МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра интеллектуальных информационных технологий

# Методические указания

## "РАЗРАБОТКА ПРИЛОЖЕНИЙ НА БАЗЕ КАРКАСА ДОКУМЕНТ-ВИД"

БРЕСТ 2011

УДК 681.3 (075.8)

Методическая разработка предназначена для знакомства с основами создания пользовательских Windows-приложений в системе Microsoft Visual Studio C++ на базе универсального каркаса документ-вид, поддерживаемого средствами библиотеки MFC. Рассматриваются общие принципы создания и просмотра документов, а также примеры разработки простейших приложений.

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

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

# 1. ОБЩИЕ СВЕДЕНИЯ О КАРКАСНОМ ПРОГРАММИРОВАНИИ

Для упрощения и ускорения программирования приложений современные системы программирования предлагают каркасы (типовые каркасы приложений) – заготовки текстов программ, ориентированных как на тип поддерживаемого интерфейса, так и на предполагаемый круг решаемых задач. Интегрированная среда разработки фирмы Microsoft (Microsoft Developer Studio) - универсальная среда, поддерживающая различные языковые средства разработки, включая C++. Поддерживает процессы 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), мастер классов ClassWizard (для создания шаблонов новых производных классов и их методов), редакторы ресурсов (для создания меню, диалоговых окон, элементов управления и т.п.).

Объектно-ориентированная библиотека MFC (Microsoft Foundation Class Library) направлена на существенное упрощение работы с прикладным программным интерфейсом (API) Windows, который включает сотни разрозненных API функций, за счет их структуризации и поддержки каркасов. Практически классы MFC представляют собой каркас, на основе которого можно писать программы для ОС Windows. Соответственно технологически процесс программирования в MFC реализуется как "каркасное" программирование. Результатом применения этого стиля в системе программирования Visual Studio являются оконные MFC-приложения как "статичного", так и "динамического" типов.

Большинство классов библиотеки MFC являются производными от базового класса CObject ("объект"). Названия всех классов и шаблонов классов библиотеки MFC начинаются с заглавной буквы C. Имена переменных, входящих 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.

Фрагмент соответствующей диаграммы классов приведен на рисунке 1. Здесь класс CCmdTarget является основой для формирования архитектуры приложения (двух ее компонентов – интерфейсного класса и класса, управляющего 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 окон, в том числе, окон с рамкой класса CFrameWnd, диалоговых окон класса CDialog, специализированных окон семейства элементов управления (ЭУ), класса видов CView; - класс CDocument служит для организации работы с данными - документами.

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.

Библиотека MFC поддерживает разнообразные каркасы. Базовые каркасы, образующие "ядро" программирования в MFC, представлены рисунком 2. Это типовой каркас приложения (ТКП) типа "одиночный документ" с интерфейсом SDI (Single Document Interface) и типовой каркас приложения (ТКП ДВ) типа "документ-вид" с интерфейсом SDI, MDI (Multiple Document Interface).

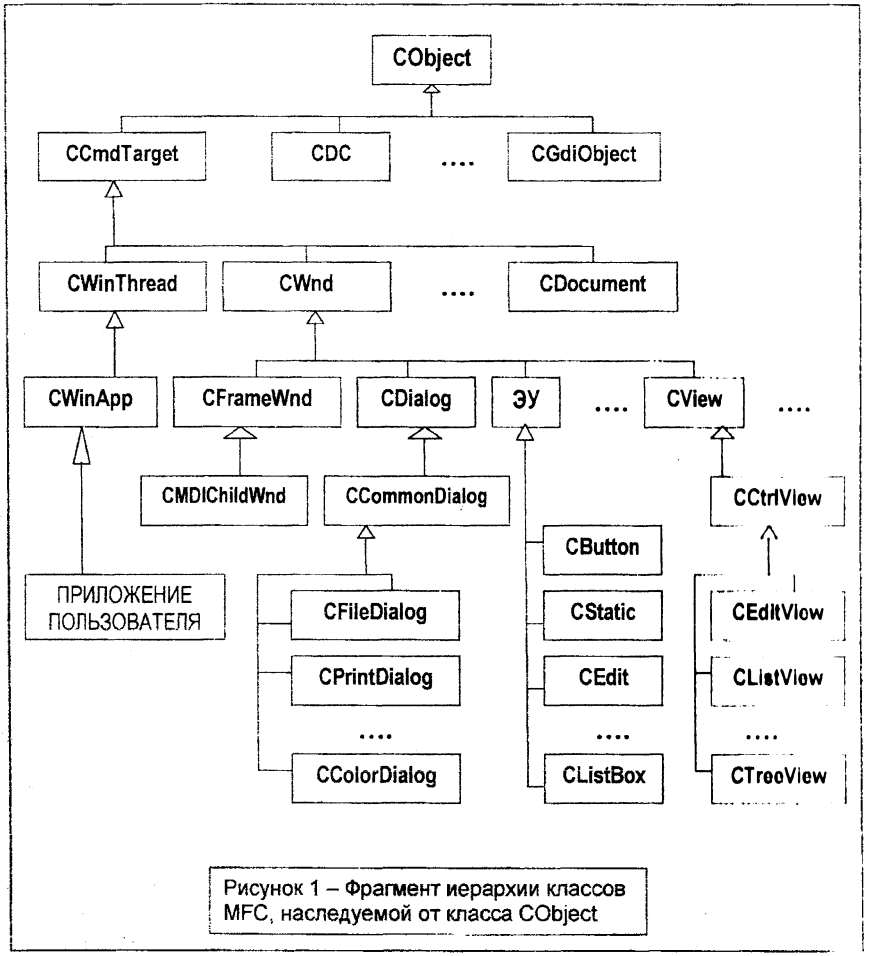



Рисунок 1 – Фрагмент иерархии классов MFC, наследуемой от класса CObject

Первый каркас использует класс "окно с рамкой" (CFrameWnd) в роли главного для организации всех функций по обработке данных (документов), включая их хранение, визуализацию (ТКП ГО) или класс "диалоговое окно" в роли главного (ТКП ДО). Соответственно здесь создается два основных объекта: объект приложения и объект окна. Иногда в литературе такой каркас называют архитектурой создания приложений. Соответствующие приложения строятся почти по принципу обычных Windows-приложений без использования MFC. Однако MFC автоматизирует многие типовые действия, типовое поведение приложений, что и упрощает разработку. В прило-


Рисунок 2 – Типовые (автоматические) каркасы MFC приложений

жениях на базе ТКП сами данные, способ их представления и методы обработки тесно связаны между собой. Данные (документ), а также методы их обработки, методы их отображения как правило, хранятся как члены объекта окна, то есть реализуются в одном классе. В то время как концептуально данные могут быть отделены от способа их представления.

В большом спектре автоматизируемых задач целесообразным является разделение самих данных и типовых действий над ними (загрузка, выгрузка), методов, способов их отображения. Второй 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. В литературе соответствующие приложения называют приложениями с архитектурой создания и просмотра документов.

# 2. ТИПОВОЙ КАРКАС ПРИЛОЖЕНИЯ ДОКУМЕНТ-ВИД

## 2.1. Общая характеристика каркаса

Основные разновидности каркаса документ-вид представлены рисунком 3. Создание приложений на базе ТКП ДВ в Visul Studio поддерживается соответствующим мастером AppWizard, позволяющим получать каркасы для разных вариантов применения приложений в соответствии с рисунком 3. При использовании автоматических средств генерации каркаса соответствующие особенности каркаса могут быть выбраны в процессе генерации. Как видно из классификации, каркас обеспечивает широкий круг возможностей, включая работу с разнообразными базами данных, автоматическую поддержку базовых действий пользователя по созданию, загрузке, сохранению документов, сосредоточенных, как правило, в пункте меню File, возможность работы с составными документами, поддержку многодокументной и многооконной работы.

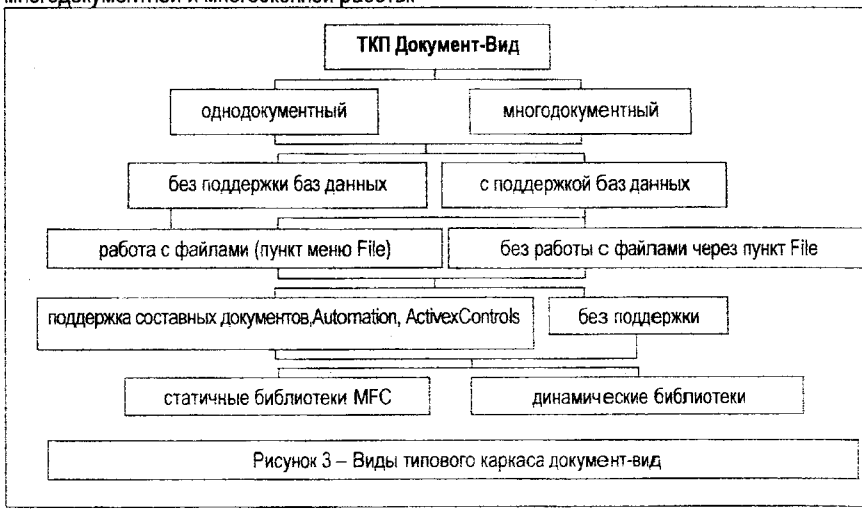



Рисунок 3 – Виды типового каркаса документ-вид

Типовой состав интерфейса, вид меню и окон представлены на рисунках 4-7. Это главное окно (поддерживаемое классом CFrameWnd) с типовым меню, панелью базовых инструментов, строкой подсказки, системным меню, клиентской областью для отображения текстовой и графической информации. При этом клиентская область поддерживается классом CView и конструктивно перекрывает клиентскую область, поддерживаемую

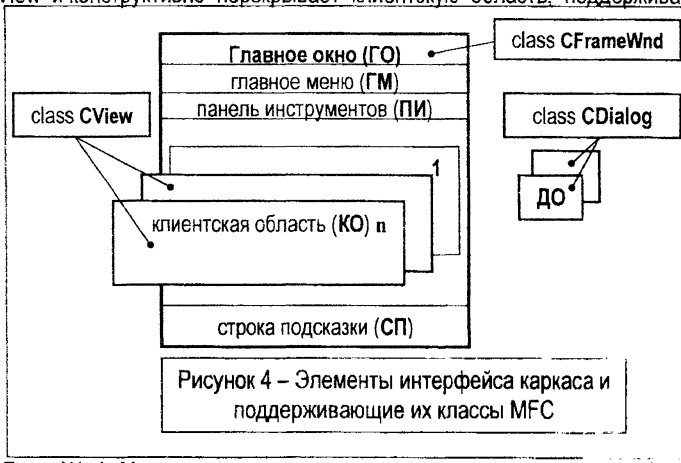



Рисунок 4 – Элементы интерфейса каркаса и поддерживающие их классы MFC

классом CFrameWnd. Меню поддерживает типовые пункты, правильная работа которых требует дополнительной настройки каркаса и допрограммирования. При необходимости интерфейс оснащается диалоговыми окнами (ДО) и др. элементами.

Функционирование приложений представлено рисунком 8, где представлены основные состояния приложения и изображен соответствующий граф переходов, напоминающий граф для приложений с ТКП. Общая схема функционирования представлена

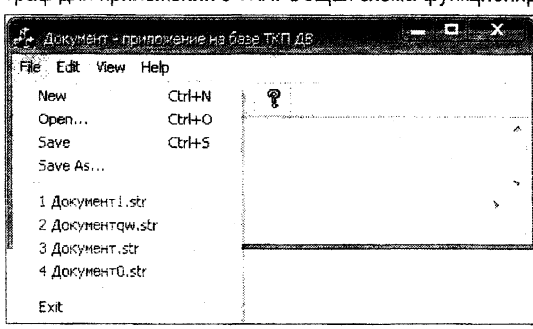
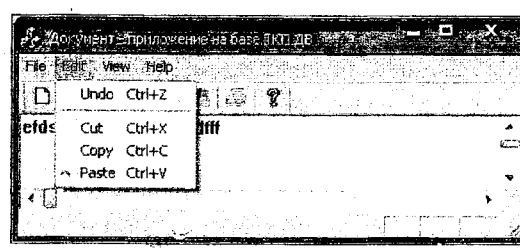



Рисунок 5 – Вид интерфейса ТКП ДВ

Рисунок 6 – Вид интерфейса ТКП ДВ



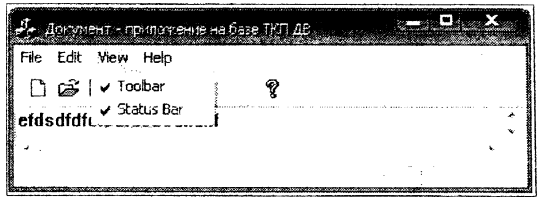



Рисунок 7 – Вид интерфейса ТКП ДВ

рисунком 9, отображающим как состав приложения, так и взаимодействие основных частей. Приложение здесь строится на базе так называемого шаблона документов, изображенного на рисунке 12, объединяющего такие понятия, как документ, вид документа, окно (главное), графические ресурсы, включая ресурс-строку, задающую системные настройки приложения. Здесь данные (т.е. документ) отделены от своего представления (т.е. вида документа или области просмотра). Это достигается использованием специальной иерархии классов.

