static Library) так и в виде динамически подключаемых библиотек DLL (Shared Dynamic Link Libraries). Считается, что компилятор в оптимизирующем режиме создает MFC-приложения, несущественно отличающиеся по своим характеристикам от аналогичных приложений на языке C++, созданных с непосредственным использованием API функций. Но при существенно меньшем размере кода (в три и более раза). При этом, что немаловажно, большинство частных деталей скрыто от программиста и не требует знания.

Библиотека MFC содержит типовые средства, необходимые для функционирования Windows-приложений. Это, в частности, средства управления сообщениями, окнами, диалоговыми окнами, ресурсами типа меню, кисти, перья, растровые изображения и т.п., средства работы с документами и многодокументным интерфейсом (MDI). Библиотека MFC поддерживает механизм обработки исключительных ситуаций, обеспечивая восстанавливаемость приложений после появления ошибок. MFC также обеспечивает механизм динамического определения типов объектов, что позволяет производить проверку и преобразование типов динамически в процессе выполнения программы.

1.3. Характеристика классов библиотеки MFC

Большинство классов библиотеки MFC являются производными от базового класса CObject ("объект") и предназначено для построения компонентов приложения. Подробную информацию о классах, а также полный вариант диаграммы классов MFC можно найти в справочной информации Visual Studio – MSDN. Названия всех классов и шаблонов классов библиотеки MFC начинаются с заглавной буквы C. Имена переменных, вхо-

дящих в класс MFC начинаются с префикса m_. Названия методов классов, как правило, начинаются с заглавной буквы. Названия служебных (глобальных) функций библиотеки MFC начинаются с префикса Afx, например AfxMessageBox. Фрагмент соответствующей диаграммы классов приведен на рисунке 2 ниже. Здесь CCmdTarget является основой для формирования архитектуры приложения (двух ее компонентов – интерфейсного класса и класса, управляющего движением и обработкой сообщений). Соответственно у CCmdTarget есть производные классы: - CWinThread позволяет создавать поток приложения, а его производный класс CWinApp - основа приложения; - класс CWnd является базовым для создания разнообразных окон, в том числе, окон с рамкой класса CFrameWnd, диалоговых окон класса CDialog, специализированных окон ЭУ, обликов класса CView; - класс CDocument служит для организации работы с документами.

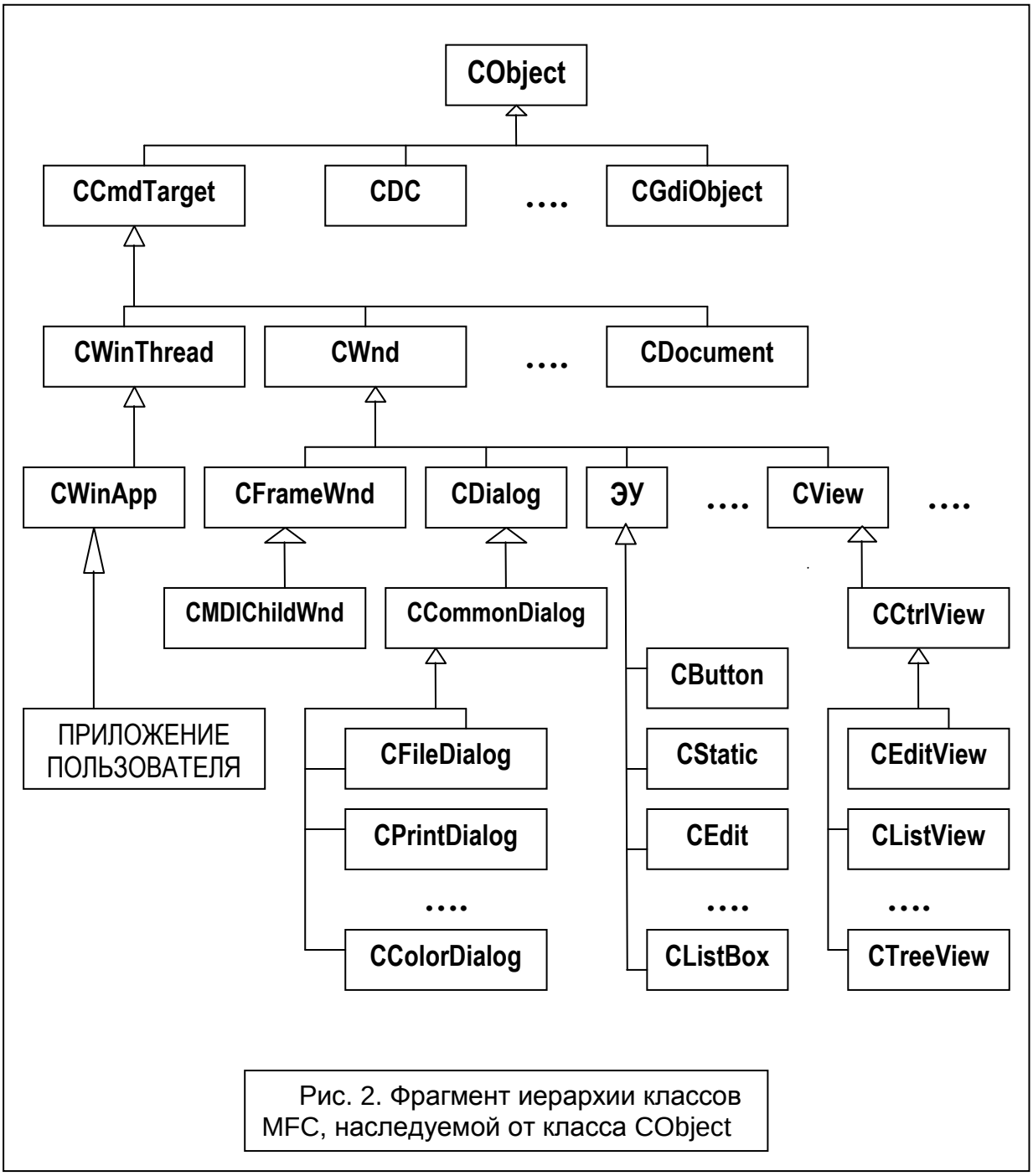


Рис. 2. Фрагмент иерархии классов MFC, наследуемой от класса CObject

С типовым MFC-приложением связывается определяющий его на верхнем уровне объект, принадлежащий классу, производному от класса CWinApp. 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. Рамочные окна типа главного окна, масштабируемые и перекрываемые, с меню и без, с клиентской областью, предназначенной для вывода текстовой и графической информации на экран, производятся от класса CFrameWnd. Диалоговые окна интерфейса, являющиеся подложкой для размещения и компоновки различных ЭУ, производятся от класса CDialog. Каждый ЭУ, представляющий собой специфическое окно, производится от соответствующего класса семейства классов элементов управления.

Для приложений с многодокументной архитектурой документы представляются как объекты, производные от класса CDocument. С каждым из документов связаны объекты, производные от класса CView (класс "облик ") и определяющие облик документа.

1.4. Класс CWnd

Класс CWnd – базовый для создания окон разных типов в MFC-приложениях. Окна MFC-приложений (CWnd-объекты) отличаются организацией от классических Windows-окон, хотя и тесно связаны с ними функционально. Класс CWnd, механизм обработки событий (карта событий) позволили скрыть функцию, обработчик главного окна. Сообщения, автоматически маршрутизируясь на основе карты сообщений, направляются на обработку в соответствующие функции-обработчики класса CWnd. Класс CWnd позволяет создавать производные пользовательские классы и на их основе объекты – пользовательские окна. При этом программист в своих производных классах переописывает (override) обработчики, определенные в классе CWnd, чтобы обрабатывать свои сообщения нужным образом. Такие окна создаются: 1) вызовом конструктора CWnd() для создания CWnd-объекта; 2) вызовом метода Create для создания пользовательского окна, связанного с этим CWnd-объектом. Дескриптор созданного окна хранится в атрибуте m_hWnd (HWND CWnd::m_hWnd).

Класс CWnd содержит методы получения контекста BeginPaint, EndPaint, перерисовки клиентской области Invalidate (перерисовка всей клиентской области), InvalidateRgn, InvalidateRect (перерисовка части клиентской области в пределах заданной прямоугольной области путем ее добавления к текущей области обновления), методы визуализации и обновления окна ShowWindow, UpdateWindow, методы управления таймером SetTimer, KillTimer. Отдельную группу составляют обработчики событий. В том числе, обработчики сообщений инициализации (например, OnInitMenu вызываемый при активизации меню), обработчики системных событий (например, OnSysCommand, вызываемый, когда пользователь выбирает команды системного меню, использует кнопки минимизации-максимизации размеров окна), обработчики общих событий и т.д.

Для создания дочернего (child) окна и связывания его (прикрепления) с CWnd-объектом применяется метод Create, который возвращает ненулевое значение при успешном завершении, и имеет прототип

```
virtual BOOL CWnd::Create
                    (LPCTSTR lpszClassName,
                     LPCTSTR lpszWindowName,
                     DWORD dwStyle,
```

const RECT &*rect*,
CWnd ***pParentWnd*,
UINT *nID*,
CCreateContext * *Context* = NULL**)** .

Параметры метода:

- lpszClassName – указатель на строку С (с нуль-символом), которая содержит имя класса окна. Этим именем может быть любое имя, зарегистрированное с помощью глобальной функции AfxRegisterWndClass или API функцией RegisterClass. Значение NULL позволяет использовать типовое MFC-окно;
- lpszWindowName - указатель на строку С (с нуль-символом), которая содержит заголовок окна;
- dwStyle – специфицирует атрибуты стиля окна;
- rect - специфицирует размер и положение окна;
- pParentWnd – указатель на родительское окно;
- nID – ID дочернего (child) окна;
- pContext – контекст окна.

ПРИМЕР. Динамическое создание ЭУ типа static в окне класса CMyDlg, путем запуска обработчика окна с именем CMyDlg::OnCreateStatic(). В обработчике использован метод CWnd::Create (вместо метода CStatic::Create)

```
void CMyDlg::OnCreateStatic()
{
    CWnd   *pWnd = new CWnd;
    pWnd -> Create(_T("STATIC"),"Hi",WS_CHILD | WS_VISIBLE,CRect(0,0,20, 20),
                   this, 1234);
}  .
```

1.5. Класс CFrameWnd

Класс CFrameWnd предоставляет возможности создания объектов-окон с рамками типа overlapped или pop-up, обеспечивающими одно документный интерфейс (SDI). Чтобы создать такое пользовательское окно надо: - создать производный класс; - добавить члены-данные для хранения специфических данных; - настроить карту сообщений окна; - описать обработчики сообщений.

Непосредственное создание, инициализация окон производится методом CFrameWnd::Create, который возвращает ненулевое значение при успешном завершении, и имеет прототип (похожий на прототип метода CWnd::Create)

BOOL Create (LPCTSTR *lpszClassName*,
LPCTSTR *lpszWindowName*,
DWORD *dwStyle* = WS_OVERLAPPEDWINDOW,
const RECT& *rect* = rectDefault,
CWnd* *pParentWnd* = NULL,
LPCTSTR *lpszMenuName* = NULL,
DWORD *dwExStyle* = 0,
CCreateContext* *pContext* = NULL **)** .

Здесь параметры метода:

- dwStyle – специфицирует атрибуты стиля окна (см. ПРИЛОЖЕНИЕ 1) и по умолчанию (значение WS_OVERLAPPEDWINDOW) задает создание перекрывающегося масштабируемого окна с системным меню и системными кнопками;

- rect - специфицирует размер и положение окна, которые по умолчанию (значение rectDefault - свойство класса CFrameWnd::rectDefault типа CRect) определяются ОС Windows автоматически;

- lpszMenuName – определяет имя ресурса-меню, используемого окном (по умолчанию NULL). Если ресурс-меню идентифицируется числовым значением ID, то можно использовать функцию MAKEINTRESOURCE для получения строкового эквивалента;

- pContext – указатель на структуру типа CCreateContext (по умолчанию NULL), которая используется системой, когда создаются масштабируемые окна или объекты класса ВИД, ассоциируемые с объектами класса ДОКУМЕНТ.

