Самоучитель

Татьяна Сидорина

Microsoft® Visual Studio C++ и MFC

- Создание различных типов приложений
- Разработка интерфейса: окна диалога, меню, панель управления, строка состояния
- Работа с графикой и использование Unicode
- Обработка графических и текстовых файлов
- Работа с мастером построения приложения MFC AppWizard и редактором ресурсов
- Создание справочной системы приложения
- Справочник по классам и функциям библиотеки MFC

bhv + CD

Татьяна Сидорина

Самоучитель Microsoft® Visual Studio C++ и MFC

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.06
ББК 32.973.26-018.2
С34

Сидорина Т. Л.

С34 Самоучитель Microsoft Visual Studio C++ и MFC. — СПб.: БХВ-Петербург, 2009. — 848 с.: ил. + CD-ROM

ISBN 978-5-9775-0284-9

Книга предназначена для обучения разработке различных типов Windows-приложений с использованием библиотеки MFC в среде Microsoft Visual Studio C++. Описано создание и работа с элементами интерфеса приложения: окна диалога, меню, панель управления, строка состояния и др. Показана работа с графическими и текстовыми файлами: отображение графической и текстовой информации, масштабирование изображения, работа с метафайлами, просмотр видеороликов. Уделено внимание редактору ресурсов, созданию справочной системы с помощью HTML Help WorkShop. Приведена справочная информация по классам и функциям библиотеки MFC. Компакт-диск содержит демонстрационные примеры, рассмотренные в книге.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.09.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 68,37.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0284-9

© Сидорина Т. Л., 2008
© Оформление, издательство "БХВ-Петербург", 2008

Оглавление

Введение

Книга предназначена для обучения создания Windows-приложений с использованием MFC (Microsoft Foundation Classes — фундаментальная библиотека классов Microsoft) в среде программирования Microsoft Visual Studio 2005 или Microsoft Visual Studio 2008.

Книга ориентирована на программиста, имеющего базовые навыки программирования на языках C/C++ в среде Microsoft Visual Studio (2002, 2003, 2005) или Visual C++ 6 и знакомого с объектно-ориентированным программированием (ООП).

Для иллюстрации материала и практического закрепления навыков используются примеры, посвященные различным аспектам проектирования приложения, комбинирование которых позволяет получить в результате законченное решение.

Книга написана в стиле самоучителя, с тем расчетом, что читатель сможет самостоятельно освоить технологию создания Windows-приложений. Для каждой рассматриваемой темы подробно показаны все этапы создания и редактирования кода программы. Большое внимание уделяется работе с MFC AppWizard (мастером создания приложения), который выполняет всю рутинную работу по созданию и изменению кода программы.

Также книга будет полезна и опытным программистам, т. к. включает большой объем справочной информации о библиотеке MFC. Вся справочная информация коррелированна со справочной системой Microsoft Visual Studio 2005 Documentation, входящей в состав пакета.

Каждая глава посвящена отдельной теме и состоит из описания последовательности действий с приведением фрагментов изменяемого кода и рисунков, поясняющих процесс создания этих изменений с помощью мастера. При использовании какого-либо компонента из библиотеки MFC приводится справочная информация по нему. Также дается дополнительная информация о некоторых компонентах, которые не приводятся в примерах, но могут быть полезны. К каждой главе прилагаются листинги с окончательными фрагментами изменений программы.

К книге прилагается CD-диск с полными исходными кодами и исполняемыми файлами всех программ. Каждой главе соответствует свой проект (подробное описание содержимого CD-диска приведено в *приложении 1*). Все проекты построены в среде Microsoft Visual Studio 2005, но могут быть легко

конвертируемы под среду Microsoft Visual Studio 2008, которая появилась в конце 2007 года.

Структура книги

Данная книга состоит из 15 глав.