Документ - это любые виды данных, блок данных, отображающих текущее состояние документа и связанные с программой. Важнейшая черта документа в ТКП ДВ: динамическое хранение и автоматическая загрузка-выгрузка (сохранение) во внешней памяти. В соответствующем классе (класс документ), предназначенном для создания документа, инкапсулируются данные и методы как загрузки, подготовки документа для работы, так и методы для выгрузки, сохранения документа по завершении сеанса работы. Класс не имеет средств класса CWnd. Как правило, все, что делается с документом без оконного интерфейса, например, очистка содержимого и т.д., лучше производить напрямую методами-обработчиками класса документ и сообщать об изменениях в окна просмотра документа, используя методы SetModifiedFlag( ) и UpdateAllViews( NULL ).

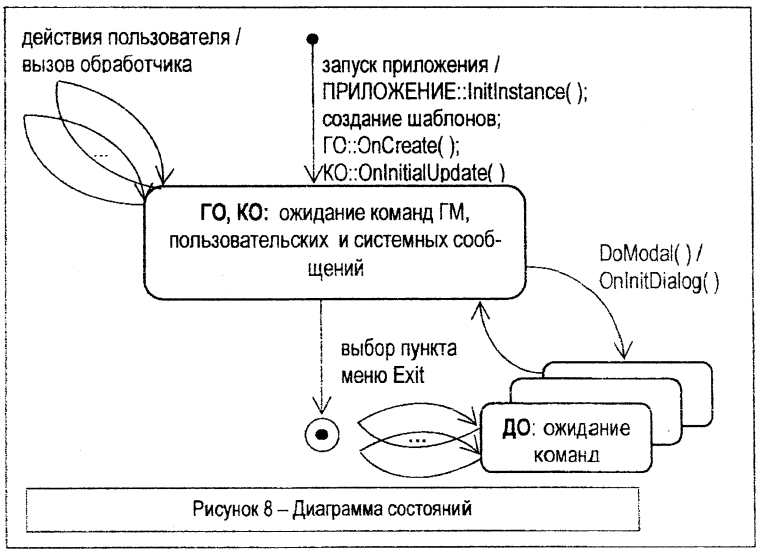



Рисунок 8 – Диаграмма состояний

Вид, область просмотра (view) соответствует виду документа, содержит физическое представление данных. При этом отображение документа в области просмотра может быть существенно разным. Областью просмотра наряду с наиболее часто используемыми экранными представлениями могут быть и любые другие отображения документа, например, его твердые копии. А в соответствующем классе (класс вида, просмотра), предназначенном

для организации просмотра документа, инкапсулируются методы отображения этих данных, а также методы управления изменением документа пользователем. Класс наследует средства класса CWnd.

На практике возникают различные соотношения между понятиями документ и вид (облик) документа. Например, для документа могут потребоваться разные способы отображения. Один и тот же вид может использоваться при работе с несколькими документами. В Visual Studio в архитектуре ТКП ДВ для создания и просмотра документов приложения могут быть использованы интерфейсы SDI (Single Document Interface - интерфейс единого документа) или MDI (Multiple Document Interface - интерфейс составного документа). При этом в качестве стандарта фирма Microsoft рекомендует использовать интерфейс SDI и соответствующий тип документов.

Как правило, все, что требуется для работы с документом, в том числе через оконный интерфейс, например, изменение, редактирование содержимого и т.д., лучше производить методами-обработчиками данного класса, с использованием окон, при необходимости также извещая систему об изменениях, используя вышеуказанные методы и метод Invalidate.

В ТКП ДВ объект документа порождается от класса CDocument, а объект области просмотра порождается от класса CView (или производных от него классов). При этом физически область просмотра, создаваемая классом CView, перекрывает главное окно приложения, формируемое классом CFrameWnd.

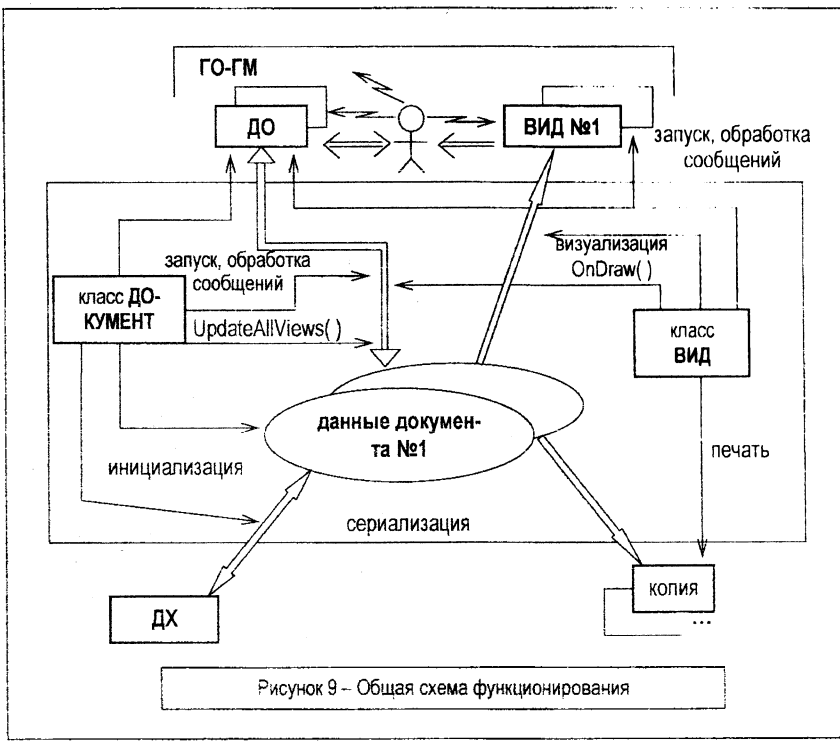



Рисунок 9 – Общая схема функционирования

Характерные черты ТКП ДВ: - управление хранением документов, сериализацией, идентификацией (serialization) документов; - динамическое создание объектов.

В архитектуре ТКП ДВ сериализация - это процесс сохранения и восстановления текущего состояния документа, что позволяет автоматизировать хранение документов в файлах дисковой памяти. Механизм идентификации реализован в классе CDocument.

В программах на базе ТКП ДВ базовые объекты – объекты каркаса являются динамическими. Так, объекты главного окна, документа и области просмотра создаются динамически во время выполнения программы, объекты создаются динамически, например, при загрузке данных с диска. Для этого применяются специальные макрокоманды.

Макрокоманда

**DECLARE_DYNCREATE(*ИмяКласса*)**

используется в объявлении класса для указания того, что объекты класса могут создаваться динамически.

Макрокоманда

**IMPLEMENT_DYNCREATE (*ИмяКласса*, *ИмяБазовогоКласса*)**

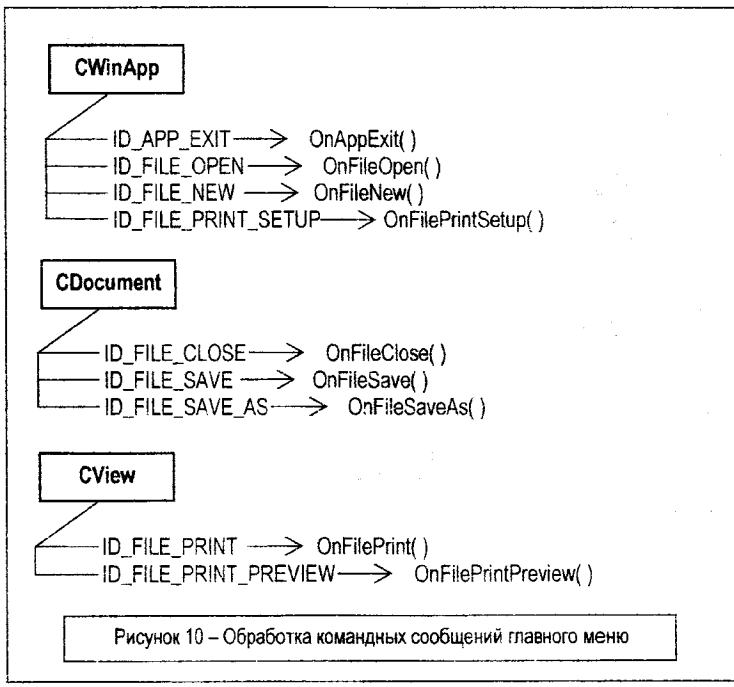



Рисунок 10 – Обработка командных сообщений главного меню

используется в самом программном файле. Здесь ИмяКласса идентифицирует класс, для которого разрешается динамическое создание объектов, а параметр ИмяБазовогоКласса указывает соответствующий класс библиотеки MFC, используемый в качестве базового. После вызова обеих макрокоманд имя класса может быть указано в качестве параметра макроса

**RUNTIME_CLASS (*ИмяКласса*)** ,

который возвращает указатель на структуру типа CRuntimeClass, связанную с заданным классом. Здесь параметр ИмяКласса содержит имя динамического класса, которое ранее было указано в макросе DECIARE_DYNCREATE, а полученный указатель используется далее при создании шаблона документа.

Включение макрокоманды DECLARE_DYNCREATE в порождаемый класс является первым из двух действий, которые необходимо выполнить, чтобы разрешить динамическое создание объектов данного класса. На следующем шаге необходимо включить в программу макрос IMPLEMENT_DYNCREATE. После этого классы могут быть использованы для создания шаблона документа.

В приложениях на базе ТКП ДВ автоматически поддерживаются такие типовые действия, как открытие и закрытие файлов, сохранение и печать документов, поэтому в MFC определен ряд стандартных идентификаторов (называющихся командными), которые автоматически посылаются программе и позволяют идентифицировать выполняющуюся операцию. Некоторые из стандартных идентификаторов обрабатываются автоматически, поэтому соответствующие им функции не нужно включать в очереди сообщений. Для остальных это необходимо. Существуют также другие системные идентификаторы, чьи обработчики вызываются автоматически. Однако для этого соответствующие сообщения должны включаться в карты сообщений приложения. Более того, если макрокоманды для данных идентификаторов не поместить в очередь сообщений, то соответствующие им элементы меню могут оказаться неактивными. Список основных идентификаторов вместе с соответствующими им обработчиками, а также необходимость включения макросов в карту сообщений представлены на рисунке 10 и в таблице 1.

Таблица 1 – Команды главного меню

| пункт меню | подпункт меню | идентификатор подпункта меню | автоматическая обработка |
|---|---|---|---|
| File | Close | ID_FILE_CLOSE | |
| | New | ID_FILE_NEW | |
| | Open | ID_FILE_OPEN | |
| | Save | ID_FILE_SAVE | да |
| | Save As | ID_FILE_SAVE_AS | да |
| | Print | ID_FILE_PRINT | |
| | Print Preview | ID_FILE_PREVIEW | |
| | Print Setup | ID_FILE_PRINT_SETUP | |
| | Exit | ID_APP_EXIT | да |
| Edit | Undo | ID_EDIT_UNDO | |
| | Cut | ID_EDIT_CUT | |
| | Copy | ID_EDIT_COPY | |
| | Paste | ID_EDIT_PASTE | |
| | Open | ID_FILE_OPEN | |

При запуске приложения автоматически создается окно просмотра документа и новый документ (экземпляр документа). Документ автоматически инициализируется методом OnNewDocument. Активизируется метод OnDraw.

При выборе пункта меню File-New ТКП ДВ создает новое окно просмотра документа и новый, не связанный с другими, документ (экземпляр документа). Документ автоматически инициализируется методом OnNewDocument. OnNewDocument. Активизируется метод OnDraw.

При выборе пункта меню Window-New window ТКП ДВ создает новое окно просмотра текущего документа, помечаемое как ИмяОкна: НомерОкна.

При выборе пункта меню File-Save ТКП ДВ выводит окно Сохранить как для нового документа (еще не сохраненного) иначе сохранение текущего варианта производится автоматически.

При выборе пункта меню File-SaveAs ТКП ДВ выводит окно Сохранить, как и далее сохранение производится автоматически.

При обновлении документа и генерации соответствующих сообщений (выполнении методов Invalidate для обновления конкретного вида, UpdateAllViews для обновления всех видов, запускающих обработчик перерисовки окон просмотра OnUpdate) пункта меню File-New ТКП ДВ создает новое окно просмотра документа и новый, не связанный с другими, документ (экземпляр документа).

При выборе пункта меню File-Open ТКП ДВ выводит окно Открыть с предложением сохранения текущего документа, если таковой имеется.

При выборе пункта меню Exit ТКП ДВ при наличии модифицированного, но не сохраненного в последней версии документа, выводит окно Сохранить, как и далее сохранение производится автоматически.

### 2.2. Основные классы каркаса

В таком приложении используются как минимум объекты пяти классов, служащих как для организации оконного, графического интерфейса и обработки сообщений, так и для реализации функций самого приложения по запуску, завершению, обработке документов и их отображению.

Соответствующая диаграмма классов представлена на рисунках 11, 13. Это класс ОКНО (производный от CFrameWnd), который служит базовым для создания интерфейса приложения, поддерживает масштабируемость, меню, панель инструментов и т.п. Класс ДОКУМЕНТ (производный от CDocument) обеспечивает хранение и редактирование данных, их сериализацию. Класс ВИД (производный от CView) управляет отображением данных.