1.6. Сообщения. Обработчики сообщений

"Общение" MFC-приложения и его окружения (внешней среды) производится через систему сообщений. Сообщения соответствуют происходящим событиям, связаны с конкретным источником (например, сообщения "мыши", клавиатуры, диалогового окна, списка, кнопки и т.п.), предполагают реакцию приложения (обработку сообщений) путем запуска функций-обработчиков сообщений. Конкретный алгоритм обработки сообщений, как правило, определяется программистом.

В качестве внешней среды приложения выступают:

– операционная система, через которую собственно реализуется обмен сообщениями;

- пользователь, который оказывает воздействие на приложение (посылая сообщения) через устройства ввода ("мышь", клавиатуру и т.п.) и к которому направлен поток информации (на устройства вывода – экран, принтер и т.п.), являющийся результатом обработки сообщений;

- другие приложения, которые через ОС посылают-принимают сообщения (кроме этого, возможно и классическое взаимодействие приложений, например, через общую информацию, файлы, "почтовые ящики" и т.д.);

- устройства, в том числе удаленные.

Основные, типовые объекты, средства, обеспечивающие общение пользователя с приложением, – это графические ресурсы приложения, образующие видимый (визуальный) интерфейс на экране дисплея. В Windows это окна (окна с клиентской областью, диалоговые, окна сообщений, модальные и немодальные окна, специализированные окна - элементы управления), в том числе, встраиваемые в приложение редактором ресурсов путем редактирования программистом соответствующих шаблонов, и встраиваемые программно как "самостоятельные" окна, создаваемые create. Реализация общения с целью ввода-вывода информации предполагает организацию в приложении связи (канала связи) ПРИЛОЖЕНИЕ-РАБОЧЕЕ_ОКНО.

Таким образом, приложение может как посылать сообщения (в том числе и самому себе), требуя определенной реакции от внешней среды (ОС, других приложений), так и принимать сообщения (через ОС от приложений, от самой ОС), реагируя соответствующим образом путем запуска функций-обработчиков сообщений. Это могут быть как готовые, типовые обработчики (например, обработчик OnOk() сообщения "нажатие кнопки - ОК" диалогового окна), которые программист может тем не менее переопределить. Так и "пустые" обработчики сообщений со стандартными интерфейсами, функционирование которых определяется программистом.

В библиотеке MFC для обработки сообщений применяются аналогичные используемым в Windows-приложениях (по назначению, по наименованию и схожие по сигнатуре) функции-обработчики, являющиеся членами классов MFC (например, класса CWnd). В них отсутствует первый параметр – дескриптор приложения HWND hwnd, могут присутствовать дополнительные уточняющие параметры. Типовой прототип такой afx-функции

afx_msg void ИМЯ_ОБРАБОТЧИКА (*ПараметрыСообщения*) .

Например, прототип обработчика события - нажатие клавиши клавиатуры (сообщение WM_CHAR), *ПараметрыСообщения* - ASCII-код нажатой клавиши, число повторных нажатий символа из-за удерживания клавиши, сопутствующие события (нажатые клавиши)

afx_msg void OnChar (UINT ch, UINT count,UINT flags)

Ниже приведен пример прототипа пользовательского обработчика события – нажатие кнопки диалогового окна класса MY_DIALOG (сообщение WM_COMMAND),

afx_msg void OnButtonOK() .

Сам обработчик описывается как

afx_msg void MY_DIALOG::OnButtonOK() {…}.

Прототипы некоторых функций-обработчиков сообщений библиотеки MFC приведены в ПРИЛОЖЕНИИ 2.

1.7. Характеристика обработчиков типовых сообщений

1. Сообщение WM_CHAR вызывается каждый раз после нажатия несистемной клавиши (перед методом OnKeyUp и после метода OnKeyDown), содержит код клавиши (нажатой или отжатой) и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnChar(UINT *ASCII_КодКлавиши***, UINT** *ЧислоПовторов***, UINT** *ОписаниеСитуации***)** ,

где параметр *ЧислоПовторов* – указывает число повторных нажатий символа из-за удерживания клавиши;

параметр *ОписаниеСитуации* (флаги) содержит скэн-код клавиши, предыдущее состояние клавиши, описывает сопутствующие события (нажатые клавиши) (см. таблицу ниже).

Таблица 1. Значения флагов

Флаг	Значение
16-23	скэн-код клавиши, зависящий от конкретной аппаратуры
24	1, если клавиша является функциональной (на расширенной клавиатуре, например, правосторонние клавиши ALT, CTRL)
29	1, если была нажата и клавиша ALT (иначе 0)
30	1, если клавиша нажата до посылки сообщения (описывает предшествующее состояние клавиатуры)
31	1, если клавиша отпускается; 0, если удерживается нажатой (описывает переходное состояние клавиатуры)

2. Сообщение WM_COMMAND вызывается каждый раз, когда пользователь выбирает элемент управления (например, “мышью” выбирает конечный пункт меню, нажимает

кнопку и т.п.). Для включения реакции приложения на происшедшее событие используется макрокоманда

ON_COMMAND(*ID_ЭлементаУправления*, *ИмяОбработчика***) ,**

где параметр *ID_ЭлементаУправления* задает идентификатор выбранного ресурса интерфейса (например, пункта меню, кнопки и т.п.), а *ИмяОбработчика* определяет вызываемую функцию.

3. Сообщение WM_DESTROY вызывается каждый раз, когда CWnd-объект удаляется с экрана, и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnDestroy( ).

4. Сообщение типа WM_KILLFOCUS вызывается каждый раз - сразу же перед потерей фокуса ввода (the input focus) и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnKillFocus(CWnd * *УказательНовогоОкна* **)** ,

где параметр сообщает *УказательНовогоОкна*, которое принимает фокуса ввода (может быть NULL).

5. Сообщение типа WM_LBUTTONDOWN вызывается каждый раз после нажатия соответствующей клавиши “мыши” и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnLButtonDown(UINT *ОписаниеСитуации*, **CPoint** *Координаты Курсора* **)** ,

где параметр *КоординатыКурсора* – определяет *КоординатуКурсора.x* и *КоординатуКурсора.y* в момент нажатия *клавиши. А параметр ОписаниеСитуации (флаги),* описывает, была ли попутно нажата и удерживалась клавиша клавиатуры или “мыши” (например, MK_CONTROL - была нажата клавиша CTRL; MK_SHIFT - нажата клавиша SHIFT; MK_LBUTTON, MK_MBUTTON, MK_RBUTTON - удерживалась соответственно левая, средняя или правая кнопка “мыши”).

Аналогичные прототипы у обработчиков сообщений типа WM_RBUTTONDOWN, WM_LBUTTONDBLCLK, WM_RBUTTONDBLCLK - afx_msg void OnRButtonDown(…), afx_msg void OnLButtonDblClk(…), afx_msg void OnRButtonDblClk(…). Последние два сообщения принимают только окна со стилем CS_DBLCLKS (заданным в параметре style структуры WNDCLASS).

6. Сообщение WM_TIMER вызывается каждый раз после истечения каждого интервала времени, указанного в методе SetTimer, используемом для установки таймера, и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnTimer(UINT *ИдентификаторТаймера* **)** ,

где параметр идентифицирует сработавший таймер.

7. Сообщение WM_PAINT вызывается каждый раз после выполнения методов UpdateWindow, RedrawWindow, вызывается каждый раз, когда требуется перерисовка-обновление рабочей, клиентской области экрана и т.п. и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnPaint( ) .

8. Сообщение WM_SIZE вызывается каждый раз, после того как изменились размеры окна приложения, и обрабатывается функцией-обработчиком с прототипом

afx_msg void OnSize(UINT *ОписаниеСитуации*, **int** *Ширина*, **int** *Высота* **) ,**

где параметр *ОписаниеСитуации* указывает причину сообщения (например, SIZE_MAXIMIZED, SIZE_MINIMIZED - окно было максимизировано или минимизировано, SIZE_RESTORED - размеры окна произвольно изменил пользователь и т.д.); параметры *Ширина* и *Высота* дают новые размеры клиентской области окна.

1.8. Контекст устройства (КУ)

Для управления устройствами (например, принтером, дисплеем) в операционной системе Windows реализована концепция аппаратно-независимого программирования без прямого доступа к устройствам. Доступ осуществляется через специальный однотипный (по функциональному составу) интерфейс, реализуемый функциями API, при этом конкретные устройства представляются их виртуальными, графическими аналогами. В библиотеке MFC указанное поддерживается с помощью атрибутов класса CDC. Класс содержит целый ряд свойств, называемых атрибутами контекста устройства (содержащих данные о графическом устройстве вывода, задающими, например, цвет линии и т.п.). Кроме этого, класс CDC содержит все методы для построения изображения в окне.

Для осуществления в приложении операций по вводу-выводу информации необходимо организовать связь (канал связи) ПРИЛОЖЕНИЕ—РАБОЧЕЕ_ОКНО (также как при проведении файловых операций ввода-вывода необходимо открыть канал связи ПРИЛОЖЕНИЕ—ФАЙЛ). По терминологии Windows это означает, что приложению требуется контекст устройства (КУ). С информационной точки зрения контекст устройства - специальная служебная структура, которая предварительно создается, инициализируется (заполняется). Для этого используется класс CDC и производные классы CClientDC, CPaintDC, определенные в библиотеке MFC, а при реализации ввода-вывода применяются их методы.

Получение контекста устройства производится путем создания экземпляра класса CClientDC либо экземпляра класса CPaintDC. В первом случае используется конструктор

CClientDC (*УказательНаРабочееОкноПриложения*)

или

CClientDC (CWnd *Window).

Здесь роль указателя на класс CWnd - РабочееОкно обычно выполняет указатель по умолчанию this (т.е. *ЭтоРабочееОкно*). Например, команда

CClientDC TheDC (this);

позволяет создать (получить) КУ как экземпляр TheDC класса CClientDC, инициализированный значением this.

Во втором случае, который применяется при получении КУ в обработчике сообщения WM_PAINT, используется конструктор

CPaintDC (*УказательНаРабочееОкноПриложения*)

или

CPaintDC TheDC (CWnd *Window).

Например,

```
OnPaint()
{
    CPaintDC  TheDC (this);
…
}  .
```

Ниже на рисунке 3 иллюстрируется получение КУ в обработчике сообщения “нажатие левой клавиши мыши” и вызов справки по методам КУ (здесь объект TheDC).

1.9. Перерисовка

При работе с интерфейсным элементом типа окно возникают проблемы перерисовки (обновления) окна. Они связанны как с потерей и соответственно необходимо-

стью восстановления облика окна, которое было свернуто, закрыто, перекрыто и т.п. в ходе работы приложения так и с необходимостью обновления облика окна при изменении выводимой информации. Говорят, что изображение в окне может быть “повреждено” - на время перекрыто другим окном. В Windows нет автоматического сохранения текущего облика с последующим автоматическим восстановлением окна, нет автоматического перерисовки. За исключением ряда специфических манипуляций, когда оно сохраняется и восстанавливается в прежнем виде – например, при пе-

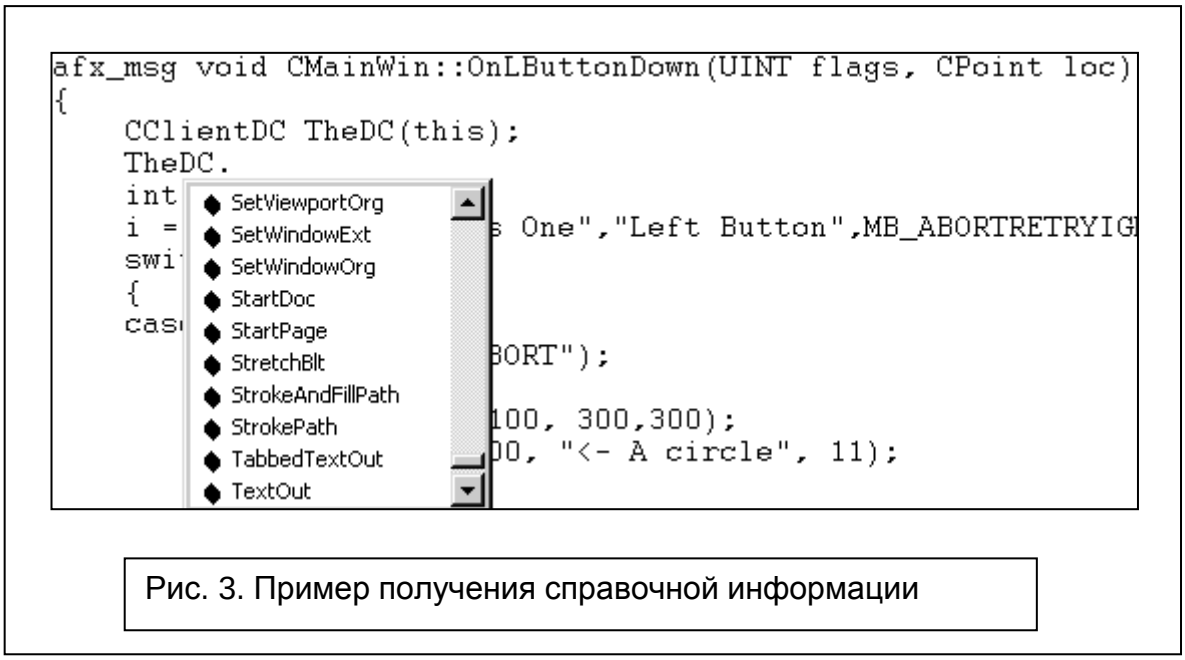


Рис. 3. Пример получения справочной информации

ремещении окна, масштабировании. Если область перекрытия незначительная, например, окном меню, то перекрытая часть восстанавливается системой Windows. Это выполнено сознательно, чтобы не работать с избыточной информацией и не замедлять обслуживание параллельно исполняемых приложений. Решение указанных проблем возложено на программиста (для восстановления при необходимости облика окна он может сам сохранять нужные данные либо вычислять их по известным ему протоколам или алгоритмам работы приложения, использовать средства MFC по организации “виртуальных окон” и т.д.). В большинстве же случаев Windows лишь сообщает приложению посылкой сообщения о необходимости восстановить окно. Для этого система Windows фиксирует информацию о поврежденной области (invalid region) или о поврежденный прямоугольнике области усечения (invalid rectangle), т.е. минимальном прямоугольнике, покрывающем поврежденную область. Описание поврежденного прямоугольника и поврежденной области можно получить методами CWnd ::GetUpdateRect и CWnd ::GetUpdateRgn.

Для восстановления поврежденного окна используются обработчики типа OnPaint,, OnDraw. Для запуска этих обработчиков можно генерировать программно соответствующие сообщения типа WM_PAINT, передавая указатели на контекст устройства, используя который, можно получить параметры поврежденной области или прямоугольника. Для этого используются методы CWnd ::Invalidate, CWnd ::InvalidateRect и CWnd ::InvalidateRgn, которые позволяют объявить соответственно клиентскую область, некоторый прямоугольник и область окна поврежденными и послать сообщение типа WM_PAINT в очередь приложения. Методы CWnd ::ValidateRect и CWnd ::ValidateRgn позволяют отменить объявление некоторого прямоугольника или облас-

ти поврежденными, что ведет к корректировке текущего поврежденного прямоугольника.

Метод InvalidateRect позволяет объявить часть клиентской области (в указанном прямоугольнике) “недействительной”, “поврежденной”, требующей перерисовки и добавить ее к текущей области, требующей обновления. Это вынуждает Windows послать приложению сообщение WM_PAINT. Указанная область вместе с другими, требующими обновления, может быть перерисована по получении нового сообщения WM_PAINT. Метод имеет прототип

ПЕРЕРИСОВАТЬ_ОКНО (**LPCRECT** *УказательОбластиПерерисовкиВнутриОкна*, **BOOL** *СтеретьФонОкна* = TRUE)

или

void CWnd:: **InvalidateRect (LPCRECT** *lpRect*, **BOOL** *bErase* = *TRUE* **)** ,

где параметр lpRect – указатель на CRect-объект или RECT-структуру, которая идентифицирует прямоугольник, требующий перерисовки и добавляемый к области обновления (если NULL, то обновляется вся клиентская область);

- bErase – флаг перерисовки, который определяет необходимость перерисовки фона области (при этом прежнее изображение стирается). Если новое изображение перекрывает старое, то фон можно не стирать.

ПРИМЕР использования метода представлен ниже. Здесь прямоугольные области, описанные переменными rctRect1 и rctRect2, требуют перерисовки. Причем фон первой области стирается, а второй – нет.

```
RECT rctRect1;
rctA.left = 20;
rctA.top = 20;
rctA.right  = 80;
rctA.bottom = 80;
RECT rctRect2= {100,100,300,400};
InvalidateRect ( LPCRECT rctRect1);
InvalidateRect ( LPCRECT rctRect2, FALSE );   .
```

Метод Invalidate объявляет всю клиентскую область CWnd-объекта “недействительной” и имеет прототип

void CWnd:: **Invalidate(BOOL** *bErase* = TRUE **)**.

Метод InvalidateRgn позволяет объявить указанную область (часть клиентского окна) “недействительной”, “поврежденной”, требующей перерисовки, и имеет прототип

void CWnd:: **InvalidateRgn(CRgn*** *pRgn***, BOOL** *bErase* = TRUE **)** ,

где параметр pRgn – указатель на CRgn-объект, который идентифицирует область (регион), добавляемый к области обновления (если NULL, то это вся клиентская область).

1.10. Характеристика среды Microsoft Developer Studio

Интегрированная среда разработки фирмы Microsoft (Microsoft Developer Studio) - универсальная среда, поддерживающая различные языковые средства разработки, например Visual C++, Visual J++ и т.д. (здесь рассматривается Visual C++).

Назначение среды – поддержка процесса создания Windows-приложений с использованием библиотеки базовых классов MFC и других библиотек (как статических так и динамических) как “вручную” так и с использованием средств автоматизации. К числу

средств автоматизации можно отнести мастера приложений (генераторы каркасов приложений нужной архитектуры для последующей доработки), мастер классов ClassWizard (для создания шаблонов новых производных классов и их методов), редакторы ресурсов (для создания меню, диалоговых окон, элементов управления и т.п.).

При работе с MFC-приложениями можно использовать мастера приложений MFC AppWizard (exe), MFC AppWizard (dll), позволяющие создавать каркасы разных типов статических и динамических (dll) проектов. С их помощью можно создать проект Windows-приложения с однодокументным, многодокументным или диалоговым интерфейсом. Однодокументное приложение предоставляет пользователю возможность работы в любой момент времени только с одним файлом. Многодокументное приложение предоставляет пользователю возможность одновременной загрузки и обработки нескольких документов (каждый в собственном окне). Пользовательский интерфейс диалогового приложения базируется на диалоговом окне, используемом в роли главного окна приложения. Мастер ClassWizard позволяет добавлять в приложение новые классы, созданные на основе базовых классов (как правило, классов, унаследованных от класса CCmdTarget или класса CRecordset), автоматизировать добавление и описание новых методов (функций) классов, методов, обрабатывающих сообщения ресурсов, меню, ЭУ и т.д. Средства AppWizard и ClassWizard для указания принадлежности того или иного объекта (функции, переменной, ключевого слова и т.д.) к библиотеке MFC используют сокращение AFX (от Application FrameworkX - основа приложения, его внутреннее устройство). Например, в процессе генерации они размещают в исходном тексте приложения комментарии следующего вида:

***//{{AFX**_ИдентификаторСекции ... ***//}}AFX**_ИдентификаторСекции*

Эти комментарии образуют секции (блоки) кода программы, которые управляются только средствами MFC (AppWizard, ClassWizard) и не должны меняться пользователем вручную. Идентификаторы некоторых секций приведены в таблице ниже.

Таблица 2. AFX-Секции

Секция	Назначение
//{{AFX_DATA //}}AFX_DATA	описание (объявление) элементов данных, используемых в классах - диалоговые окна
//{{AFX_DATA_INIT //}}AFX_DATA_INIT	описание инициализации элементов данных (в файле реализации классов - диалоговые окна)
//{{AFX_DATA_MAP //}}AFX_DATA_MAP	описание макрокоманд DDX, предназначенных для связывания элементов данных класса и органов управления диалоговых окон (в файле реализации классов - диалоговые окна)
//{{AFX_MSG //}}AFX_MSG	описание методов классов, которые предназначены для обработки сообщений
//{{AFX_MSG_MAP //}}AFX_MSG_MAP	описание макрокоманд таблицы сообщений
//{{AFX_VIRTUAL //}}AFX_VIRTUAL	описание переопределенных виртуальных методов класса

1.11. Настройка среды программирования для работы с библиотекой MFC

В пункте главного меню ГМ-Project, предназначенном для управления открытыми проектами, можно задать конкретные настройки проекта (языковые настройки, режимы транслирования, оптимизации, отладки, компоновки, использования библиотек) командой ГМ-Project-Settings… . При этом в появившемся окне Project Settings на вкладке General надо подключить библиотеку MFC с указанием технологии подключения. В пункте главного меню ГМ-Tools командой Options необходимо вызвать окно Options и на вкладке Directories (Пути) при необходимости уточнить местоположение библиотек, папок Visual Studio, файлов пользователя и порядок их просмотра при работе системы.

Соответствующие окна представлены на рисунках 4-6 ниже.