Рисунок 11 – Диаграмма классов

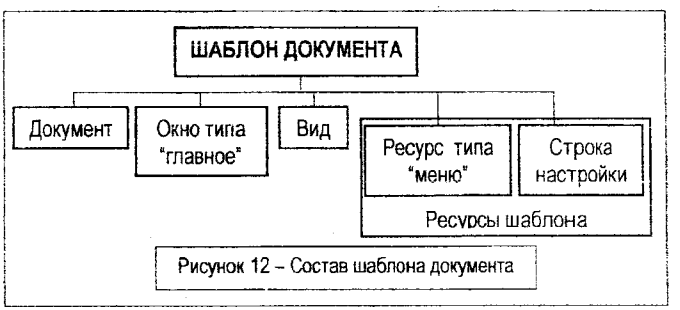



Рисунок 12 – Состав шаблона документа

На базе перечисленных выше пользовательских классов, обеспечивающих работу с документом, областью просмотра и главным окном создается шаблон документа (ШД), предназначенный для связывания всех перечисленных классов в единый каркас (рисунок 14).

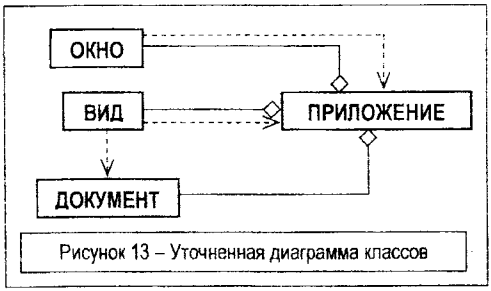


Рисунок 13 – Уточненная диаграмма классов

При этом можно создавать шаблоны как составных (когда допустима одновременная работа как со многими экземплярами и обликами-видами документа так и с документами разного типа), так и единых документов (когда допустима работа только с одним документом), использующие MDI или SDI интерфейсы. Шаблон документа с SDI-интерфейсом является объектом класса CSingleDocTemplate.

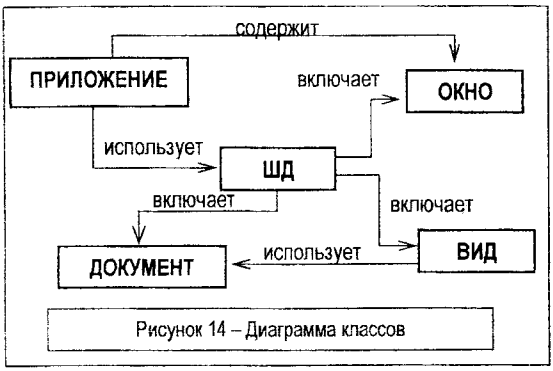


Рисунок 14 – Диаграмма классов

В приложении за создание всех объектов и связывание их в виде шаблона документа (рисунки 14-15) отвечает объект класса ПРИЛОЖЕНИЕ (производный от CWinApp), который 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.

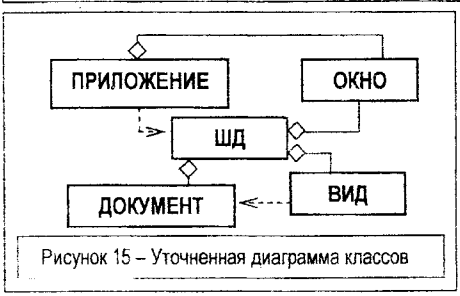


Рисунок 15 – Уточненная диаграмма классов

Как видно из рисунка 15, шаблон документа агрегирует классы ОКНО, ВИД, ДОКУМЕНТ. Создаваемые динамически соответствующие объекты используются приложением для обеспечения требуемой функциональности пользовательского приложения. Соответственно ПРИЛОЖЕНИЕ использует шаблон документа, в том числе, для подготовки приложения к работе.

### 2.3. Шаблон документа. Инициализация каркаса

Создание шаблона реализуется при создании объекта приложения путем запуска его метода InitInstance (см. рисунок 16), где и создается объект ШД. При этом используется конструктор

**CSingleDocTemplate:: CSingleDocTernplate (UINT** *ID_Ресурсов_ШД*, **CRuntimeClass** * *Документ*, **CRuntimeClass** * *Окно*, **CRuntimeClass** **Вид*) .

Здесь *ID_Ресурсов_ШД* - идентификатор ресурсов шаблона (меню, таблицы быстрого доступа, пиктограммы документа, а также строки настройки программы). Параметр Документ - указатель на класс документа, Окно - указатель на класс главного окна, Вид - указатель на класс области просмотра. Их получают с помощью макроса RUNTIME_CLASS, т.е. указанные классы предварительно должны быть объявлены динамическими.

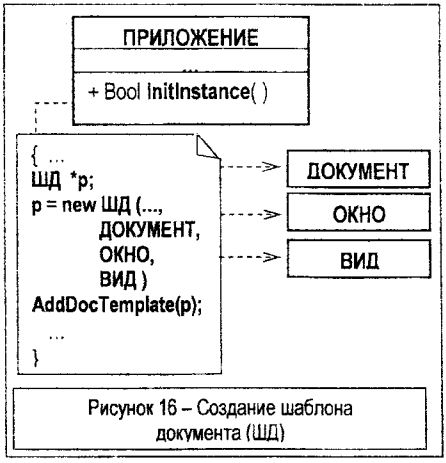



Рисунок 16 – Создание шаблона документа (ШД)

Для создания пользовательского класса приложения в качестве базового используется класс MFC CWinApp. В пользовательском классе необходимо описать метод InitInstance ( **virtual BOOL CWinApp:: InitInstance( )** ), вызываемый автоматически при запуске приложения и выполняющий роль конструктора и отвечающий за инициализацию приложения. Метод аналогичен используемому в ТКП, однако здесь при аналогичности общих действий он реализует иную, более сложную последовательность действий для создания ШД. Примерный текст метода приведен ниже

```
BOOL ПРИЛОЖЕНИЕ::InitInstance ( )
{
    CSingleDocTemplate *УказательШД = new CSingleDocTemplate(ID_Ресурсов_ШД,
                                          RUNTIME_CLASS( ДОКУМЕНТ ),
                                          RUNTIME_CLASS( ОКНО ),
                                          RUNTIME_CLASS( ВИД ) );
    AddDocTemplate( УказательШД );
    EnableShellOpen( );
    RegisterShellFileTypes( );
    CCommandLineInfo CLInfo;
    ParseCommandLine( CLInfo );
    if ( ! ProcessShellCommand ( CLInfo ) )
      return FALSE;
    // m_pMainWnd -> ShowWindow( SW_SHOW );
    return TRUE;
}
```

Здесь вначале создается УказательШД и сам ШД, конструируемый с помощью CSingleDocTemplate. Созданный шаблон документа, заданный указателем УказательШД, добавляется к списку шаблонов, поддерживаемых приложением, с помощью функции

**void CWinApp:: AddDocTemplate (CDocTemplate** **УказательШД*).

Для обеспечения пользователю возможности запуска приложения по щелчку на одном из его документов вызывается функция EnableShellOpen

**void CWinApp::EnableShellOpen( )** .

Для регистрации созданного типа документа в системной базе регистрации в соответствии с описанием ресурса-строки, заданной при создании шаблона документа, вызывается функция RegisterShellFileTypes

**void CWinApp:: RegisterShellFileTypes( )** .

Далее выполняется обработка аргументов командной строки, которые можно задать при запуске Windows-приложения с помощью команды Run или из командной строки. В приложениях, построенных на основе ТКП ДВ, в командной строке обычно указывается имя документа, открываемого при запуске приложения. Информация из командной строки инкапсулируется в классе CCommandLineInfo. Синтаксический разбор командной строки, оценка ее корректности и обработка реализуются методами (при этом, если метод возвращает ноль, то функция InitInstance( ) должна вернуть значение FALSE)

**CWinApp::ParseCommandLine( ) ,**
**CWinApp:: ProcessShellCommand ( ) .**

Формат строки описания настроек представлен на рисунке 17. Здесь ЗО – заголовок окна; НД – название документа по умолчанию (при отсутствии в качестве первичного имени документа системой будет предлагаться "Untitled"); ТД – тип документа (расширение), если приложение предусматривает работу с несколькими типами; ФФ - описание фильтра файлов, используемого в типовых окнах загрузки-сохранения документов (список "тип файлов"); РД – расширение файлов для документов данного типа; ИТСР – идентификатор типа файлов для базы системной регистрации; ИмТСР – имя типа файлов для базы системной регистрации. Указанные поля приводятся в одной строке в том же порядке. В качестве разделителя полей используется символ "новая строка" ("\n"). Этот же символ замещает отсутствующее поле строки. Сам ресурс-строка описывается в файле ресурсов оператором STRINGTABLE как

```
STRINGTABLE
{
    ID_Ресурсов_ШД "строка"
}
```

и может быть задан средствами редактора ресурсов. Здесь ID_Ресурсов_ШД - идентификатор строки (как правило, совпадающий с идентификатором ресурсов шаблона документа – меню и т.д.). Например, строка IDR_FRAMEWIN

Document-View\nDocName\nDocType\nDoc Type (*.dvi)\n.dvi\nDVTest\nDV Test ,

где "Document-View" - заголовок окна; для новых документов будет использоваться имя "DocName"; название типа документа "DocType". В списке "тип файлов" файлам данных документов будет соответствовать элемент "Doc Type (*.dvi)". Файлы будут иметь расширение dvi. В системной базе регистрации документам будет соответствовать идентификатор "DVTest" и тип файлов "DV Test ".

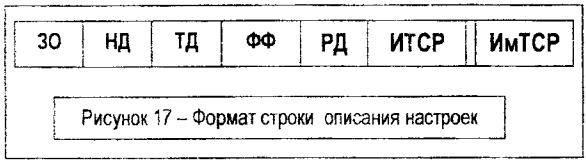



Рисунок 17 – Формат строки описания настроек

### 2.4. Структура каркаса

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. Ниже рассматривается и комментируется примерный текст ТКП ДВ, создаваемый "вручную". Соответствующий текст с "заглушками" для обработчиков приведен в ПРИЛОЖЕНИИ 1. В его тексте жирным шрифтом выделены идентификаторы, которые может менять пользователь по своему усмотрению. Закомментированные обработчики также добавляются и доопределяются пользователем при необходимости.

Строка настройки **IDR_MENU**.

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

// ==========================
class ОКНО : public CFrameWnd
{
    DECLARE_DYNCREATE(ОКНО)
public:
    ОКНО ( );
     DECLARE_MESSAGE_MAP( )
};

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

// ============================
class ДОКУМЕНТ: public CDocument
{
    DECLARE_DYNCREATE( ДОКУМЕНТ )
     // описание данных документа
public:
    ДОКУМЕНТ ( );
    BOOL OnNewDocument( );
    void Serialize(CArchive &Архив);
     // прототипы обработчиков документа
     // afx_msg void  ОбработчикДокумента ( );
    DECLARE_MESSAGE_MAP( )
};

// ==================
class ВИД: public CView
{
    DECLARE_DYNCREATE( ВИД )
public:
    void OnDraw(CDC *КУ);
     // прототипы обработчиков данных документа
     // afx_msg void  ОбработчикВида ( );
    DECLARE_MESSAGE_MAP( )
};