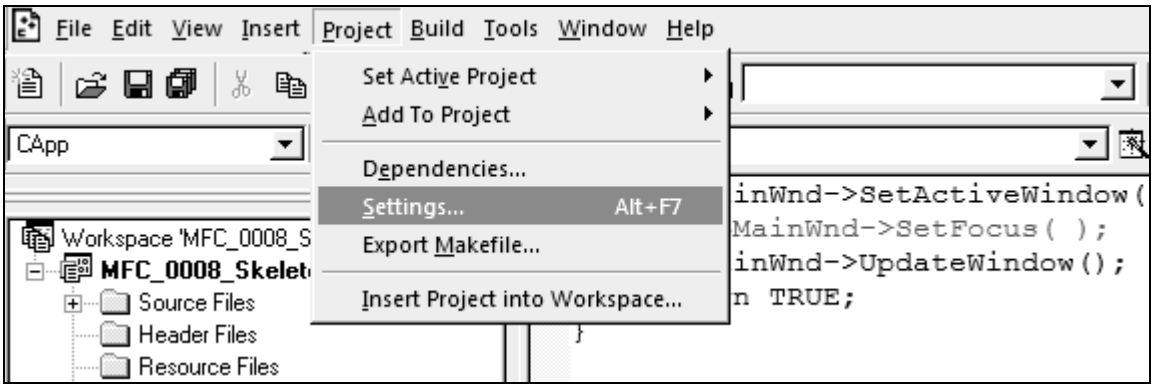


Рис. 4. Окно Project

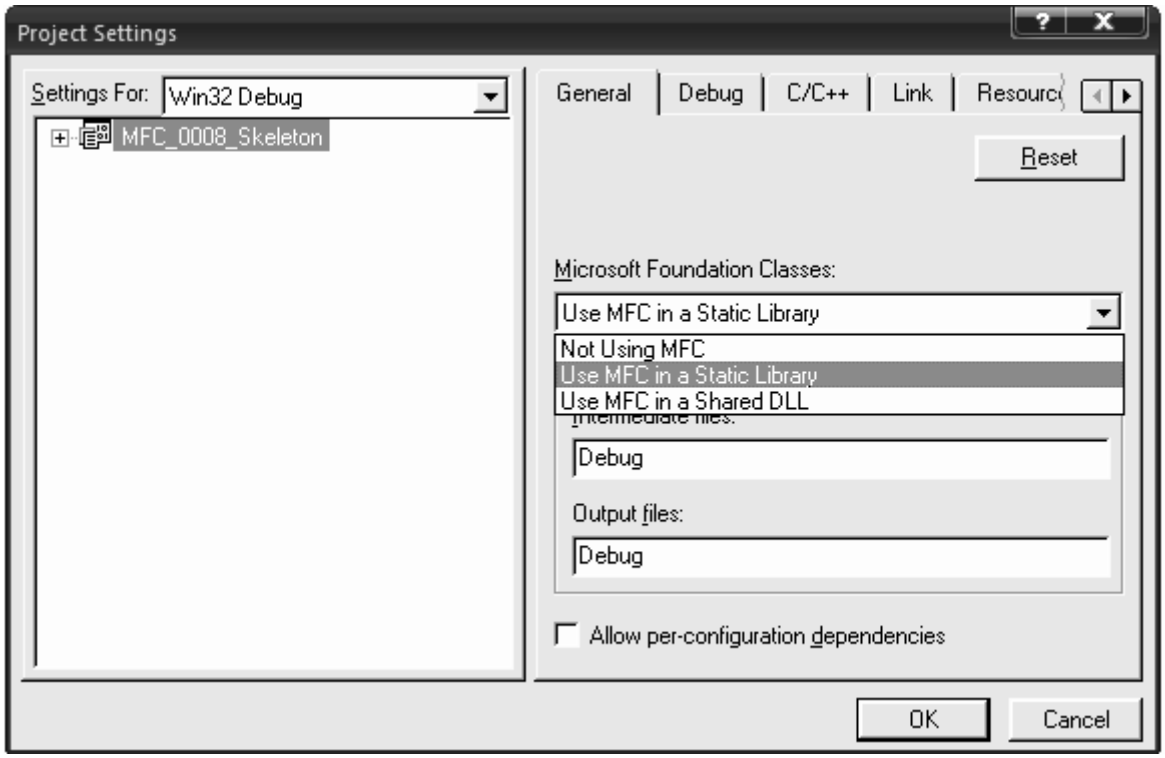


Рис. 5. Окно Project Settings

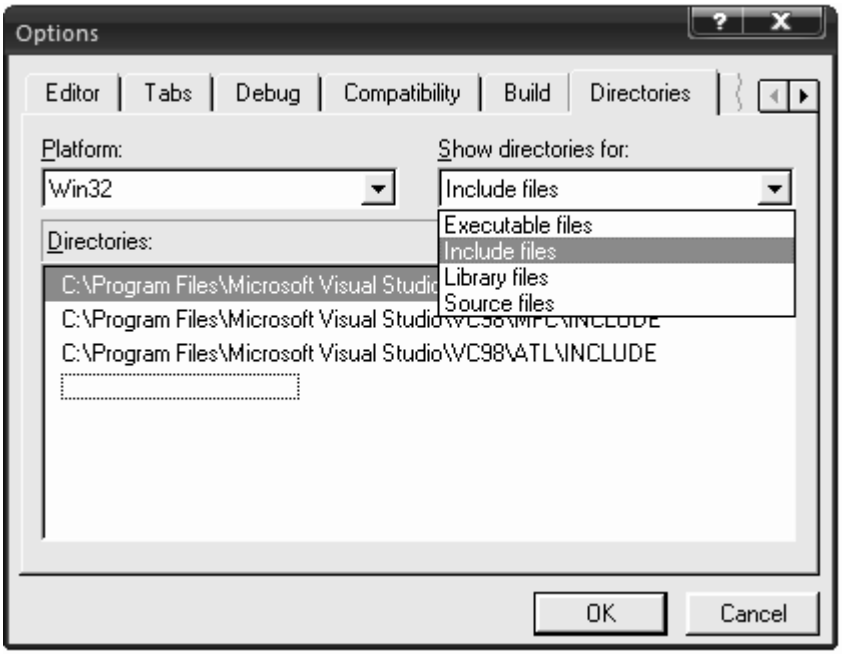


Рис. 6. Окно Options

2. MFC-ПРИЛОЖЕНИЕ. ТИПОВОЙ КАРКАС ПРИЛОЖЕНИЯ (ТКП)

2.1. Базовые классы MFC-приложения

У всех Windows-приложений фиксированная структура, определяемая функцией WinMain. Структура приложения, построенного из объектов классов библиотеки MFC, является еще более определенной. В таком приложении используются как минимум объекты двух типов классов, служащих как для организации оконного, графического интерфейса и обработки сообщений, так и для реализации функций самого приложения по запуску, завершению, восприятию и диспетчированию движением сообщений. Так однодокументное MFC-приложение, как правило, строится на базе двух классов библиотеки MFC.

Это класс CFrameWnd (класс ОКНО_С_РАМКОЙ), который служит базовым для создания интерфейса приложения (например окна приложения – класс моеОКНО). Окно с рамкой (типа главного окна) - это окно, внутри которого как бы размещены все остальные окна приложения.

Второй класс CWinApp (класс ПРИЛОЖЕНИЕ_WINDOWS) служит базовым для создания самого приложения (например класса моеПРИЛОЖЕНИЕ). Он является производным в том числе и от класса CCmdTarget и наследует от него таблицу (карту) сообщений (Message Map) - набор макрокоманд, позволяющий сопоставить сообщения Windows и команды метода класса.

Тогда скелет приложения выглядит как

```
Class  моеОКНО: class CMainWin
{ …};
class  моеПРИЛОЖЕНИЕ: public CWinApp
{ …};
```

Объект класса CWinApp (соответственно и объект производного от него класса, например, моеПРИЛОЖЕНИЕ) является сердцевиной приложения, отвечает за создание всех остальных объектов и обработку очереди сообщений. Он является глобальным и

создается еще до начала работы функции WinMain. Работа функции WinMain заключается в последовательном вызове двух методов объекта класса CWinApp: InitInstance и Run. В терминах сообщений можно сказать, что WinMain посылает объекту сообщение InitInstance, которое приводит в действие метод InitInstance. Получив сообщение InitInstance, объект (моеПРИЛОЖЕНИЕ) создает внутренние объекты приложения. Процесс создания выглядит как последовательное порождение одних объектов другими. Набор объектов, порождаемых в начале этой цепочки, определен структурой MFC. Затем сообщение Run - приводит в действие метод Run. В результате объект (моеПРИЛОЖЕНИЕ) начинает циклически выбирать сообщения из очереди и инициировать обработку сообщений обработчиками приложения.

Объект класса CFrameWnd (соответственно и объект производного от него класса, например, моеОКНО) имеет графический образ на экране, с которым может взаимодействовать пользователь. Это интерфейсный объект, связанный с Windows-окном. У него есть главная рамка и с ним могут быть связаны объекты документ и облик. При этом документ содержит данные приложения, облик организует представление этих данных на экране, а окно главной рамки содержит все остальные окна приложения. Когда приходит сообщение (пользователь, например, выбирает команду меню главного окна и возникает командное сообщение), то объектом класса CWinApp (моеПРИЛОЖЕНИЕ) они отправляются сначала объекту главная рамка, а затем объектам документ и облик, информируя их о событии, о пришедшей от пользователя команде. Фрагмент иерархии классов MFC, используемой при создании MFC-приложений, представлен на рисунке 2.

2.2. Создание окна (класса окна) приложения. Обработка сообщений, адресуемых окну

Как правило, все происходящие события и соответствующие сообщения адресуются активному окну приложения, точнее специальным функциям, обработчикам (программам обратного вызова), выполняющим их обработку. Сообщения несут в своих атрибутах всю информацию о происшедшем событии, необходимую для организации их обработки. Стандартные имена сообщений (см. файл windows.h) формируются по правилам

<ИдентификаторСообщения> ::= WM_<НазваниеСообщения>.

В MFC есть набор предопределенных обработчиков, которые являются членами класса CWnd и могут быть переопределены в приложении, где пользователь и задает алгоритм обработки. Имена обработчиков формируются по правилам

<ИмяОбработчика> ::= On<НазваниеСообщения*> ,

где НазваниеСообщения* конструируется как составное имя, образованное без символов подчеркивания.

Прототип функции-обработчика

afx_msg void <ИмяОбработчика> ([СписокПараметров]).

Сам обработчик имеет вид

```
afx_msg void  моеОКНО:: <ИмяОбработчика> ([СписокПараметров ])
{
    CPaintDC dc(this); или CClientDC dc(this);
    <ФрагментПользователя>
};  .
```

Для указания сообщений, которые должно обрабатывать приложение, используется макрокоманда включения. Это реализуется описанием (макрокоманда DECLARE_MESSAGE_MAP) очереди сообщений (карты подключения) окна вида

```
BEGIN_MESSAGE_MAP(моеОКНО, CFrameWnd)
    Список макрокоманд включения
END_MESSAGE_MAP()   .
```

Макрокоманда включения формируется по правилам
ON_< ИдентификаторСообщения >([СписокПараметров]).
Например, для сообщения LBUTTONDOWN получим ИдентификаторСообщения WM_LBUTTONDOWN, ИмяОбработчика OnLButtonDown, макрокоманда включения ON_WM_LBUTTONDOWN(), прототип функции-обработчика
afx_msg void OnLButtonDown(UINT flags, CPoint loc).
Однако для сообщения COMMAND ИдентификаторСообщения WM_COMMAND, ИмяОбработчика OnCommand, макрокоманда включения ON_COMMAND (*ИмяРесурса*, *ИмяОбработчика*) !
Таким образом, для того, чтобы конкретное окно (например, описанное в классе моеОКНО) реагировало на сообщения, необходимо для каждого из сообщений:
- описать в классе окна соответствующий обработчик;
- включить чувствительность окна к этому типу сообщений.
Примерная структура описаний приведена ниже

```
class моеОКНО : public CFrameWnd
{
  public:
      моеОКНО();
      afx_msg void <ИмяОбработчика> ([СписокПараметров ]);
       ….
       afx_msg void <ИмяОбработчика> ([СписокПараметров ]);
       DECLARE_MESSAGE_MAP()
};

моеОКНО :: моеОКНО()
{
    Create(NULL,”НазваниеОкна”);
};

afx_msg void моеОКНО ::<ИмяОбработчика> ([СписокПараметров ])
{
    CPaintDC dc(this); или CClientDC dc(this);
    <ФрагментПользователя>
};

…..

BEGIN_MESSAGE_MAP(моеОКНО,CFrameWnd)
    ON_< ИдентификаторСообщения >([СписокПараметров ])
     ……
```

```
END_MESSAGE_MAP()   .
```

Например, для обработки сообщений WM_PAINT, WM_LBUTTONDOWN

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

CMainWin::CMainWin()
{
      Create(NULL, "Пример");
}

BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
      ON_WM_PAINT()
      ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()

afx_msg void CMainWin::OnPaint()
{
       CPaintDC   TheDC(this);
        ……
};

afx_msg void CMainWin::OnLButtonDown(UINT flags, CPoint loc)
{
            ……
};    .
```

2.3. Создание приложения (класса приложения)

Для создания пользовательского класса приложения (например, класса моеПРИЛОЖЕНИЕ) в качестве базового используется класс MFC – приложение CWinApp. В создаваемом классе необходимо описать метод InitInstance (virtual BOOL CWinApp:: InitInstance( )), выполняющий роль конструктора и отвечающий за инициализацию приложения (см. § 2.1). В нем используется свойство класса CWinThread - m_pMainWnd (CWnd * CWinThread:: m_pMainWnd) для указания на CWnd-объект (окно), т.е. на интерфейсный объект пользовательского класса, принадлежащий приложению.

В методе InitInstance вначале динамическим образом конструируется интерфейсный объект пользовательского класса (например, моеОКНО, см. § 2.2). Его адрес сохраняется в переменной m_pMainWnd. Затем объект (окно) выводится методом CWnd::ShowWindow(m_nCmdShow) и обновляется методом CWnd::UpdateWindow ( ). Примерный текст описания класса приведен ниже

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

BOOL CApp :: InitInstance ( )
{
     m_pMainWnd = new моеОКНО;
     m_pMainWnd -> ShowWindow ( m_nCmdShow );
     m_pMainWnd -> UpdateWindow ( );
     return TRUE;
}  .
```

2.4. Структура типового MFC-приложения

Общая структура типового MFC-приложения, реализованного в виде проекта с одним модулем на базе двух пользовательских классов, производных от CFrameWnd (например, моеОКНО) и CWinApp (например, моеПРИЛОЖЕНИЕ) **,** представлена ниже. Подключение заголовка <iostream> делает видимым описание стандартного пространства имен std и позволяет использовать команду using namespace std.

```
#include  <afxwin.h>
#include  <iostream>
[ < Команды_препроцессора > ]
using namespace std;
[ < Прототипы_прикладных_функций > ]
[ < Описания_глобальных_объектов > ]
< Описание_класса_ГЛАВНОЕ_ОКНО_ПРИЛОЖЕНИЯ_ПОЛЬЗОВАТЕЛЯ >
< Описание_класса_ПРИЛОЖЕНИЕ_ПОЛЬЗОВАТЕЛЯ >
< Создание_экземпляра_класса_ПРИЛОЖЕНИЕ_ПОЛЬЗОВАТЕЛЯ >  .
```

Секция <Описание_класса_ГЛАВНОЕ_ОКНО_ПРИЛОЖЕНИЯ_ПОЛЬЗОВАТЕЛЯ> рассмотрена в § 2.2, а секция < Описание_класса_ПРИЛОЖЕНИЕ_ПОЛЬЗОВАТЕЛЯ > приведена в § 2.3.