// ==============================
```

**ПРИЛОЖЕНИЕ моеПриложение;**
IMPLEMENT_DYNCREATE( **ОКНО**, CFrameWnd )
IMPLEMENT_DYNCREATE( **ВИД**, CView )
IMPLEMENT_DYNCREATE( **ДОКУМЕНТ**, CDocument )

BEGIN_MESSAGE_MAP( **ОКНО**, CFrameWnd )
**// макрокоманды включения обработчиков**
END_MESSAGE_MAP( )

BEGIN_MESSAGE_MAP(**ПРИЛОЖЕНИЕ**, CWinApp)
**// макрокоманды включения обработчиков**
END_MESSAGE_MAP( )

BEGIN_MESSAGE_MAP(**ДОКУМЕНТ**, CDocument)
**// макрокоманды включения обработчиков**
END_MESSAGE_MAP( )

BEGIN_MESSAGE_MAP(**ВИД**, CView)
**// макрокоманды включения обработчиков**
END_MESSAGE_MAP( ) .

### 2.5. Построение каркаса в Visual Studio C++

Для построения каркаса и приложения на базе ТКП ДВ необходимо:

1. Описать производные классы, предназначенные для создания приложения, главного окна, документа и области просмотра.
2. Разрешить динамическое создание объектов для классов главного окна, документа и области просмотра.
3. Создать шаблон документа, объединяющий классы главного окна, документа и области просмотра.
4. Описать обработку командной строки приложения.
5. Переопределить и дописать необходимые функции, например, Object::Serialize( ), CView::OnDraw( ) и т.д.

Соответственно в Visual Studio C++ необходимо:

1. Создать "пустой" каркас (тип empty) мастером Win32 Application. Для этого выполнить команду File-New, в окне New на вкладке Projects выбрать мастер Win32 Application, задать режим "An empty project".
2. Создать в каркасе пустой файл *.cpp. Для этого выполнить команду File-New, в окне New на вкладке Files выбрать тип файла C++ Source File, задать его (пользовательское) имя.
3. Вставить текст ТКП ДВ в файл *.cpp. Для этого открыть указанный файл и скопировать в него текст каркаса (см. ПРИЛОЖЕНИЕ 1).
4. Включить в главном меню в пункте Project-Settings, на вкладке General режим - "use MFC in Static library" (или подключить динамическую библиотеку).
5. Создать ресурсный файл *.rc.
6. Создать ресурс-меню (с идентификатором, например, IDR_MENU) в редакторе меню хотя бы с одним "фиктивным" пунктом.
7. Создать ресурс-строку (с тем же идентификатором, например, IDR_MENU) в редакторе ресурсов с начальным заполнением, например, из шаблона

Document-View\nDocName\nDocType\nDoc Type (*.dvi)\n.dvi\nDVTest\nDV Test .

8. Добавить, редактировать необходимые ресурсы, редактировать главное меню.
9. Добавить, редактировать члены классов каркаса, в первую очередь классов документа и вида.

# 3. ОСНОВЫ СОЗДАНИЯ ПРИЛОЖЕНИЙ НА БАЗЕ КАРКАСА ДОКУМЕНТ-ВИД

## 3.1. Создание пользовательского класса для работы с документом

Класс, предназначенный для работы с документом, порождается от класса CDocument (рисунок 18). Для него должно быть разрешено динамическое создание объектов с помощью макроса DECLARE_DYNCREATE. Обычно этот класс обрабатывает все сообщения, связанные с изменением состояния документа. Сам базовый класс CDocument (рисунок 19) содержит готовые методы, которые можно использовать или переопределять.

Некоторые из методов практически всегда необходимо дописывать (рисунок 20). Это функция virtual BOOL CDocument::OnNewDocument( ), которая вызывается при создании нового документа. При ее описании необходимо сначала вызывать базовую функцию из класса CDocument. Функция должна возвращать ненулевое значение при успешном завершении и ноль - в противном случае.

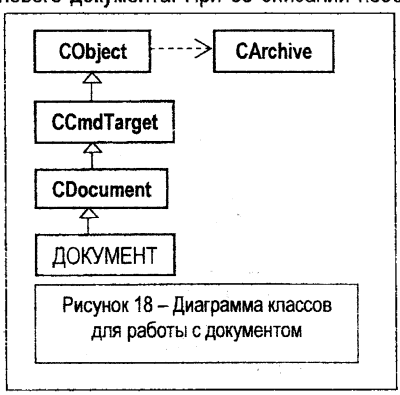



Рисунок 18 – Диаграмма классов для работы с документом

Функция Serialize(), которая является членом класса CObject, также должна быть переописана для поддержки процессов сериализации.

При работе в MFC при создании приложений, использующих файловый ввод-вывод, возможны следующие варианты:

- создание приложения без использования объектов класса документ, при этом файловый ввод-вывод реализуется стандартными библиотеками ввода-вывода С либо методами класса CFile MFC;

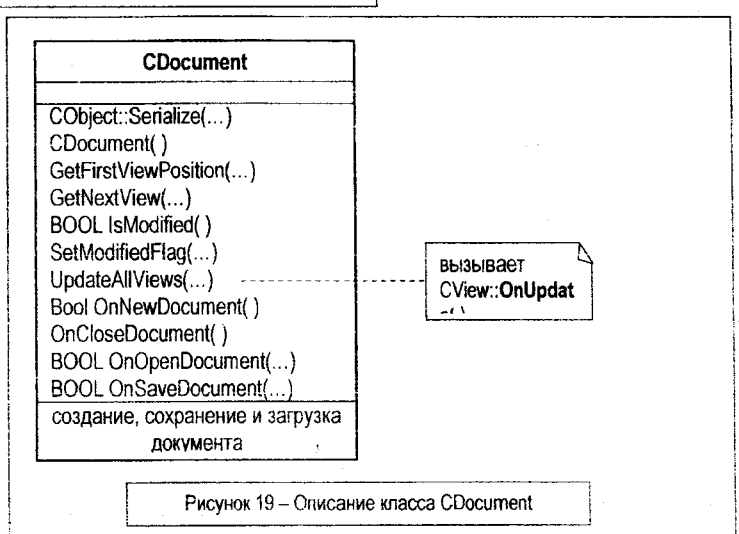



Рисунок 19 – Описание класса CDocument

- создание приложения с использованием объектов класса документ, при этом файловый ввод-вывод наряду со стандартными библиотеками ввода-вывода С либо методами класса CFile MFC реализуется и посредством механизмов сериализации.

Как указывалось ранее, ТКП ДВ автоматизирует процессы сохранения и загрузки документов, размещаемых во внешней памяти. Это реализуется на базе метода Serialize класса CObject

**virtual void CObject::Serialize (CArchive &*Arch*)** ,

который вызывается автоматически каждый раз в ситуациях, когда требуется сохранение или загрузка документа. Например, при выборе пунктов меню типа Сохранить, Сохранить как, Открыть, при запуске приложения и т.д.
Здесь параметр Arch является объектом типа CArchive, определяющим поток архивирования данных.

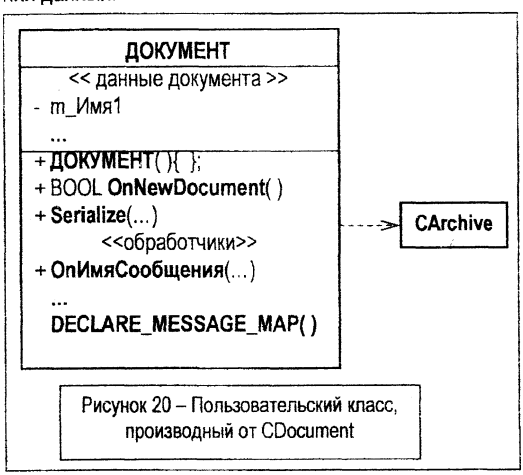



Рисунок 20 – Пользовательский класс, производный от CDocument

Функция Serialize должна переопределяться в классе документа. Соответственно и пользовательская функция должна записывать и читать документ из архивного потока, определяемого параметром Arch, используя для записи и загрузки документов внутри функции Serialize операторы "<<" и ">>".

Непосредственные действия по автоматизации функций работы с файлами в MFC поддерживаются классом CArchive (рисунок 21), который упрощает операции, связанные с сохранением данных - документов во внешней памяти. Класс включает базовые средства поддержки объектной модели времени выполнения.

Поскольку один метод Serialize вызывается как при сохранении, так и при загрузке документа, то определение конкретной ситуации выполняется методами IsStoring и IsLoading класса CArchive:

**BOOL CArchive:: IsLoading ( ) const;**
**BOOL CArchive:: IsStoring () const;** .

Функция IsLoading возвращает ненулевое значение, если документ загружается. Соответственно функция IsStoring возвращает ненулевое значение при сохранении документа.

В классе CArchive также содержатся перегруженные версии операторов ввода-вывода ("<<" и ">>"), предназначенные для работы как со стандартными типами данных, так и с типами классов, определенных в библиотеке MFC. В пользовательском применении в качестве левого операнда этих операторов должен использоваться параметр ArchOb.

Каждый раз при модификации документа в процессе работы с ним необходимо вызывать метод UpdateAllViews

**void CDocument:: UpdateAllViews( CView **Источник*, LPARAM *lParam* = 0,**
**CObject **Объект* = NULL)** ,

посылающий всем областям просмотра, связанным с данным документом, сообщение о том, что их экранное представление должно быть обновлено. При этом автоматически для всех областей просмотра, связанных с документом (кроме области, заданной Источник), этот метод вызывает функцию

**CView::OnUpdate( )** ,

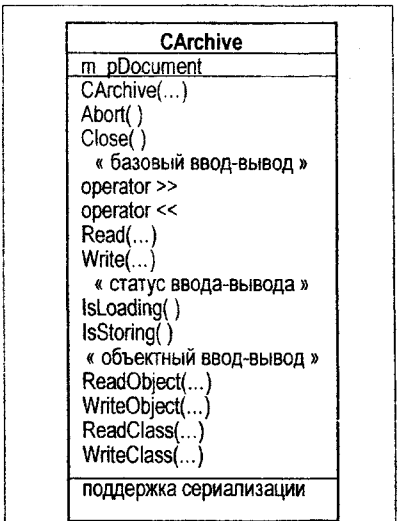



Рисунок 21 – Состав класса CArchive

которую при необходимости можно переопределить. Здесь параметр Источник является указателем на объект области просмотра, в которой было внесено изменение в документ.

Если он равен NULL, то необходимо обновить содержимое всех областей просмотра, связанных с документом. Параметры lParam и Объект нужны для оптимизации процесса обновления областей просмотра.

При каждой модификации документа необходимо уведомить систему, чтобы, в частности, при завершении сеанса работы документ был сохранен в последней версии. Соответственно пользователь будет получать предупреждение о том, что перед закрытием документа его необходимо сохранить. Для этого пользователь должен предусмотреть вызов метода

**void CDocument::SetModifiedFlag( BOOL *IsModified* = TRUE )** ,

где ненулевое значение параметра IsModified вызывает установку флага модификации документа (иначе этот флаг сбрасывается).

### 3.2. Создание пользовательского класса для работы с приложением

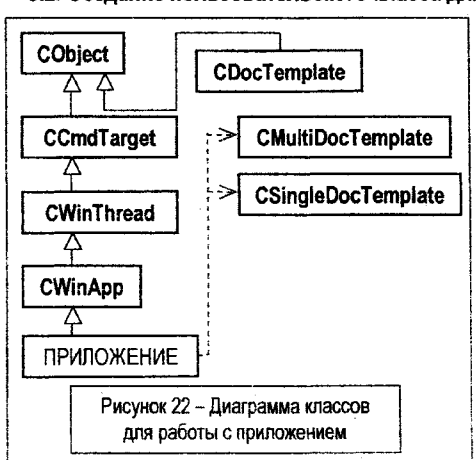



Рисунок 22 – Диаграмма классов для работы с приложением

будет включать дополнительные функции поддержки механизма идентификации (здесь не надо разрешать динамическое создание объектов), метод инициализации приложения InitInstance, необходимые обработчики, включая стандартные, приведенные в таблице 1.

Класс, предназначенный для создания приложения (рисунок 22), порождается аналогично тому, как это делалось в приложениях на базе ТКП от класса CWinApp (рисунок 23).

При этом класс приложения, порождаемый от класса CWinApp (рисунок 24),

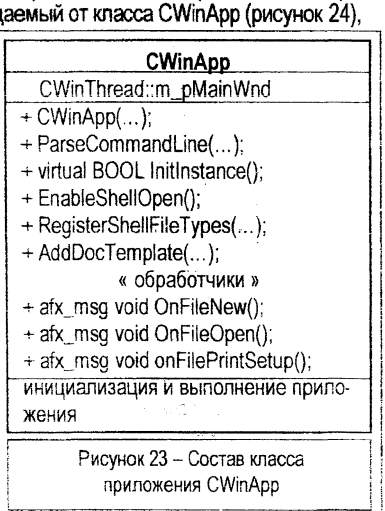



Рисунок 23 – Состав класса приложения CWinApp

### 3.3. Создание пользовательского класса для работы с видом

Класс области просмотра управляет отображением документа, предназначен для обработки действий, выполняемых пользователем в области просмотра. Область просмотра перекрывает главное окно. Класс области просмотра (рисунок 25) порождается от класса CView или от производных от него классов, например класса CScrollView, поддерживающего полосы прокрутки (рисунок 26). Класс должен быть объявлен динамическим с помощью макроса DECLARE_DYNCREATE. Класс CView содержит множество методов (рисунок 27).

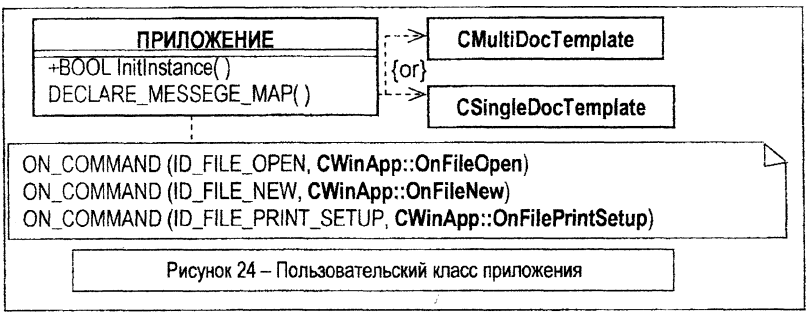



Рисунок 24 – Пользовательский класс приложения

Часть методов в пользовательском классе переописывается (рисунок 28). Так, виртуальный метод

**virtual void CView::OnDraw(CDC *DC) = 0** ,

вызываемый, когда содержимое области просмотра должно быть изменено (при перекрытии ее другим окном или изменении данных), необходимо переопределить. Метод аналогичен используемому в ТКП методу OnPaint.

Для использования метода необходимо располагать в нем:

- указателем на контекст устройства, связанным с текущей областью просмотра, для организации вывода в окно просмотра. Так как параметр DC и является указателем на контекст устройства, связанным с текущей областью просмотра, то нет необходимости получать его явно, как это делалось в методе OnPaint;
- указателем DocumentPtr на объект класса документ (DOCUMENT * DocumentPtr = (DOCUMENT *) GetDocument( ) ) для доступа как к самому документу, так и методам соответствующего класса.

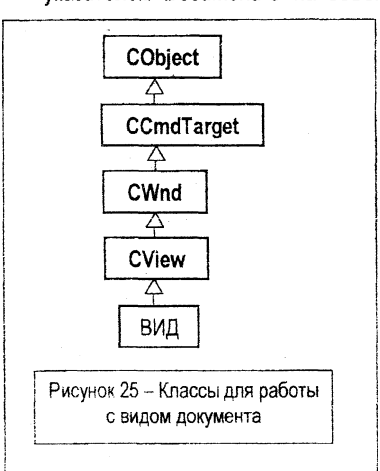



Рисунок 25 – Классы для работы с видом документа

Поскольку в ТКП ДВ данные инкапсулированы в документе, а пользовательская обработка в значительной мере в классе вид, то требуется из объекта вида получать доступ к данным документа.

Для этого можно использовать метод GetDocument

**CDocument *CView::GetDocument( ) const** ,

который возвращает указатель на объект документа класса CDocument, связанный с вызвавшей его областью просмотра, и позволяет из класса области просмотра вызывать не только данные, но и методы класса CDocument.

Примерная структура пользовательского класса вид приведена на рисунке 28.

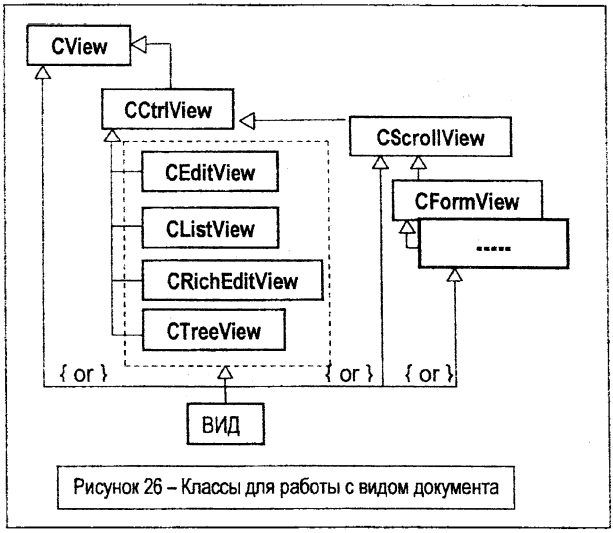



Рисунок 26 – Классы для работы с видом документа

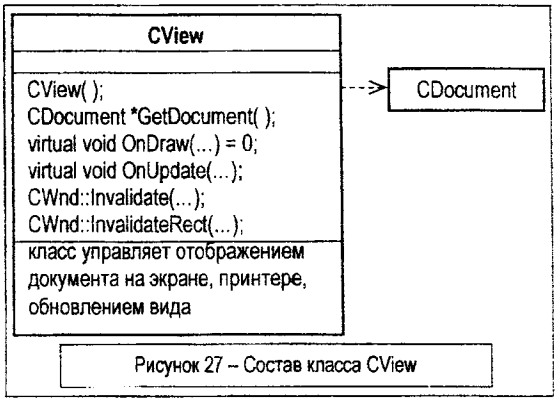



Рисунок 27 – Состав класса CView

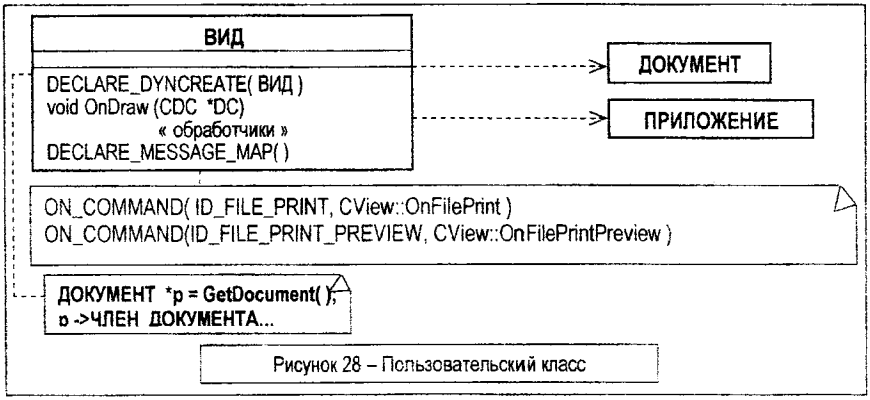



Рисунок 28 – Пользовательский класс

### 3.4. Создание пользовательского класса для работы с главным окном

Классы, предназначенные для создания главного окна, порождаются аналогично тому, как это делалось в приложениях на базе ТКП.

Класс главного окна порождается от класса CFrameWnd (рисунок 29). Для него должно быть разрешено динамическое создание объектов, т.е. включается макрос DECLARE_DYNCREATE. Класс главного окна будет включать меньше функций, переадресованных в класс вида, сохранив за собой поддержку главного меню, управление масштабированием и минимизацией. Он также инкапсулирует объекты элементов управления, связанные с окном (панель инструментов, строку состояния).

На рисунке 30 приведен базовый фрагмент класса, порождаемого от CFrameWnd и содержащего макрос DECLARE_DYNCREATE. Макрос может быть объявлен в закрытой или в защищенной и даже открытой части класса, если это необходимо.

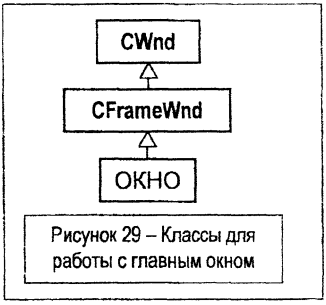



Рисунок 29 – Классы для работы с главным окном

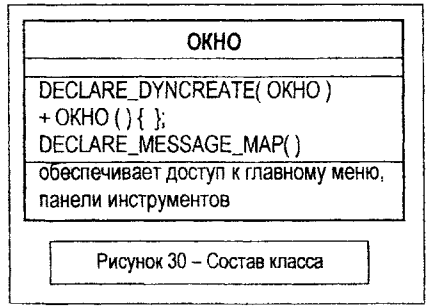



Рисунок 30 – Состав класса

### 3.5. Иерархия классов и взаимодействие объектов приложения

На рисунке 31 приведена упрощенная диаграмма классов каркаса приложения на базе архитектуры документ-вид в терминах языка UML.

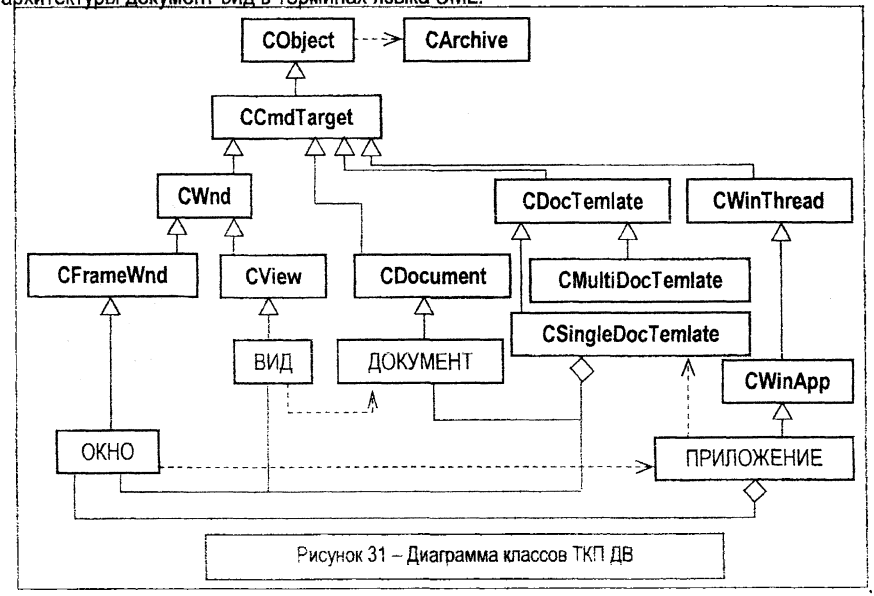



Рисунок 31 – Диаграмма классов ТКП ДВ

Пользовательские классы каркаса получают наследованием соответствующих базовых классов библиотеки MFC, включая классы CFrameWnd, CView, CDocument, CWinApp, CSingleDocTemplate, CMultiDocTemplate и др.

Кроме этого, используются классы библиотеки MFC, не входящие в иерархию наследования от класса CObject, например, класс CArchive.

Пользовательские классы соответствующего приложения объединяются в так называемый шаблон документа посредством динамического создания объектов документа, вида, окна и их агрегации в соответствующем объекте класса CSingleDocTemplate или CMultiDocTemplate.

Порядок взаимодействия объектов основных классов каркаса приложения архитектуры документ-вид в процессе создания объекта приложения и его инициализации иллюстрируется рисунком 32, где представлена диаграмма последовательностей в терминах языка UML.

Как видно из диаграммы, при создании объекта автоматически запускается метод InitInstance, который в свою очередь активизирует конструкторы пользовательских объектов, производных от классов CFrameWnd, CView, CDocument и вызывает создание нового документа (подготавливает исходный документ, инициализирует его данные), обновляет область просмотра и т.д.

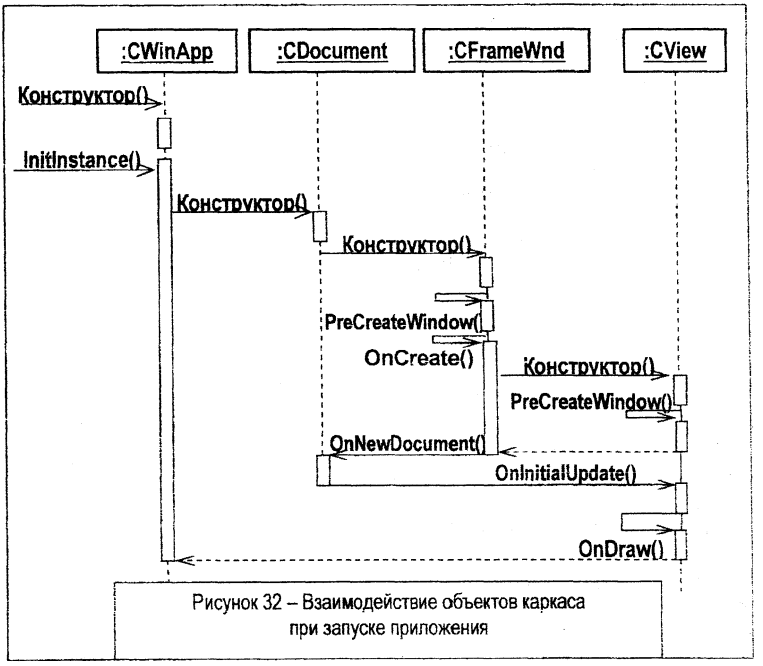



Рисунок 32 – Взаимодействие объектов каркаса при запуске приложения

# 4. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ КАРКАСА ДОКУМЕНТ-ВИД

## 4.1. Обработка строки символов

Приложение предназначено для ввода, редактирования, хранения-загрузки, вывода, визуализации строки символов, представляемой переменной CString Str. Диаграмма прецедентов приложения приведена на рисунке 33, где отображены базовые прецеденты, прецеденты расширения и включения. Состав ресурсов, вид интерфейсных форм (рисунки 34-38), описание потоков событий, основные сценарии работы приложения описаны ниже.

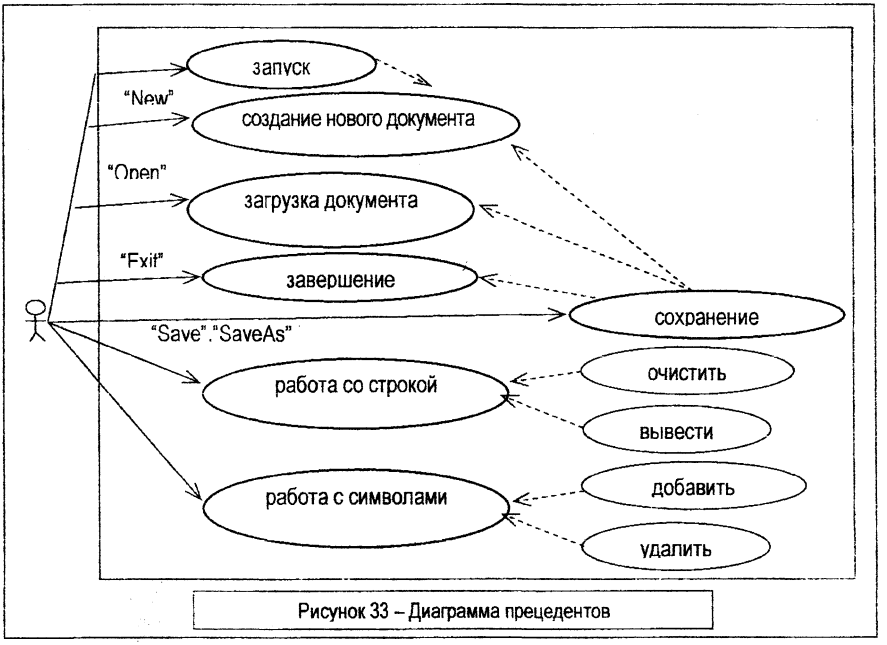



Рисунок 33 – Диаграмма прецедентов

При запуске приложения автоматически исполняется обработчик OnNewDocument( ) и создается новый документ с именем по умолчанию и расширением ".dvi", при этом имя можно изменить при сохранении документа. При выборе пункта меню New, если перед этим был загружен или создан новый документ, делается запрос на его сохранение и затем автоматически запускается обработчик OnNewDocument( ). При выборе пункта меню Open, если перед этим был загружен или создан новый документ, то делается запрос на его сохранение и только затем загружается указанный документ. При выборе пункта меню Exit, если перед этим был загружен или создан новый документ, то делается запрос на его сохранение и только затем приложение завершает работу. При выборе пункта меню Save или SaveAs выполняется сохранение текущего документа.

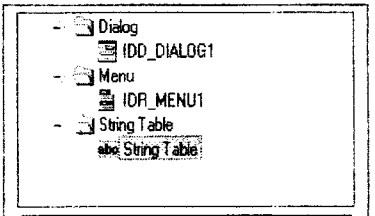



Рисунок 34 – Состав ресурсов

При выборе пункта меню Document-Clear очищается содержимое строки (обработчик DOCUMENT:: ToPrint ( ) ). При выборе пункта меню Document-Print содержимое строки выводится (обработчик VIEW:: ToPrint() ).

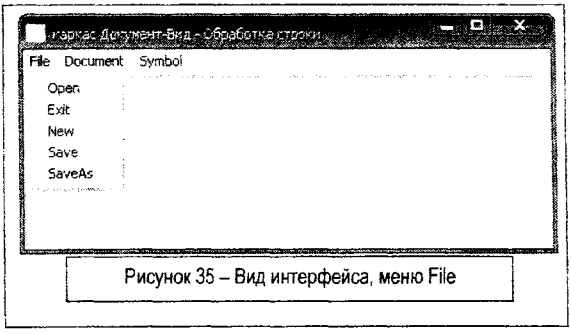



Рисунок 35 – Вид интерфейса, меню File

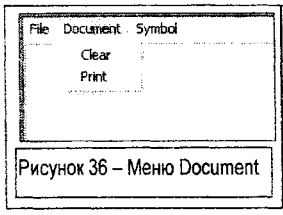



Рисунок 36 – Меню Document

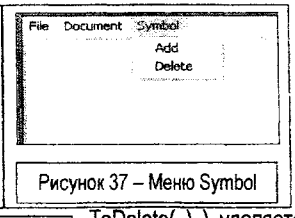



Рисунок 37 – Меню Symbol

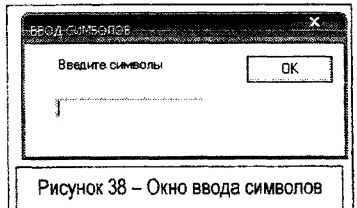



Рисунок 38 – Окно ввода символов

При выборе пункта меню Symbol-Add (обработчик VIEW::ToAdd()) пользователю предоставляется возможность ввести новые символы посредством диалогового окна. Корректировка строки реализуется обработчиком события - нажатие кнопки Enter диалогового окна (myDIALOG::OnEnter() ) путем добавления в ее конец введенных символов. При этом контролируется предельная длина строки и при необходимости выводится предупреждающее сообщение. Ввод символов можно производить и непосредственно с клавиатуры (обработчик VIEW::OnChar(...)). При выборе пункта меню Symbol-Delete (обработчик VIEW::ToDelete( ) ) удаляется символ с конца строки. При этом контролируется размер строки и при необходимости выводится предупреждающее сообщение. Удаление символов можно производить и непосредственно мышью - нажатием-щелчком правой клавиши (обработчик VIEW::OnRButtonDown(...) ).

При создании меню (идентификатор IDR_MENU1) были использованы следующие идентификаторы исполнимых пунктов. Для команд пункта File: ID_FILE_OPEN; ID_APP_EXIT; ID_FILE_NEW; ID_FILE_SAVE; ID_FILE_SAVE_AS. Для команд пункта Document: IDM_CLEAR; IDM_PRINT. Для команд пункта Symbol: IDM_ADD; IDM_DELETE.

При создании окна ввода символов (идентификатор IDD_DIALOG1) были использованы следующие идентификаторы ЭУ. Для окна редактирования (тип CEdit) - IDC_EDIT1, для кнопки (типа CButton) - ID_ENTER.

Состав классов, диаграммы классов приложения приведены на рисунках 39-41. Строка настройки и текст приложения приведены ниже.

Строка IDR_MENU (101): "Обработка строки\nкаркас Документ-Вид\nField_3\nFild_4 (*.dvi)\n.dvi\nDVTest\nDV Test".