Более детально структура типового MFC-приложения описана ниже

```
#include  <afxwin.h>
#include  <iostream>
using namespace std;
//=== Описания глобальных объектов ============================
//=== Прототипы_прикладных_функций ============================
//=== Описание класса ГЛАВНОЕ_ОКНО_ПРИЛОЖЕНИЯ_ПОЛЬЗОВАТЕЛЯ
class моеОКНО: public CFrameWnd
{
 public:
```

```
    CMainWin ( );
    afx_msg void On<ИмяСообщения>( < ПараметрыСообщения > );
    ……….
    afx_msg void On<ИмяСообщения>( < ПараметрыСообщения > );
    afx_msg void OnPaint ( );
    DECLARE_MESSAGE_MAP( )
};

BEGIN_MESSAGE_MAP(CMainWin,CFrameWnd)
    ON_WM_<ИмяСообщения> ( < ПараметрыСообщения > )
    ……..
    ON_WM_<ИмяСообщения> ( < ПараметрыСообщения > )
    ON_WM_PAINT( )
END_MESSAGE_MAP( )

моеОКНО:: моеОКНО ( )
{
    Create (NULL, " КАРКАС MFC-ПРИЛОЖЕНИЯ ");
};

afx_msg void On<ИмяСообщения>( < ПараметрыСообщения > )
{
     CClientDC  dc (this);
     < ФрагментПользователя >
};
     ……

afx_msg void CMainWin :: OnPaint ( )
{
     CPaintDC  dc (this);
     < ФрагментПользователя >
};

//=== Описание_класса_ПРИЛОЖЕНИЕ_ПОЛЬЗОВАТЕЛЯ ========
class моеПРИЛОЖЕНИЕ: public CWinApp
{
 public:
    BOOL InitInstance ( );
};

BOOL CApp :: InitInstance ( )
{
    m_pMainWnd = new моеОКНО;
    m_pMainWnd -> ShowWindow ( m_nCmdShow );
    m_pMainWnd -> UpdateWindow ( );
    return TRUE;
};   .
```

2.5. Структура и текст типового каркаса MFC-приложения

Большинство однотипных MFC-приложений имеют одинаковую базовую структуру, называемую скелетом, каркасом приложения. Соответственно в Visual Studio существуют мастера приложений, которые автоматически генерируют тексты каркасов приложений заданных типов. Простейший, типовой каркас MFC-приложения (ТКП) функционально совпадает с типовым каркасом Windows-приложением и выводит на экран окно с рамкой, клиентской областью (как на рисунке 7 ниже), содержащее системное меню, системные кнопки (закрытия, минимизации-максимизации окна). Окно масштабируемое, перекрывающееся (с автоматической посылкой сообщений на перерисовку при искажении), с возможностью перемещения (при этом оно автоматически перерисовывается) и изменения размеров. Без пользовательской реакции на сообщения (события).

Базируется на двух пользовательских классах производных от CFrameWnd (например, пользовательский класс моеОКНО) и CWinApp (например, пользовательский класс моеПРИЛОЖЕНИЕ). Для получения контекста устройства – создания соответствующего объекта используется класс CDC и его производные классы. Фрагмент иерархии классов MFC, используемых при создании каркасов приложений, приведен на рисунке 8.

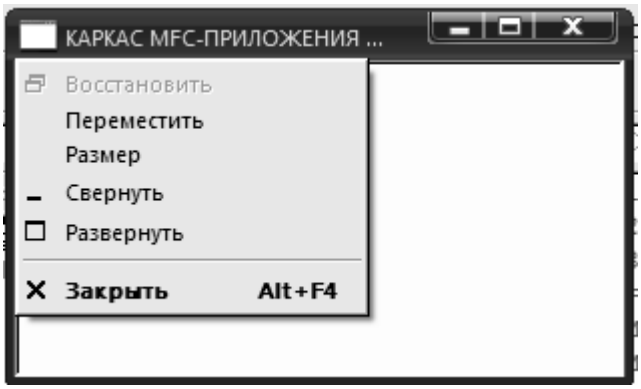


Рис. 7. Интерфейс типового каркаса приложения

Отношения классов каркаса MFC-приложения приведены на рисунке 9. Указанные классы связаны отношением типа “агрегация”, т.к. класс моеПРИЛОЖЕНИЕ включает объект класса моеОКНО (точнее указатель m_pMainWnd на объект класса моеОКНО). Кратность отношения 1:1. Кроме этого, в методе InitInstance класс моеПРИЛОЖЕНИЕ использует методы класса моеОКНО для его запуска, визуализации, обновления. Поэтому класс моеПРИЛОЖЕНИЕ и использует класс моеОКНО (существует отношение типа “зависимость”).

Подобные каркасы можно получить автоматически, используя мастер MFC AppWizard(exe) в таких режимах как Single document, Multiple document, Dialog based и т.п. Однако такие автоматические каркасы включают большое количество дополнительных ресурсов, компонентов, средств.

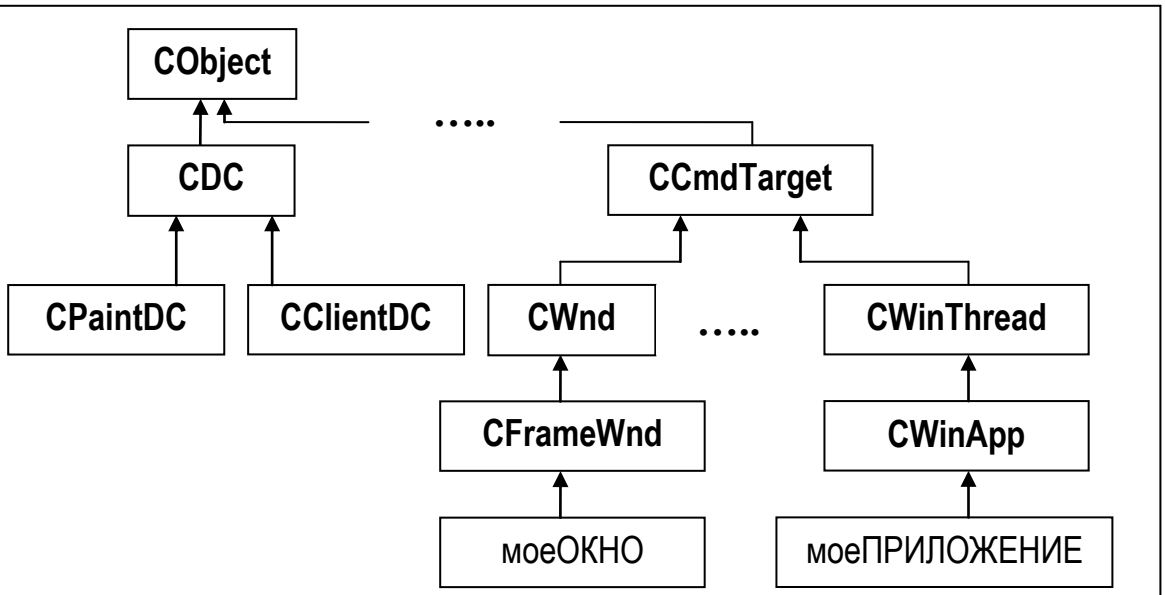


Рис. 8. Фрагмент иерархии классов MFC, используемых при создании каркасов приложений

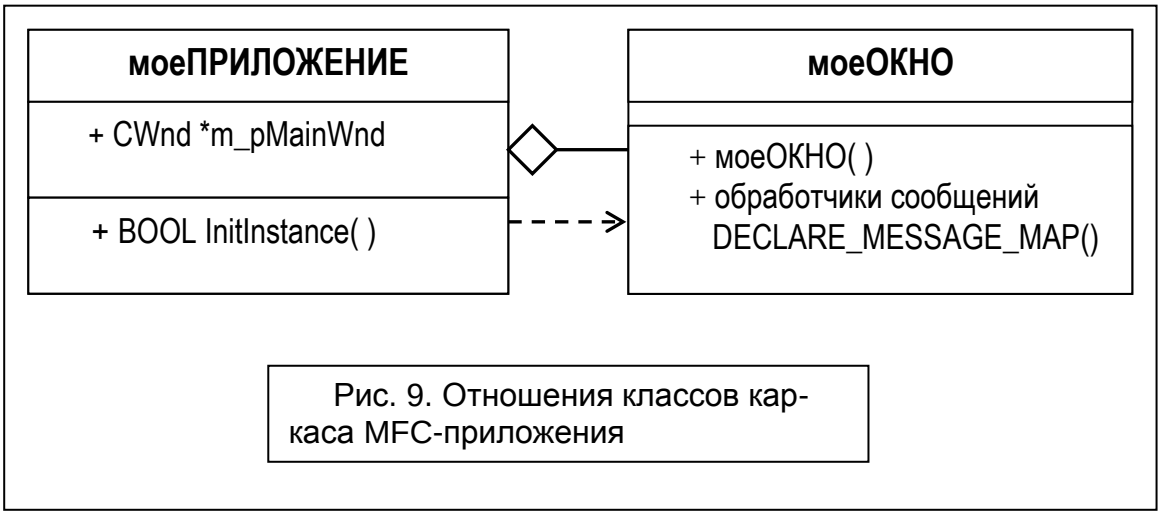


Рис. 9. Отношения классов каркаса MFC-приложения

Ниже рассматривается и комментируется примерный текст ТКП.

```
#include    <afxwin.h>
//#include  <afxext.h>
//#include  <afxdisp.h>
//#include  <afxdtctl.h>
//#ifndef   _AFX_NO_AFXCMN_SUPPORT
//#include  <afxcmn.h>
//#endif   // _AFX_NO_AFXCMN_SUPPORT
#include  <iostream>
using namespace std;
```

```
// ===== класс главного ОКНА приложения ===================
class WINDOW : public CFrameWnd
{
public:
    // Конструктор окна
      WINDOW ();
     // Прототип обработчика сообщения WM_LBUTTONDOWN
     // afx_msg void  OnLButtonDown (UINT flags, CPoint loc);
     // Макрос объявления очереди сообщений окна
       DECLARE_MESSAGE_MAP()
};

// ===== конструктор окна - Создание окна
WINDOW:: WINDOW ()
{
     Create ( NULL,   // имя стиля окна
                 "Главное окно MFC-приложения",  // заголовок окна
               // DWORD Style = WS_OVERLAPPEDWINDOW,  // стиль окна
               // const RECT &XYsize =  rectDefault,  // положение и размеры окна
               // CWnd *Parent = NULL,  // окно-предок
               // LPCSTR MenuName = NULL,  // имя главного меню
               // DWORD ExStyle = 0,  // спецификатор расширенного стиля
               // CCreateContext  * Context  = NULL);  // дополнительная информация
}

// ===== метод Обработчик сообщения
//          нажата левая клавиша “мыши”  WM_LBUTTONDOWN
// afx_msg void WINDOW::OnLButtonDown (UINT flags, CPoint loc)
//  { < ФРАГМЕНТ_ПОЛЬЗОВАТЕЛЯ > };

// ===== очередь сообщений главного окна
BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
      // Макрокоманды включения обрабатываемых сообщений
      // ON_WM_LBUTTONDOWN()
      //  ……
END_MESSAGE_MAP()

// ===== класс ПРИЛОЖЕНИЕ ===================
class APPLICATION : public CWinApp
{
public:
   // Конструктор приложения
     BOOL InitInstance();
```

```
};

// ===== конструктор приложения - Инициализации приложения
BOOL APPLICATION::InitInstance ()
{
    // Создание объекта класса WINDOW
      m_pMainWnd = new  WINDOW;
    // Визуализация окна в заданном при запуске приложения стиле m_nCmdShow
      m_pMainWnd -> ShowWindow ( m_nCmdShow );
    // Посылка сообщения о необходимости обновить содержимое окна
      m_pMainWnd -> UpdateWindow ();
      return TRUE;
}

// ===== создание и запуск экземпляра приложения ====================
APPLICATION    TheApplication;    .
```

Здесь заголовочный файл <afxwin.h> используется для подключения стандартных компонентов MFC; <afxext.h> используется для подключения расширений MFC; <afxdisp.h> - для подключения классов автоматизации MFC; <afxdtctl.h> - для подключения MFC-поддержки общих элементов управления Internet Explorer 4; <afxcmn.h> - для подключения MFC-поддержки общих элементов управления Windows.