```
#include      <afxwin.h>
#include      <afxres.h>
#include      "resource.h"
#include      <iostream>
#include      <string>
using namespace std;
const MAX_LENGTH = 15;
CString InStr;
```

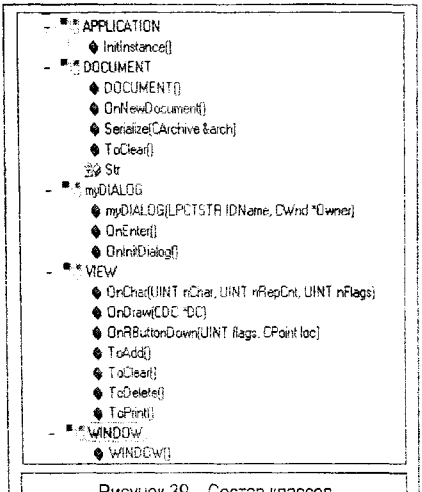



Рисунок 39 – Состав классов

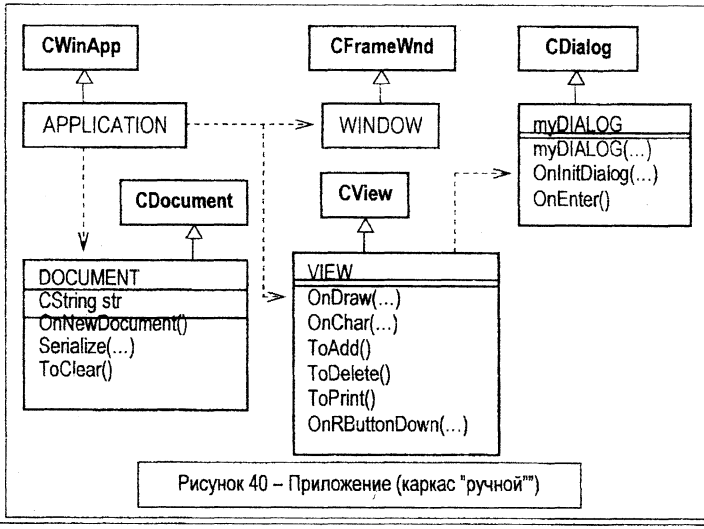



Рисунок 40 – Приложение (каркас "ручной"")

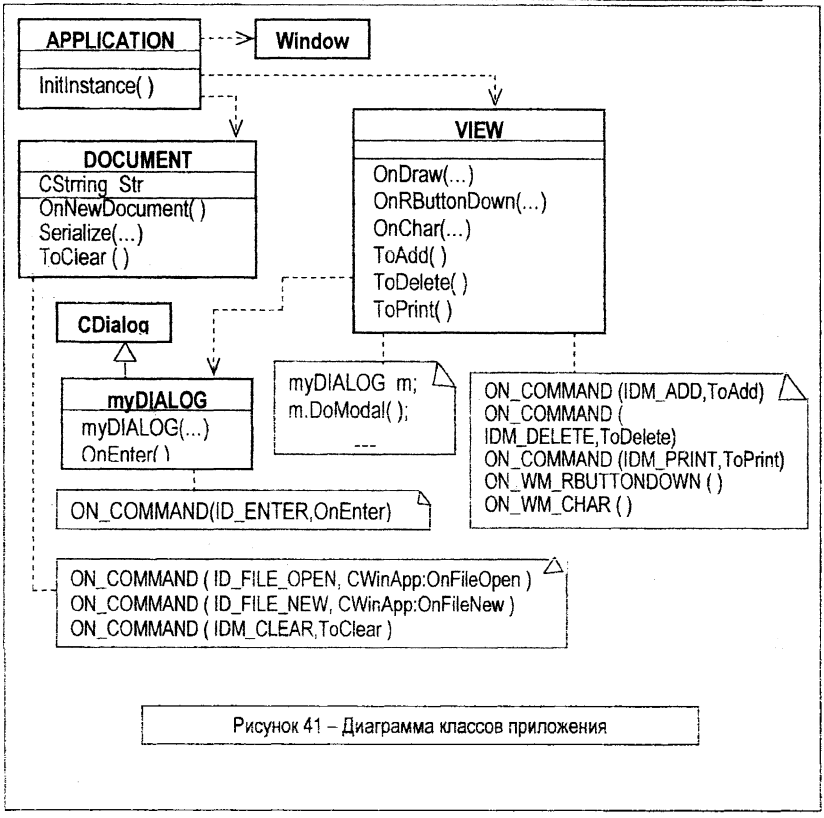



Рисунок 41 – Диаграмма классов приложения

```
// ========== ОКНО =============
class WINDOW : public CFrameWnd
{
    DECLARE_DYNCREATE(WINDOW)
public:
      WINDOW ();
      DECLARE_MESSAGE_MAP()
};
// ========= ПРИЛОЖЕНИЕ ==========
class APPLICATION : public CWinApp
{
public:
    BOOL InitInstance();
    DECLARE_MESSAGE_MAP()
};
// ========== ДОКУМЕНТ ==========
class DOCUMENT : public CDocument
{
    DECLARE_DYNCREATE(DOCUMENT)
    CString Str;
public:
    DOCUMENT ();
    BOOL OnNewDocument();
    void Serialize(CArchive &arch);
    afx_msg void  ToClear();
    DECLARE_MESSAGE_MAP()
};
// ============ ВИД ==================
class VIEW : public CView
{
    DECLARE_DYNCREATE(VIEW)
public:
    void OnDraw(CDC *DC);
    afx_msg void  OnRButtonDown (UINT flags, CPoint loc);
    afx_msg void  OnChar(UINT nChar, UINT nRepCnt, UINT nFlags );
    afx_msg void  ToAdd();
    afx_msg void  ToDelete();
    afx_msg void  ToPrint();
    DECLARE_MESSAGE_MAP()
};
// ======= ДИАЛОГОВОЕ ОКНО =========
class myDIALOG : public CDialog
{
public:
    myDIALOG(LPCTSTR IDName, CWnd *Owner): CDialog (IDName, Owner ){ };
    BOOL OnInitDialog()
    {
        CDialog::OnInitDialog();
        CEdit *pEditBox1 = (CEdit *) CDialog::GetDlgItem(IDC_EDIT1);
        InStr = "";
        pEditBox1->SetWindowText(InStr);
        pEditBox1->SetFocus();
```

```
        return 0;
    }
    afx_msg void OnEnter();
    DECLARE_MESSAGE_MAP()
};

// ===== создание экземпляра приложения ===
APPLICATION   TheApplication;
IMPLEMENT_DYNCREATE(WINDOW,CFrameWnd)
IMPLEMENT_DYNCREATE(VIEW,CView)
IMPLEMENT_DYNCREATE(DOCUMENT,CDocument)

//-----------------------------
WINDOW::WINDOW () { }

BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
END_MESSAGE_MAP()

//-----------------------------------------------
BOOL APPLICATION::InitInstance ()
{
    CSingleDocTemplate *DocPtr = new CSingleDocTemplate(IDR_MENU1,
        RUNTIME_CLASS(DOCUMENT), RUNTIME_CLASS(WINDOW),
                                        RUNTIME_CLASS(VIEW));
    AddDocTemplate(DocPtr);
    EnableShellOpen();
    RegisterShellFileTypes();
    CCommandLineInfo CLInfo;
    ParseCommandLine(CLInfo);
    if (!ProcessShellCommand(CLInfo)) return FALSE;
    return TRUE;
}
BEGIN_MESSAGE_MAP(APPLICATION, CWinApp)
    ON_COMMAND(ID_FILE_OPEN,CWinApp::OnFileOpen)
    ON_COMMAND(ID_FILE_NEW,CWinApp::OnFileNew)
END_MESSAGE_MAP()

//-------------------------------------
DOCUMENT::DOCUMENT () { }
//-----------------------------------------------
BOOL DOCUMENT::OnNewDocument ()
{
  if (!CDocument::OnNewDocument()) return FALSE;
    Str = "";
    return TRUE;
}
//-----------------------------------------------
void DOCUMENT::Serialize(CArchive &arch)
{
    if (arch.IsLoading())   arch >> Str;
    else    arch << Str;
}
```

```
//---------------------------------------------
afx_msg void  DOCUMENT::ToClear()
{    Str = "";  }

BEGIN_MESSAGE_MAP(DOCUMENT, CDocument)
   ON_COMMAND(IDM_CLEAR,ToClear)
END_MESSAGE_MAP()