В качестве имени стиля окна здесь указан NULL, что соответствует стандартному стилю MFC-окна с клиентской областью, с рамкой. По умолчанию параметр стиля окна задается как WS_OVERLAPPEDWINDOW, что означает создание перекрывающегося окна на базе стилей WS_OVERLAPPED, WS_THICKFRAME и WS_SYSMENU, WS_CAPTION, WS_MINIMIZEBOX, WS_MAXIMIZEBOX, т.е. масштабируемого окна с системными меню и кнопками. По умолчанию положение и размеры окна задаются как rectDefault, т.е. определяется автоматически системой Windows.

Примерный текст ТКП с “заглушками” для обработчиков приведен в ПРИЛОЖЕНИИ 3. Примерный текст “минимального” ТКП без обработчиков приведен в ПРИЛОЖЕНИИ 4.

2.6. Построение типового каркаса MFC-приложения в Visual Studio C++

Используя мастер каркасов MFC AppWizard(exe), можно получить достаточно сложные каркасы (заготовки) для создания приложений различных типов. Например, можно создать каркасы типа Single, Multiple, Dialog Based documents. Но в целом все они повторяют структуру типового каркаса MFC-приложения.

Для создания MFC-приложения на базе ТКП необходимо:

1) создать ТКП;

2) добавить в ТКП все необходимые компоненты, исходя из назначения приложения.

Для создания ТКП надо:

1) создать “пустой” каркас мастером Win32 Application – выполнить команду File-New, в окне New на вкладке Projects выбрать мастер Win32 Application, задать режим “An empty project”;

2) создать “пустой” файл *.cpp – выполнить команду File-New, в окне New на вкладке Files выбрать тип файла C++ Source File, задать его (пользовательское) имя;

3) открыть указанный файл и скопировать в него текст ТКП (см. ПРИЛОЖЕНИЯ 3, 4);

4) включить в главном меню в пункте Project-Settings, на вкладке General режим – “use MFC in Static library”.

3. ОСНОВЫ РАБОТЫ В MFC-ПРИЛОЖЕНИЯХ С КЛИЕНТСКОЙ ОБЛАСТЬЮ ОКНА

3.1. Преобразование типов данных

Операции ввода-вывода данных, как правило, производятся с данными в строковом формате. В то время как в самом приложении выводимые, а также вводимые данные могут принадлежать к типам, отличным от строкового типа. Это соответственно требует перевода данных из строкового формата или в строковый формат.

1. Преобразование данных в строковый формат может производиться, например, с использованием функций wsprintf, sprintf, _gcvt.

Функция wsprintf имеет прототип

int wsprintf(LPTSTR lpOut, **LPCTSTR** lpFmt, ... **)**

или ПРЕОБРАЗОВАТЬ_В_СТРОКУ (*РезультирующаяСтрока*, *ШаблонВывода, СписокВывода*).

Здесь параметры:

- *РезультирующаяСтрока* – результат преобразования *СпискаВывода*;
- *ШаблонВывода* – строка форматных кодов, управляющих преобразованием;
- *СписокВывода* – преобразуемые данные.

Пример использования функции приведен ниже

```
char szMyString1 [80] = “ ”;
char szMyString2 [80] = “ ”;
int    intNumber = 34;
……..
wsprintf (szMyString1, "Был год - %d", 1952);
wsprintf (szMyString2, "%d", intNumber);   .
```

Пример использования аналогичной функции **sprintf** приведен ниже.

```
sprintf(szMyString1,"%f",FloatNumber);
```

```
#include <stdio.h>
……
float FloatNumber = 12.345;
char OutputStr[100];
…….
sprintf(OutputStr,"%f",FloatNumber);   .
```

Другая функция, применяемая для указанных преобразований, _gcvt. Функция преобразует значение, представленное в формате с плавающей точкой, в строку и имеет прототип

char *_gcvt(double *Value*, **int** *Digits*, **char *** *Buffer*);

или ПРЕОБРАЗОВАТЬ_В_СТРОКУ(*ВыводимоеЧисло*, *ДлинаСтроки*, *Строка*).
Пример использования функции приведен ниже

```
#include  <stdlib.h>
#include  <stdio.h>
……
char RezultString [50];
double Value = -3.1415e5;
_gcvt(Value, 7, RezultString);  .
```

2. Преобразование данных из строкового формата выполняется, например, функциями типа sscanf, atof и другими.

3.2. Ввод-вывод данных

В общем случае для организации вывода информации, например, в рабочую область окна приложения, следует предварительно получить контекст устройства КУ и соответственно получить доступ к необходимым функциям КУ.

1. ВЫВОД информации (числовых и не числовых данных) в виде строк в специальном окне - окне сообщений. Для этих целей можно использовать метод MessageBox() класса CWnd либо глобальную функцию afxMessageBox().
Прототип метода CWnd::MessageBox

int CWnd:: MessageBox (LPCSTR *lpszText*, **LPCSTR** *lpszTitle* = NULL, **UINT** *MBType* = MB_OK**)**

или ВЫВЕСТИ_ОКНО_СООБЩЕНИЯ (*СтрокаСообщения*, *НазваниеОкнаСообщения*, *СоставЭлементовОкнаСообщения*).

Метод создает и выводит окно с заголовком, сообщением, комбинацией пиктограмм и командных кнопок. Возвращает значение, определяющее результат работы пользователя с окном - код нажатой кнопки (или 0, если недостаточно памяти для создания окна). Коды завершения приведены в таблице ниже

Таблица 3. Коды завершения

Значение	Описание
IDABORT	выбрана кнопка Abort
IDCANCEL	выбрана кнопка Cancel
IDIGNORE	выбрана кнопка Ignore
IDNO	Ни одна кнопка не была выбрана
IDOK	выбрана кнопка OK

IDRETRY	выбрана кнопка Retry
IDYES	выбрана кнопка Yes

Параметры:

- *СтрокаСообщения* - указатель на CString-объект или строку С (с нуль-символом), которая содержит выводимое сообщение;

- *НазваниеОкнаСообщения* - указатель на CString-объект или строку С (с нуль-символом), которая содержит заголовок окна сообщения. Если указатель равен NULL, то по умолчанию используется заголовок “Error”;

- *СоставЭлементовОкнаСообщения* - определяет состав окна сообщения (комбинацию пиктограмм и командных кнопок) и его поведение.

Используемые пиктограммы: MB_ICONHAND, MB_ICONSTOP, MB_ICONERROR, MB_ICONQUESTION, MB_ICONEXCLAMATION, MB_ICONWARNING, MB_ICONASTERISK, MB_ICONINFORMATION.

Используемые комбинации кнопок: MB_ABORTRETRYIGNORE, MB_OK, MB_YESNO, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNOCANCEL.

Установки модальности окна: MB_APPLMODAL (используется по умолчанию. В этом случае пользователь для продолжения работы в текущем окне должен ответить на окно сообщения. Тем не менее, он может перейти в окна других приложений и продолжить работу там), MB_SYSTEMMODAL (применяется для оповещения пользователя о серьезных ситуациях, не терпящих отлагательства).

Установки кнопок по умолчанию: MB_DEFBUTTON1 (первая кнопка), MB_DEFBUTTON2 (вторая кнопка), MB_DEFBUTTON3 (третья кнопка).

Пример использования функции представлен ниже

```
MessageBox (“Работает MessageBox ”, "Демо", MB_OK); .
```

Пример использования окна сообщения для ввода управляющей информации в виде данных о реакции пользователя (нажатых кнопках), представлен ниже

```
if  ( MessageBox ("Работает MessageBox ", "Демо", MB_OKCANCEL) == IDCANCEL)
      MessageBox ("Нажата кнопка CANCEL", "Демо", MB_OKCANCEL);  .
```

Прототип функции afxMessageBox, которая функционально работает так же как и предыдущий метод, но реализована как глобальная функция MFC, приведен ниже

int afxMessageBox(LPCSTR *lpszText*, **UINT** *MBType* = MB_OK, **UINT** *HelpID* = 0**)**

или ВЫВЕСТИ_ОКНО_СООБЩЕНИЯ (*СтрокаСообщения*, *СоставЭлементовОкнаСообщения*, *УказательРазделаСправки*).

Другая используемая форма

int AFXAPI AfxMessageBox(UINT *nIDPrompt*, **UINT** *nType* = MB_OK, **UINT** *nIDHelp* = (UINT) -1),

где

- *lpszText* указатель на CString-объект или строку С (с нуль-символом), которая содержит выводимое сообщение;

- *nType* определяет стиль окна - состав окна сообщения и его поведение;

- *nIDHelp* определяет ID–идентификатор контекстно-зависимого справочного сообщения, связанного с этим окном;

- *nIDPrompt* - определяет ID–идентификатор строки в таблице строк ресурсов приложения, которая используется в качестве выводимого сообщения.

2. ВЫВОД информации (числовых и не числовых данных) в виде строк в клиентской области окна. Требует предварительного получения КУ. Сам вывод можно организовать методом TextOut, который перегружен в двух вариантах – для работы с классическими строками С и со string-строками.

Прототип метода CDC:: TextOut

virtual BOOL CDC:: TextOut (int *X*, **int** *Y*, **LPCSTR** *lpszString*, **int** *Length***)** ,
BOOL CDC:: TextOut (int *X*, **int** *Y*, **const CString &***StrOb*)

или ВЫВЕСТИ_ТЕКСТ (*КоординатаX*, *КоординатаY*, *СтрокаВывода*, *ДлинаСтрокиВывода*) и соответственно ВЫВЕСТИ_ТЕКСТ (*КоординатаX*, *КоординатаY*, *СтрокаВывода*).

Пример использования метода

```
CClientDC  dc(this);
char  OutputString;
char  Comment [25] = “Выводится строка - ”;
……
dc.TextOut(1,1, Comment, strlen(Comment));
dc.TextOut(1,20, OutputString, strlen(OutputString));
```

В примере, приведенном ниже, в приложении, созданном на базе ТКП, при нажатии левой кнопки “мыши” (сообщение WM_LBUTTONDOWN и обработчик OnLButtonDown) осуществляется вывод заранее инициализированной строки OutputStr в клиентскую область экрана, начиная с позиции, в которой было зафиксировано положение курсора “мыши” в момент события.

```
char OutputStr[80] = " Строка вывода ";

class CMainWin : public CFrameWnd
{
public:
      ……
      afx_msg void OnLButtonDown(UINT flags, CPoint loc);
      DECLARE_MESSAGE_MAP()
};

CMainWin::CMainWin()
{
      Create(NULL, "Пример");
}

BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
  ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()

afx_msg void CMainWin::OnLButtonDown(UINT flags, CPoint loc)
{
```

```
        CClientDC dc(this);
        dc.TextOut(loc.x,loc.y, OutputStr, strlen(OutputStr));
};

class CApp : public CWinApp
{
public:
        BOOL InitInstance();
};

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

CApp  App;  .
```

3. ВЫВОД информации (числовых и не числовых данных) может производиться в виде строк в стандартных элементах управления (ЭУ), например в окошке редактирования (класс CEdit), в списковом окне (класс CList) и т.п..

4. ВВОД ограниченной информации может производиться в виде получения реакции, ответа пользователя на полученное в окне сообщение. Вводимая информация представляет собой код нажатой пользователем кнопки окна сообщения, выведенного функцией MessageBox (см. п. 1 данного параграфа).