//---------------------------------------------
void VIEW::OnDraw(CDC *DC)
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    DC->TextOut(1,10,DocPtr->Str, DocPtr->Str.GetLength() );
}
//-------------------------------------------------------------------------
afx_msg void VIEW::OnRButtonDown(UINT flags, CPoint loc)
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    DocPtr->SetModifiedFlag();
    DocPtr->UpdateAllViews(NULL);
}
//-------------------------------------------------------------------------
afx_msg void  VIEW::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    if ( (DocPtr->Str.GetLength()) < MAX_LENGTH )
    {
        DocPtr->Str += nChar;
        DocPtr->SetModifiedFlag();
        DocPtr->UpdateAllViews(NULL);
    }
    else
        MessageBox("строка исчерпана","Внимание",MB_OK |
                MB_ICONEXCLAMATION );
}
//---------------------------------------------
afx_msg void  VIEW::ToAdd()
{
    int DeltaLength;
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    myDIALOG MyDialog ((LPCTSTR) IDD_DIALOG1,this);
    MyDialog.DoModal();
    if ( DocPtr->Str.GetLength() < MAX_LENGTH )
    {
        if ( InStr.GetLength() != 0 )
        {
            DeltaLength = DocPtr->Str.GetLength() + InStr.GetLength() -
                    MAX_LENGTH;
            if (DeltaLength) // ---ЕСТЬ лишние символы
                InStr.Delete(InStr.GetLength()-DeltaLength, DeltaLength);
            DocPtr->Str += InStr;
        }
```

```
            DocPtr->SetModifiedFlag();
            DocPtr->UpdateAllViews(NULL);
    }
    else
            MessageBox("строка исчерпана","Внимание",MB_OK |
                        MB_ICONEXCLAMATION );
}
//-----------------------------------------
afx_msg void  VIEW::ToDelete()
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    if (DocPtr->Str.GetLength() > 0)
    {
      DocPtr->Str.Delete(DocPtr->Str.GetLength()-1,1);
            DocPtr->SetModifiedFlag();
            DocPtr->UpdateAllViews(NULL);
    }
    else
            MessageBox("строка удалена","Внимание",MB_OK |
                        MB_ICONEXCLAMATION );
}
//-----------------------------------------
afx_msg void VIEW::ToPrint()
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    DocPtr->SetModifiedFlag();
    DocPtr->UpdateAllViews(NULL);
}

BEGIN_MESSAGE_MAP(VIEW, CView)
    ON_COMMAND(IDM_ADD,ToAdd)
    ON_COMMAND(IDM_DELETE,ToDelete)
    ON_COMMAND(IDM_PRINT,ToPrint)
    ON_WM_LBUTTONDOWN()
    ON_WM_RBUTTONDOWN()
    ON_WM_CHAR()
END_MESSAGE_MAP()

//-----------------------------------------------------
afx_msg void myDIALOG::OnEnter()
{
    CEdit *pEditBox1 = (CEdit *) CDialog::GetDlgItem(IDC_EDIT1);
    InStr = "";
    pEditBox1->GetWindowText(InStr);
    EndDialog(0);
};

BEGIN_MESSAGE_MAP(myDIALOG, CDialog)
    ON_COMMAND(ID_ENTER,OnEnter)
END_MESSAGE_MAP()
```

## 4.2. Обработка строки символов с использованием пользовательского класса

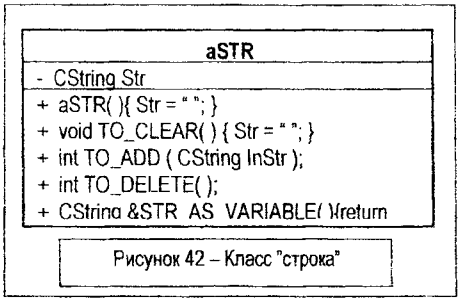



Рисунок 42 – Класс "строка"

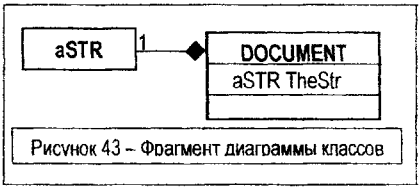



Рисунок 43 – Фрагмент диаграммы классов

Приложение предназначено для ввода, редактирования, хранения-загрузки, вывода, визуализации строки символов, представляемой переменной CString Str. Но в отличие от предыдущего варианта все операции со строками инкапсулированы в пользовательском классе aSTR (рисунок 42). Соответственно диаграмма прецедентов приложения (рисунке 33), состав ресурсов, вид интерфейсных форм (рисунки 34-38), описание потоков событий, основные сценарии работы приложения совпадают с приведенными выше для предыдущего приложения. Состав классов, диаграммы классов приложения приведены на рисунках 43, 44. Строка настройки и текст приложения приведены в ПРИЛОЖЕНИИ 2. Описание класса aSTR представлено ниже.

```
class aSTR
{
  CString Str;
public:
    aSTR ( ) { Str = ""; };
    void TO_CLEAR ( ) { Str = ""; };
    int  TO_ADD( CString InStr )
    {
        int DeltaLength;
        if ( Str.GetLength( ) < MAX_LENGTH )
        {
            if ( InStr.GetLength( ) != 0 )
            {
                DeltaLength = Str.GetLength( ) + InStr.GetLength( ) -
                              MAX_LENGTH;
                if ( DeltaLength )    // ---ЕСТЬ лишние символы
                   InStr.Delete( InStr.GetLength( ) - DeltaLength, DeltaLength );
                Str += InStr;
                return 1;
            }
        }
        else return 0;
    };
    int  TO_DELETE( )
    {
        If ( Str.GetLength( ) > 0 )
        {
            Str.Delete( Str.GetLength( ) - 1,1 );
            return 1;
        }
        else    return 0;
    };
    CString  &STR_AS_VARIABLE( )
    {  return Str; };
};
```

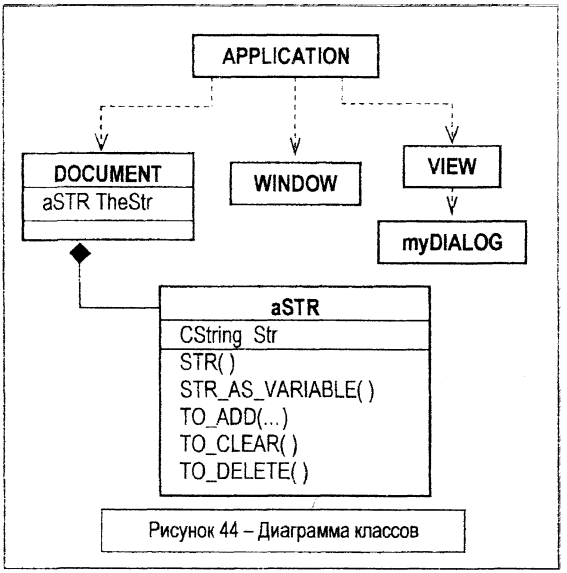



Рисунок 44 – Диаграмма классов

### 4.3. Модификация компоновки приложения обработки строки символов

В отличие от предыдущего варианта все операции со строками, инкапсулированные в пользовательском классе aSTR, скомпонованы в виде двух файлов aSTR.cpp, aSTR.h.


Рисунок 45 – Состав классов

Для этого можно использовать пункт меню Project-AddToProject-New, создав соответствующие пустые файлы и внеся в них описания интерфейса и реализации класса aSTR.

Состав классов представлен на рисунке 45, а диаграмма компонентов приложения представлена на рисунке 46.

Соответственно файл aSTR.h будет включать:

```
#include      <afxwin.h>
#include      <afxres.h>
const MAX_LENGTH = 15;
class aSTR
{
   CString Str;
public:
     aSTR ( ) ;
     void TO_CLEAR ( ) ;
     int  TO_ADD( CString InStr );
     int  TO_DELETE( );
     CString  &STR_AS_VARIABLE( );
};
```

Соответственно файл aSTR.cpp будет включать:

```
#include      <afxwin.h>
#include      <afxres.h>
#include      "aSTR.h"
```

```
aSTR::aSTR ( ) { Str = ""; };
void aSTR::TO_CLEAR ( ) {     Str = ""; };
int  aSTR::TO_ADD( CString  InStr )
{
    int DeltaLength;
    if ( Str.GetLength( ) < MAX_LENGTH )
    {
        if ( InStr.GetLength( ) != 0 )
        {
            DeltaLength = Str.GetLength( ) + InStr.GetLength( ) - MAX_LENGTH;
            if ( DeltaLength )          // ---ЕСТЬ лишние символы
                InStr.Delete( InStr.GetLength( ) - DeltaLength, DeltaLength );
            Str += InStr;
            return 1;
        }
    }
    else   return 0;
    return 0;
};
int  aSTR::TO_DELETE( )
{
    if ( Str.GetLength( ) > 0 )
    {
        Str.Delete( Str.GetLength( )-1,1 );
        return 1;
    }
    else   return 0;
};
CString  &aSTR::STR_AS_VARIABLE( )
{   return Str;    };
```

Соответственно текст модуля main теперь не содержит описания класса aSTR, а взамен включает директиву #include "aSTR.h":

```
#include      <afxwin.h>
#include      <afxres.h>
#include      "resource.h"
#include      <iostream>
#include      "aSTR.h"
```

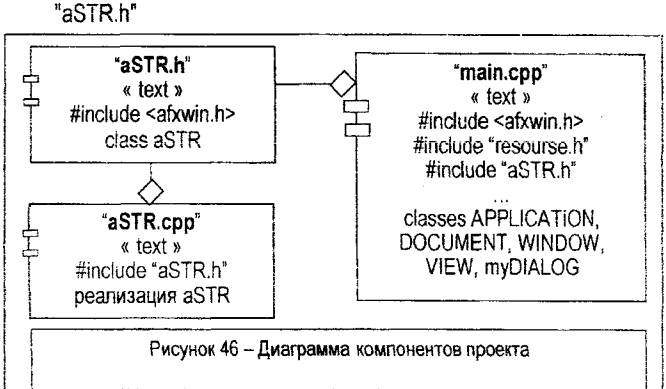



Рисунок 46 – Диаграмма компонентов проекта

## 5. ПОРЯДОК ВЫПОЛНЕНИЯ

ТЕМА РАБОТЫ: "Изучение типового каркаса MFC-приложения архитектуры документ-вид (ТКП ДВ). Базовые классы. Структура каркаса. Организация взаимодействия объектов".

ЦЕЛЬ РАБОТЫ:

- изучение ТКП ДВ, разработка и реализация MFC-приложений с использованием ТКП;
- организация обработки сообщений, текстового и графического вывода, изучение проблем перерисовки.

СОДЕРЖАНИЕ ОТЧЕТА:

- описание ТКП ДВ (интерфейса, диаграмм классов, диаграмм взаимодействия объектов);
- описание разработанных приложений.

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.

ПОРЯДОК ВЫПОЛНЕНИЯ.

1. **Изучить теоретический материал**
   - ознакомиться с общими сведениями о каркасном программировании (см. § 1);
   - изучить типовой каркас документ-вид, ознакомиться с архитектурой типового каркаса MFC-приложения, используемых классах и методах (см. § 2);
   - ознакомиться со средой разработки, с основами создания приложений на базе ТКП ДВ (см. § 3).
2. **Создать** "пустое" приложение (типовой каркас приложения) (с именем EX1) в соответствии с § 2.5. Это каркас MFC-приложения с главным окном, но без обработки сообщений. Изучить свойства приложения, разработать диаграммы UML.
3. **Воспроизвести** приложение (с именем EX2) для обработки строки символов (см. § 4.1).
4. **Воспроизвести** приложение (с именем EX3) для обработки строки символов с использованием пользовательского класса (см. § 4.2).
5. **Модифицировать** предыдущее приложение (с именем EX4), изменив его компоновку (см. § 4.3).
6. **Разработать** приложение (с именем EX5) для хранения, редактирования и отображения плоских фигур, соединенных прямыми линиями. Ввод координат линий производить в поле окна просмотра, используя клавиши "мыши".

## ЛИТЕРАТУРА


1. Шилдт, Г. Самоучитель C++, 3-е изд. – СПб.: БХВ-Петербург, 2003. – 688 с.
2. Паппас, К. Visual C++. Руководство для профессионалов: пер. с англ. / К. Паппас, У. Мюррей. - СПб: BHV-Санкт-Петербург, 1996.
3. Паппас, К. Эффективная работа: Visual C++.NET / К. Паппас, У. Мюррей. – СПб.: Питер, 2002. – 816 с.

*Дополнительная литература*

4. Поляков, А.Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А.Ю. Поляков, В.А. Брусенцев. – СПб.: БХВ-Петербург, 2003. – 560 с.
5. Орлов, С.А. Технологии разработки программного обеспечения: учебник для вузов. – СПб.: Питер, 2004. – 527 с.
6. Шилдт, Г. Справочник программиста по C/C++, 3-е изд. – М.: Изд. Дом Вильямс, 2003. – 432 с.
7. Шмуллер, Дж. Освой самостоятельно UML за 24 часа. – М.: Изд. Дом Вильямс, 2002. – 352 с.

## ПРИЛОЖЕНИЕ 1. Листинг каркаса ("ручной")

В тексте жирным шрифтом выделены идентификаторы, которые может менять пользователь по своему усмотрению. Закомментированные обработчики также добавляются и доопределяются при необходимости пользователем.

Строка настройки **IDR_MENU**:

**Document-View\nDocName\nDocType\nDoc Type (*.dvi)\n.dvi\nDVTest\nDV Test**