5. ВВОД произвольных данных в строковом формате может производиться из окна (поля) редактирования ресурса – диалоговое окно.

4. ПОРЯДОК ВЫПОЛНЕНИЯ

ТЕМА РАБОТЫ: “Изучение типового каркаса MFC-приложения (ТКП). Сообщения. Контекст устройства. Организация вывода в главное окно”.

ЦЕЛЬ РАБОТЫ:

- изучение ТКП, разработка и реализация MFC-приложений с использованием ТКП;
- организация обработки сообщений (сообщений “мыши” типа ON_WM_LBUTTONDOWN, перерисовки WM_PAINT, клавиатуры типа WM_CHAR и др.);
- организация текстового и графического вывода в главное окно (использование окон сообщений MessageBox, методов TextOut и др.);
- изучение проблем перерисовки.

СОДЕРЖАНИЕ ОТЧЕТА:

- описание ТКП (интерфейса, диаграммы классов, диаграммы взаимодействия объектов);

– описание приложения (ТКП), реализованного в виде проекта из двух и 6 файлов типов *.h, *.cpp (описание компоновки проекта приложения);

- описание приложения, реализующего ввод символов с клавиатуры и вывод в позиции, указанной “мышью”;

- описание приложения, реализующего построение фигур из линий с использованием пользовательского класса ЛИНИИ и класса MFC CPoint.

ПРИМЕЧАНИЕ:

- для всех заданий в отчете следует приводить диаграммы классов и компоновки (в нотации UML);

- каждое реализованное задание (приложение) предъявлять на ПЭВМ для проверки преподавателю;

- пункты 1-9 рассчитаны на 4 часа, пункты 10-20 рассчитаны на 4 часа;

- пункты, помеченные звездочкой, выполняются самостоятельно по указанию преподавателя.

ПОРЯДОК ВЫПОЛНЕНИЯ.

1. **Изучить теоретический материал**

- ознакомиться с общими сведениями об ОС, технологиях программирования, оконных приложениях и библиотеке MFC (см. §§ 1.1-1.5);

- изучить основные типы сообщений, прототипы соответствующих обработчиков сообщений (см. §§ 1.6-1.7);

- ознакомиться с методами управления контекстом устройства и проблемами перерисовки (см. §§ 1.8-1.9);

- ознакомиться со средой разработки (см. § 1.10);

- ознакомиться с составом классов типового каркаса MFC-приложения (см. § 2.1);

- изучить технологию обработки сообщений (см. § 2.2);

- ознакомиться с архитектурой типового каркаса MFC-приложения, используемых классах и методах (см. §§ 2.3-2.4);

2. **Создать** “пустое” приложение (типовой каркас приложения) (с именем EX1) в соответствии с § 1.11 (настройки на библиотеку MFC) и § 2.5. Это каркас MFC-приложения с главным окном, но без обработки сообщений. Запустить. Изучить его свойства для ОТЧЕТа. Разработать диаграммы UML.

3. **Модифицировать** приложение (с именем EX1), включив чувствительность к сообщению WM_LBUTTONDOWN (см. §§ 1.6-1.7). Для этого

– в классе CMainWin включить (разкомментировать) прототип функции-обработчика

```
afx_msg void OnLButtonDown(UINT flags, CPoint loc);
```

- в карте сообщений BEGIN_MESSAGE_MAP() включить (разкомментировать) чувствительность приложения к нажатию клавиши “мыши”

```
ON_WM_LBUTTONDOWN();
```

- описать (разкомментировать) соответствующую функцию-обработчик;

- выполнить приложение, исправить ошибки;

- добавить в тело обработчика вывод сообщения о его запуске с помощью метода MessageBox (см. § 3.2);

- добавить в тело обработчика вывод текста “Работает ТКП-MFC” в главном окне с помощью метода TextOut (см. § 3.2), для чего получить контекст устройства – создать объект класса CClientDC (см. § 1.8);

- запустить приложение, выполнить манипуляции с окном (перемещение, изменение размеров, перекрывание другим приложением) и анализировать эффект перерисовки.

4. **Создать** приложение (на базе ТКП) (с именем EX2), аналогичное приложению, созданному в предыдущем пункте, но включив чувствительность к сообщению WM_PAINT (см. §§ 1.6-1.7). Для этого

– в классе CMainWin включить прототип функции-обработчика

afx_msg void OnPaint();

- в карте сообщений BEGIN_MESSAGE_MAP() включить чувствительность приложения к сообщению перерисовки

ON_WM_PAINT();

- описать соответствующую функцию-обработчик;
- выполнить приложение, исправить ошибки;
- добавить в тело обработчика вывод сообщения о его запуске с помощью метода MessageBox (см. § 3.2);
- добавить в тело обработчика вывод текста “Работает ТКП-MFC” в главном окне с помощью метода TextOut (см. § 3.2), для чего получить контекст устройства – создать объект класса CClientDC (см. § 1.8);
- запустить приложение, выполнить манипуляции с окном (перемещение, изменение размеров, перекрывание другим приложением) и анализировать эффект перерисовки;
- добавить табуляцию любой функции с выводом результатов в табличном виде (аргумент-функция).

5. **Создать** приложение (на базе ТКП) (с именем EX3) для вывода координат курсора “мыши”, фиксируемых по щелчку ее левой клавиши. Для этого:

- создать ТКП и включить чувствительность к сообщению WM_LBUTTONDOWN;
- обеспечить вывод координат курсора “мыши”, фиксируемых по щелчку левой клавиши “мыши”, в обработчике сообщения WM_LBUTTONDOWN (см. § 1.8);
- запустить приложение, выполнить манипуляции с окном и анализировать эффект перерисовки.

6. **Модифицировать** приложение (на базе ТКП) (с именем EX3), решив проблему перерисовки - ввод координат производить по сообщению WM_LBUTTONDOWN, вывод по WM_PAINT. Для этого:

- включить чувствительность к сообщению WM_PAINT;
- модифицировать обработчик сообщения WM_LBUTTONDOWN, осуществляя в нем только получение и запоминание координат нажатия;

обеспечить вывод координат курсора (фиксируемых по щелчку левой клавиши “мыши” в обработчике сообщения WM_LBUTTONDOWN) в обработчике сообщения WM_PAINT;

- запустить приложение, выполнить манипуляции с окном и анализировать эффект перерисовки;
- для вызова перерисовки окна после каждого щелчка левой клавиши “мыши” в обработчике сообщений “мыши” использовать метод InvalidateRect (см. § 1.9)
- запустить приложение, выполнить манипуляции с окном и анализировать эффект перерисовки.

7. **Модифицировать** приложение (на базе ТКП) (с именем EX3) для вывода координат в той точке экрана, где был зафиксирован щелчок левой клавиши “мыши”.

8*. **Создать** приложение (с именем EX4) для рисования плоских изображений отрезками прямых линий. Координаты точек вводить нажатием левой клавиши “мыши” (по сообщению WM_LBUTTONDOWN). Введенные координаты сохранять в динамическом массиве (например, использовать шаблонный класс vector). Выводить изображение, путем соединения прямыми линиями соседних точек, по нажатию правой клавиши “мыши” (по сообщению WM_ RBUTTONDOWN).

Примечание. Для этого создать ТКП и обработчики сообщений. Для рисования точек, линий, геометрических фигур используются методы класса CDC, поэтому вначале необходимо создать объект этого класса. Для хранения координаты точки можно использовать структуру POINT или класс CPoint. Для перемещения пера (без рисования линии)

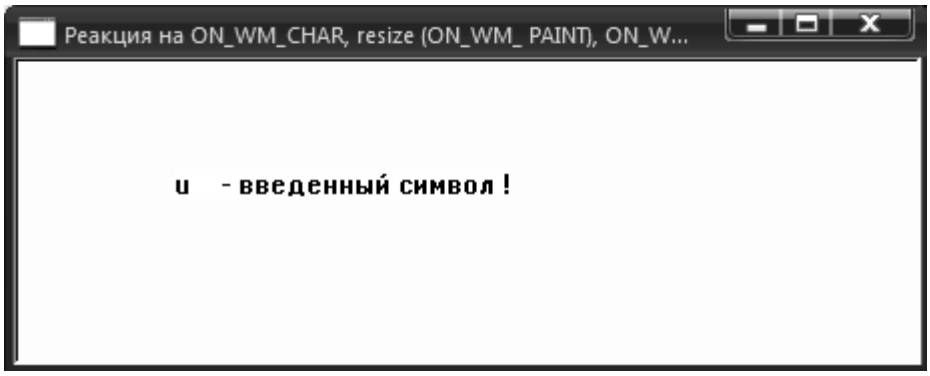
используется метод CDC::MoveTo(int x, int y) или CDC::MoveTo(POINT point). Для рисования прямой линии используется метод CDC::LineTo(int x, int y) или CDC::LineTo(POINT point). Всю недостающую информацию получить из справочной системы MSDN Visual Studio.

9*. **Создать** приложение (с именем EX5) аналогичное EX4, но с использованием пользовательского класса ЛИНИИ для работы с координатами точек, линиями.

10. **Создать** "пустое" приложение (типовой каркас приложения) (с именем EX6) так же как в п.2, выполнив перекомпоновку текста в виде двух модулей, файлов (файл интерфейсов классов и приложения и файл реализации классов и приложения). Привести диаграммы компонентов UML.

11. **Создать** "пустое" приложение (типовой каркас приложения) (с именем EX7) так же как в п.2, выполнив перекомпоновку текста в виде 6 модулей (разделив классы и приложение – для каждого класса и приложения описывать файл интерфейса и файл реализации). Привести диаграммы компонентов UML.

12. **Создать** приложение (с именем EX8) для ввода символа с клавиатуры (сообщение WM_CHAR) и эхо-вывода либо в левый верхний угол главного окна, либо в позицию, задаваемую нажатием левой клавиши мыши. Разработать диаграммы UML.



Примечание. Ввод и запоминание символа выполнять в обработчике сообщения WM_CHAR, ввод и запоминание координат выполнять в обработчике сообщения WM_LBUTTONDOWN, вывод выполнять в обработчике сообщения WM_PAINT. Вызывать перерисовку при вводе нового символа или при изменении координат вывода методом InvalidateRect(NULL).

13. **Создать** "пустое" приложение (типовой каркас приложения) (с именем EX9) с разными вариантами стиля главного окна (см. ПРИЛОЖЕНИЕ 1, см. § 1.5). В том числе, создать:

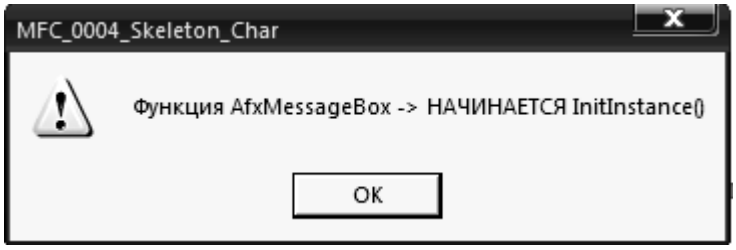
- типовое главное окно без возможности скроллинга в клиентской области окна;
- типовое главное окно со скроллингом в клиентской области окна;
- типовое главное окно без системных кнопок;
- типовое главное окно без системных кнопок заданного размера и местоположения, например с координатами левого верхнего угла 10, 10 и с координатами правого нижнего угла 200, 200. Для этого использовать сначала структуру типа RECT, а затем класс CRect (см. в справочной системе MSDN). Пример задания параметров области в структуре RECT приведен ниже

```
RECT TheRect  = {10, 10, 200, 200};
или
RECT TheRect;
TheRect..top = 10;
```

TheRect..left = 10;
TheRect..bottom = TheRect.right = 200;

- типовое главное окно с системным меню;
- типовое главное окно без возможности изменения размеров;
- типовое главное окно с возможностью изменения размеров и с начальным выводом на весь экран (см. в справочной системе MSDN параметры метода ShowWindow).

14. **Создать** “пустое” приложение (с именем EX10). Протоколировать все шаги работы его работы, начиная от запуска приложения, в том числе, начало и завершение его инициализации в конструкторе приложения BOOL CApp::InitInstance(), начало и завершение создания окна и т.д. Для этого использовать все из применимых в конкретной точке приложения средств: функции MessageBox и AfxMessageBox, метод MessageBox. При каждом выводе сообщать тип функции-метода и происшедшее событие, например



15*. **Создать** “скрытое” приложение (с именем EX11) на базе ТКП, которое после запуска не выводит окон и не имеет видимого интерфейса, но реагирует на нажатие клавиш (сообщение WM_CHAR) с восприятием символов без их отображения на экране. Работа завершается по вводу символа “n”.

ПРИМЕЧАНИЕ. Само главное окно должно создаваться, но не выводиться на экран, активизируясь методом SetActiveWindow (см. в справочной системе MSDN). Обработчик сообщения WM_CHAR только анализирует введенные символы. При вводе символа “n” производится посылка сообщения на завершение PostQuitMessage(1) (см. в справочной системе MSDN).

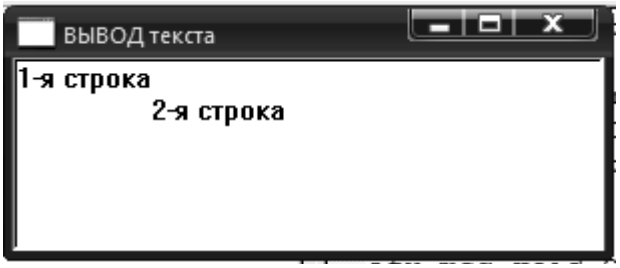
Убедиться, что после запуска приложения выход из него может быть произведен просто переходом в другое приложение, например, щелчком по экрану Studio. Однако на самом деле это не снимает приложение, а приводит к потере главным окном приложения активности. Это можно проследить по пропаданию текстового курсора при запуске приложения из Studio, а после его появления после потери активности можно контролировать корректность завершения просмотром списка задач в Диспетчере задач Windows.

16*. **Модифицировать** приложение. Вставить вывод, например, по MessageBox, повторить запуски приложения и анализировать корректность его завершения.

17*. **Модифицировать** приложение, добавив реакцию на потерю окном активности в виде завершения работы. Например, реагировать так же на сообщения WM_ACTIVATE и в случае потери активности в обработчике OnActivate использовать метод PostQuitMessage(1) (см. в справочной системе MSDN).

18*. **Модифицировать** приложение, добавив реакцию на таймер и обеспечив завершение работы приложения автоматически в течение заданного промежутка времени после потери окном активности (см. в справочной системе MSDN метод SetTimer, сообщение WM_TIMER, метод KillTimer).

19. **Создать** приложение (с именем EX12) на базе ТКП, которое выводит строки, например, как показано на рисунке ниже.



ПРИМЕЧАНИЕ. При самостоятельном выводе данных в клиентской области экрана требуется вычислять координаты начальной позиции для вывода первого символа каждой строки, а при переходе на новую строку окна вычислять новое значение координаты y. Для определения новой координаты X, после вывода текущей строки, использовать метод CDC::GetTextExtent() вычисления ее реальной длины в логических единицах. Для перехода на новую строку окна и вычисления нового значения координаты Y использовать данные из структуры TEXTMETRIC, которая хранит атрибуты (метрики) текущего шрифта, связанного с КУ. В том числе, поле tmHeight задает полную высоту символов используемого шрифта, tmExternalLeading определяет расстояние (интервал) между строками. Для доступа к структуре использовать метод CDC::GetTextMetrics() (см. в справочной системе MSDN).

20. **Создать** приложение (с именем EX13) на базе ТКП, которое выводит график функции: а) Y = sin(X); б) Y = (sin(X) + 3 * cos (X / 2) – 2 * sin (X / 5)).

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. Значения параметра dwStyle функции Create

Таблица 4. Значения параметра dwStyle

Параметр стиля	Описание
WS_BORDER	создание окна с рамкой
WS_CAPTION	добавление к окну с рамкой заголовка
WS_CHILD	создание дочернего окна. Не используется со стилем WS_POPUP
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_CAPTION, WS_MINIMIZEBOX, WS_MAXIMIZEBOX
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

ПРИЛОЖЕНИЕ 2. Прототипы обработчиков сообщений MFC-приложений

Таблица 5. Прототипы обработчиков сообщений

Сообщение	Прототип функции обработки сообщения
WM_CREATE	**afx_msg int OnCreate(LPCREATESTRUCT** *lpCreateStruct* **);** вызывается для извещения о завершении части этапа создания окна – от момента, когда окно создается функцией create, до момента возврата из функции и вывода окна на экран
WM_SHOWWINDOW	**afx_msg void OnShowWindow(BOOL** *bShow*, **UINT** *nStatus* **);** вызывается, когда окно (CWnd-объект) выводится или сворачивается методом ShowWindow, когда окно стиля overlapped максимально разворачивается, сворачивается в пиктограмму и т.п.
WM_SETFOCUS	**afx_msg void OnSetFocus(CWnd*** *pOldWnd* **);** вызывается при получении фокуса ввода
WM_KILLFOCUS	**afx_msg void OnKillFocus(CWnd*** *pNewWnd* **);** вызывается сразу перед потерей фокуса ввода
WM_SIZE	**afx_msg void OnSize(UINT** *nType*, **int** *cx*, **int** *cy* **);** вызывается после того, как размеры окна были изменены
WM_SIZING	**afx_msg void OnSizing(UINT** *nSide*, **LPRECT** *lpRect* **);** вызывается, извещая, что пользователь меняет размеры окна
WM_MOVE	**afx_msg void OnMove(int** x, **int** y **);** вызывается, когда пользователь переместил окно (CWnd-объект)
WM_MOVING	**afx_msg void OnMoving(UINT** *nSide*, **LPRECT** *lpRect* **);** вызывается, пока пользователь перемещает окно (**CWnd**-объект)
WM_TIMER	**afx_msg void OnTimer(UINT** nIDEvent **);** вызывается по истечении каждого интервала времени, специфицированного в методе установки таймера SetTimer
WM_HSCROLL WM_VSCROLL	**afx_msg void OnHScroll(UINT** *nSBCode*, **UINT** *nPos*, **CScrollBar*** *pScrollBar* **);** **afx_msg void OnVScroll(UINT** *nSBCode*, **UINT** *nPos*, **CScrollBar*** *pScrollBar* **);** вызывается, когда пользователь совершает щелчок по горизонтальной или вертикальной полосе прокрутки окна
WM_PAINT	**afx_msg void OnPaint( );** вызывается, когда Windows или приложение посылает запрос на перерисовку окна (методы UpdateWindow, RedrawWindow)
WM_CLOSE	**afx_msg void OnClose( );** система вызывает этот обработчик как сигнал, что окно (**CWnd-**объект) или приложение нужно завершить. Обработка по умолчанию вызывает DestroyWindow

WM_DESTROY	**afx_msg void OnDestroy( );** система вызывает этот обработчик, чтобы оповестить окно (**CWnd**-объект), что оно разрушается (OnDestroy вызывается после того, как окно убирается с экрана)
WM_CHAR	**afx_msg void OnChar(UINT** *nChar*, **UINT** *nRepCnt*, **UINT** *nFlags* **);** вызывается нажатием несистемной клавиши (до метода OnKeyUp и после OnKeyDown). Сообщает значение нажатой клавиши
WM_KEYDOWN	**afx_msg void OnKeyDown(UINT** *nChar*, **UINT** *nRepCnt*, **UINT** *nFlags* **);** вызывается нажатием несистемной клавиши
WM_KEYUP	**afx_msg void OnKeyUp(UINT** *nChar*, **UINT** *nRepCnt*, **UINT** *nFlags* **);** вызывается отжатием несистемной клавиши
WM_COMMAND	**virtual BOOL OnCommand(WPARAM** *wParam*, **LPARAM** *lParam* **);** вызывается, когда пользователь выбирает команду (выбирает пункт меню, нажимает кнопку и т.д.)
WM_RBUTTONDOWN, WM_LBUTTONDOWN	**afx_msg void OnLButtonDown(UINT** *nFlags*, **CPoint** *point* **);** вызывается, когда пользователь нажимает левую или правую кнопку “мыши”
WM_LBUTTONUP WM_RBUTTONUP	**afx_msg void OnLButtonUp(UINT** *nFlags*, **CPoint** *point* **);** вызывается, когда пользователь отжимает левую или правую кнопку “мыши”
WM_LBUTTONDBLCLK WM_RBUTTONDBLCLK	**afx_msg void OnLButtonDblClk(UINT** *nFlags*, **CPoint** *point* **);** вызывается двойным щелчком левой или правой кнопкой “мыши”
WM_MOUSEMOVE	**afx_msg void OnMouseMove(UINT** *nFlags*, **CPoint** *point* **);** вызывается, когда перемещается курсор “мыши”

ПРИЛОЖЕНИЕ 3. Текст типового каркаса MFC-приложения с “заглушками” для обработчиков

```
//=================================================
//              ТИПОВОЙ КАРКАС MFC-ПРИЛОЖЕНИЯ (ТКП)
//=================================================
// 1. Создать мастером Win32 Application “пустой” каркас (тип empty).
// 2. Создать в каркасе пустой файл *.cpp.
// 3. Вставить текст ТКП в файл *.cpp.
// 4. Включить в главном меню в пункте Project-Settings, на вкладке General
//    режим – “use MFC in Static library”.
//=================================================
#include       <afxwin.h>
//#include     <afxext.h>
//#include     <afxdisp.h>
//#include     <afxdtctl.h>
//#ifndef      _AFX_NO_AFXCMN_SUPPORT
   //#include  <afxcmn.h>
//#endif
#include       <iostream>
using namespace std;

// ===== класс главного ОКНА приложения ==================
class WINDOW : public CFrameWnd
{
public:
        WINDOW ();
      // afx_msg void  OnLButtonDown (UINT flags, CPoint loc);
      // afx_msg void  OnPaint ();
      // afx_msg void On<ИмяОбработчика>( < ПараметрыОбработчика > );
        DECLARE_MESSAGE_MAP()
};

// ===== конструктор Создание окна
WINDOW::WINDOW ()
{
      Create ( NULL, "Главное окно MFC-приложения” );
}

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

// ===== метод Обработчик сообщения
// afx_msg void WINDOW :: OnPaint ()
// { CPaintDC  dc (this); < ФРАГМЕНТ_ПОЛЬЗОВАТЕЛЯ >  };

// ===== метод Обработчик сообщения
//afx_msg void CMainWin::On< ИмяОбработчика >( < ПараметрыОбработчика > )
```

```
// { CClientDC  dc (this); < ФРАГМЕНТ_ПОЛЬЗОВАТЕЛЯ > };

// ===== очередь сообщений главного окна
BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
    // ON_WM_PAINT()
    // ON_WM_LBUTTONDOWN()
    // ON_WM_<ИмяСообщения>( [ < Параметры > ] )
END_MESSAGE_MAP()

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

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

// ===== создание экземпляра приложения ====================
APPLICATION    TheApplication;
```

ПРИЛОЖЕНИЕ 4. Текст типового каркаса MFC-приложения

Примерный текст ТКП с минимальным набором средств и без обработчиков сообщений приведен ниже.

```
//=================================================
//              ТИПОВОЙ КАРКАС MFC-ПРИЛОЖЕНИЯ (ТКП)
//=================================================
#include   <afxwin.h>
#include   <iostream>
using namespace std;

class WINDOW : public CFrameWnd
{
public:
        WINDOW ();
        DECLARE_MESSAGE_MAP()
};
WINDOW::WINDOW ()
{
      Create ( NULL, "Главное окно типового каркаса MFC-приложения”);
}
BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
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;
```

ОГЛАВЛЕНИЕ

Учебное издание

Муравьев Геннадий Леонидович, Мухов Сергей Владимирович,
Савицкий Юрий Викторович, Хвещук Владимир Иванович

методическое пособие

“ ОСНОВЫ СОЗДАНИЯ WINDOWS-ПРИЛОЖЕНИЙ В СИСТЕМЕ MICROSOFT VISUAL STUDIO C++ на базе библиотеки MFC”, Часть 1

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 …