```
//=======================================================
//              Создание  каркаса документ-вид
// 1. Создать мастером Win32 Application каркас (тип empty).
// 2. Создать в каркасе пустой файл  *.cpp.
// 3. Вставить листинг ТКП ДВ в файл  *.cpp.
// 4. Включить в ГМ-Project-Settings-General режим "use MFC in ...".
// 5. Создать ресурсный файл *.rc.
// 6. Создать ресурс-меню IDR_MENU хотя бы с одним фиктивным пунктом.
// 7. Создать ресурс-строку IDR_MENU с начальным заполнением
//     Document-View\nDocName\nDocType\nDoc Type (*.dvi)\n.dvi\nDVTest\nDV Test
// Далее при необходимости добавить графические ресурсы, доработать меню,
//     допрограммировать классы.
//=======================================================

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

// ===== класс главного ОКНА приложения ===
class WINDOW : public CFrameWnd
{
      DECLARE_DYNCREATE(WINDOW)
public:
      WINDOW ( );
      DECLARE_MESSAGE_MAP()
};

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

// ===== класс ДОКУМЕНТ для хранения, загрузки-выгрузки данных ========
class DOCUMENT : public CDocument
{
    DECLARE_DYNCREATE(DOCUMENT)
```

```
    // описание данных документа (члены-данные класса)
public:
    DOCUMENT ( );
    BOOL OnNewDocument ( );
    void Serialize( CArchive &Arch );
    // прототипы обработчиков документа
    // afx_msg void ОбработчикДокумента ( );
    DECLARE_MESSAGE_MAP()
};

// ===== класс ВИД для обработки-редактирования, визуализации данных =====
class VIEW : public CView
{
    DECLARE_DYNCREATE(VIEW)
public:
    void OnDraw(CDC *DC);
    // прототипы обработчиков данных документа
    // afx_msg void ОбработчикВида ( );
    DECLARE_MESSAGE_MAP()
};

// ===== создание экземпляра приложения =====
APPLICATION  TheApplication;
IMPLEMENT_DYNCREATE(WINDOW,CFrameWnd)
IMPLEMENT_DYNCREATE(VIEW,CView)
IMPLEMENT_DYNCREATE(DOCUMENT,CDocument)

//-----------------------------
WINDOW::WINDOW ( ) { }

BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
    // макрокоманды включения обработчиков
END_MESSAGE_MAP()

//-----------------------------------------
BOOL APPLICATION::InitInstance ( )
{
    CSingleDocTemplate *TemplatePtr = new CSingleDocTemplate( IDR_MENU,
    RUNTIME_CLASS(DOCUMENT), RUNTIME_CLASS(WINDOW), RUNTIME_CLASS(VIEW));
    AddDocTemplate(TemplatePtr);
    EnableShellOpen();
    RegisterShellFileTypes();
    CCommandLineInfo CLInfo;
    ParseCommandLine(CLInfo);
    if (!ProcessShellCommand(CLInfo)) return FALSE;
    return TRUE;
}

BEGIN_MESSAGE_MAP(APPLICATION, CWinApp)
    // макрокоманды включения обработчиков
    //ON_COMMAND(ID_FILE_OPEN,CWinApp::OnFileOpen)
    //ON_COMMAND(ID_FILE_NEW,CWinApp::OnFileNew)
END_MESSAGE_MAP()
```

```
//-----------------------------------------
DOCUMENT::DOCUMENT ( ) { }

//-----------------------------------------------------
BOOL DOCUMENT::OnNewDocument ( )
{
    if ( ! CDocument::OnNewDocument( ) ) return FALSE;
    // инициализация данных документа
    return TRUE;
}

//-----------------------------------------------------
void DOCUMENT::Serialize(CArchive &Arch)
{
    If  ( Arch.IsLoading( ) )
    {
        // ввод данных документа из Arch >> ...
    }
    else
    {
        // вывод данных документа в Arch << ...
    }
}

BEGIN_MESSAGE_MAP(DOCUMENT, CDocument)
    // макрокоманды включения обработчиков
    //ON_COMMAND(IDM_ПунктаМеню, ОбработчикДокумента)
    // ...
END_MESSAGE_MAP()

//-----------------------------------
void VIEW::OnDraw(CDC *DC)
{
    DOCUMENT  *DocumentPtr = (DOCUMENT *)  GetDocument( );
    // перерисовка с использованием КУ – DC и указателя Документа DocumentPtr
}

//-----------------------------------------------------------
// afx_msg void VIEW::ОбработчикВида ( параметры )
// {
//    DOCUMENT * DocumentPtr = (DOCUMENT *) GetDocument( );
//    алгоритм обработчика
//    DocumentPtr -> SetModifiedFlag( );
//    DocumentPtr -> UpdateAllViews( NULL );
// }

BEGIN_MESSAGE_MAP(VIEW, CView)
    // макрокоманды включения обработчиков
    //ON_COMMAND(IDM_ПунктаМеню, ОбработчикВида)
    // ...
END_MESSAGE_MAP()
```

## ПРИЛОЖЕНИЕ 2. Листинг приложения для обработки С-строки на базе пользовательского класса

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

const MAX_LENGTH = 15;
CString InStr;

// ===== пользовательский класс СТРОКА символов ========
class aSTR
{
  CString Str;
public:
    aSTR ( ) { Str = ""; };
    void TO_CLEAR ( ) { Str = ""; };
    int  TO_ADD(CString InStr)
    {
          int DeltaLength;
          if ( Str.GetLength() < MAX_LENGTH )
          {
                if ( InStr.GetLength() != 0 )
                {
                    DeltaLength = Str.GetLength() + InStr.GetLength() - MAX_LENGTH;
                    if (DeltaLength)   // ---ЕСТЬ лишние символы
                          InStr.Delete(InStr.GetLength()-DeltaLength, DeltaLength);
                     Str += InStr;
                     return 1;
                }
          }
          else    return 0;
    };
    int  TO_DELETE( )
    {
          if (Str.GetLength() > 0)
          {
                Str.Delete(Str.GetLength()-1,1);
                return 1;
          }
          else    return 0;
    };
    CString  &STR_AS_VARIABLE( )
    {  return Str;  };
};

    class WINDOW : public CFrameWnd
```

```
{
    DECLARE_DYNCREATE(WINDOW)
public:
      WINDOW ();
      DECLARE_MESSAGE_MAP()
};

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

class DOCUMENT : public CDocument
{
    DECLARE_DYNCREATE(DOCUMENT)
    aSTR   TheStr;

public:
    DOCUMENT ();
    BOOL OnNewDocument();
    void Serialize(CArchive &arch);
    afx_msg void  ToClear();
    DECLARE_MESSAGE_MAP()
};

class VIEW : public CView
{
    DECLARE_DYNCREATE(VIEW)
public:
    void OnDraw(CDC *DC);
    afx_msg void  OnRButtonDown (UINT flags, CPoint loc);
    afx_msg void  OnChar(UINT nChar, UINT nRepCnt, UINT nFlags );
    afx_msg void  ToAdd();
    afx_msg void  ToDelete();
    afx_msg void  ToPrint();
    DECLARE_MESSAGE_MAP()
};

// ===== класс диалогового ОКНА ========
class myDIALOG : public CDialog
{
public:
    myDIALOG(LPCTSTR IDName, CWnd *Owner): CDialog (IDName, Owner ){ };
    BOOL OnInitDialog()
    {
            CDialog::OnInitDialog();
            CEdit *pEditBox1 = (CEdit *) CDialog::GetDlgItem(IDC_EDIT1);
            InStr = "";
            pEditBox1->SetWindowText(InStr);
```

```
            pEditBox1->SetFocus();
            return 0;
    }
    afx_msg void OnEnter();
    DECLARE_MESSAGE_MAP()
};


// ===== создание экземпляра приложения ===
APPLICATION   TheApplication;
IMPLEMENT_DYNCREATE(WINDOW,CFrameWnd)
IMPLEMENT_DYNCREATE(VIEW,CView)
IMPLEMENT_DYNCREATE(DOCUMENT,CDocument)

WINDOW::WINDOW ( ) { }

BEGIN_MESSAGE_MAP(WINDOW, CFrameWnd)
END_MESSAGE_MAP()

BOOL APPLICATION::InitInstance ( )
{
    CSingleDocTemplate *DocPtr = new CSingleDocTemplate(IDR_MENU6,
    RUNTIME_CLASS(DOCUMENT), RUNTIME_CLASS(WINDOW),
                                RUNTIME_CLASS(VIEW));
    AddDocTemplate(DocPtr);
    EnableShellOpen();
    RegisterShellFileTypes();
    CCommandLineInfo CLInfo;
    ParseCommandLine(CLInfo);
    if (!ProcessShellCommand(CLInfo)) return FALSE;
    return TRUE;
}

BEGIN_MESSAGE_MAP(APPLICATION, CWinApp)
    ON_COMMAND(ID_FILE_OPEN,CWinApp::OnFileOpen)
    ON_COMMAND(ID_FILE_NEW,CWinApp::OnFileNew)
END_MESSAGE_MAP()

DOCUMENT::DOCUMENT ( ) { }
BOOL DOCUMENT::OnNewDocument ( )
{
  if (!CDocument::OnNewDocument()) return FALSE;
      TheStr.TO_CLEAR();
  return TRUE;
}
void DOCUMENT::Serialize(CArchive &arch)
{
    if (arch.IsLoading())
          arch >> TheStr.STR_AS_VARIABLE();
    else
```

```
        arch << TheStr.STR_AS_VARIABLE();
}
afx_msg void  DOCUMENT::ToClear()
{   TheStr.TO_CLEAR();   }

BEGIN_MESSAGE_MAP(DOCUMENT, CDocument)
   ON_COMMAND(IDM_CLEAR,ToClear)
END_MESSAGE_MAP()

void VIEW::OnDraw(CDC *DC)
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    DC->TextOut(1,10,DocPtr->TheStr.STR_AS_VARIABLE() );
}
afx_msg void VIEW::OnRButtonDown(UINT flags, CPoint loc)
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    DocPtr->SetModifiedFlag();
    DocPtr->UpdateAllViews(NULL);
}
afx_msg void  VIEW::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    if ( DocPtr->TheStr.TO_ADD(nChar) )
    {
        DocPtr->SetModifiedFlag();
        DocPtr->UpdateAllViews(NULL);
    }
    else   MessageBox("строка исчерпана","Внимание",MB_OK |
                MB_ICONEXCLAMATION );
}
afx_msg void  VIEW::ToAdd()
{

    int DeltaLength;
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    myDIALOG MyDialog ((LPCTSTR) IDD_DIALOG1,this);
    MyDialog.DoModal();
    if ( DocPtr->TheStr.TO_ADD(InStr) )
    {
        DocPtr->SetModifiedFlag();
        DocPtr->UpdateAllViews(NULL);
    }
    else   MessageBox("строка исчерпана","Внимание",MB_OK |
                MB_ICONEXCLAMATION );
}
afx_msg void  VIEW::ToDelete()
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    if (DocPtr->TheStr.TO_DELETE())
    {
```

```
        DocPtr->SetModifiedFlag();
            DocPtr->UpdateAllViews(NULL);
    }
    else  MessageBox("строка удалена","Внимание",MB_OK | MB_ICONEXCLAMATION );
}
afx_msg void VIEW::ToPrint()
{
    DOCUMENT *DocPtr = (DOCUMENT *) GetDocument();
    DocPtr->SetModifiedFlag();
    DocPtr->UpdateAllViews(NULL);
}

BEGIN_MESSAGE_MAP(VIEW, CView)
    ON_COMMAND(IDM_ADD,ToAdd)
     ON_COMMAND(IDM_DELETE,ToDelete)
     ON_COMMAND(IDM_PRINT,ToPrint)
     ON_WM_LBUTTONDOWN()
     ON_WM_RBUTTONDOWN()
    ON_WM_CHAR()
END_MESSAGE_MAP()

// ===== класс диалогового ОКНА ========
afx_msg void myDIALOG::OnEnter()
{
    CEdit *pEditBox1 = (CEdit *) CDialog::GetDlgItem(IDC_EDIT1);
    InStr = "";
    pEditBox1->GetWindowText(InStr);
    EndDialog(0);
};

BEGIN_MESSAGE_MAP(myDIALOG, CDialog)
    ON_COMMAND(ID_ENTER,OnEnter)
END_MESSAGE_MAP()  .
```

# ОГЛАВЛЕНИЕ

Учебное издание

*Авторы-составители:*
**Муравьев Геннадий Леонидович,**
**Мухов Сергей Владимирович,**
**Хвещук Владимир Иванович**

# Методические указания

## "РАЗРАБОТКА ПРИЛОЖЕНИЙ НА БАЗЕ КАРКАСА ДОКУМЕНТ-ВИД"

Ответственный за выпуск: Муравьев Г.Л.
Редактор: Строкач Т.В.
Компьютерный набор: Муравьев Г.Л.
Компьютерная верстка: Боровикова Е.А.
Корректор: Никитчик Е.В.

---

Подписано в печать 17.11.2011 г. Формат 60x84 1/16. Бумага «Снегурочка».
Усл. п. л. 2,55. Усл. изд. л. 2,75. Тираж 50 экз. Заказ № 1106.
Отпечатано на ризографе учреждения образования
"Брестский государственный технический университет".
224017, г. Брест, ул. Московская, 267.