- В *главе 1* рассматривается создание простого однодокументного (SDI) приложения с описанием построения готовой программы и способами нахождения и исправления ошибок в листинге. На примере этого приложения объясняется структура приложения MFC, говорится об обработке сообщений приложения. Рассказывается о ресурсах программы. Далее рассматриваются возможности изменения кода, построенного мастером: изменение заголовка, цвета и размера окна, изменение иконки приложения, изменение курсора, добавление горячих клавиш и использование альтернативного выбора пунктов меню.
- В *главе 2* рассматривается работа с графическими объектами: текстом (шрифтами), пером, кистью и фигурами.
- В *главе 3* рассказывается о добавлении дочерних объектов (рисунков и кнопок) в окно представления и о способах работы с курсором.
- В *главе 4* описываются различные способы работы с меню приложения.
- В *главе 5* говорится о проблемах перерисовки окон и использовании виртуального окна, о масштабировании изображения, создании и использовании полосы прокрутки, об обработке нажатия клавиш. Там же объясняется, как создавать свои дочерние окна.
- В *главах 6—8* рассматриваются все элементы управления, предоставляемые набором инструментов редактирования диалоговых окон. В *главе 6* также рассказывается о создании модальных и немодальных диалоговых окон. В *главе 7* обсуждается использование расширенной кодировки Unicode (чтобы показать разные возможности, в *главах 1—7* был использован Unicode, а в *главах 8—15* — нет).
- В *главе 9* показана работа с панелью инструментов и строкой статуса.
- В *главе 10* описывается архитектура документ/представление, создание окон с помощью макроса `RUNTIME_CLASS` и все возможные способы связи между ними.
- В *главе 11* показано, как создавать графическое изображение с использованием обработки сообщений мыши, как сохранять это изображение на диске

с помощью метафайла. Рассмотрена работа со стандартным диалоговым окном выбора файла (**Открыть** (Open), **Сохранить как** (Save As)).

- В *главе 12* продолжается обсуждение способов сохранения данных на диске, но уже с использованием архива.
- В *главе 13* рассказывается о возможных видах окна представления на примере разделения его на две части (верхняя часть работает с графическими данными, а в нижней части они отображаются в текстовом виде — в виде значений координат). Там же кратко рассматриваются и другие виды окон представления.
- В *главе 14* происходит логичный переход примера из *главы 13* к построению многодокументного (MDI) приложения, где графический и текстовый форматы представлены в разных окнах. Рассмотрена архитектура MDI-приложения.
- В *главе 15* рассказано о создании и работе справочной системы приложения.

Соглашения, принятые в книге

В данной книге используется специальное форматирование текста для выделения некоторых фрагментов. Далее приведены основные принципы форматирования текста.

- Тексты фрагментов программ выделяются шрифтом `Courier`.
- Изменения в листингах, сделанные с помощью мастера, выделяются шрифтом `Courier на сером фоне`.
- Изменения в листингах, внесенные непосредственно программистом, выделяются **`полужирным шрифтом Courier`**.
- Если в тексте встречается имя класса, функции или переменной, оно выделяется шрифтом `Courier`.
- Заголовки окон, команды меню и названия кнопок выделяются **полужирным шрифтом**.
- Выбор пункта выпадающего меню показан с помощью символа (|), например, **Файл | Открыть**.
- Название клавиши заключается в угловые скобки (<Enter>). Если требуется нажать комбинацию клавиш, они объединяются знаком (+), например, <Ctrl>+<C>.

- Слова и предложения, на которые надо обратить особое внимание, выделяются *курсивом*.
- В книге встречаются абзацы, текст которых выделен специальным шрифтом. Это примечание, например:

> ***ПРИМЕЧАНИЕ***
>
> Примечания содержат полезную и интересную информацию, которая позволяет более подробно изложить отдельные детали.

- Справочная информация по функциям комментируется следующим образом:

```
int NameFunctions(     // Возвращаемое значение (если оно есть)
   int arg1,           // Описание аргумента
   int arg2);          // Описание аргумента
```

- Названия элементов пользовательского интерфейса среды Visual Studio (пунктов меню, названия мастеров, окон и т. п.) даны с дословным переводом в скобках, т. к. версия не русифицирована. Заголовки меню, окон и т. п. системы Windows приведены для русифицированной версии Windows с указанием английского варианта в скобках.

Благодарности

В первую очередь я хотела бы поблагодарить моего мужа — Сидорина Юрия Сергеевича и дочку — Сидорину Елену Юрьевну, за поддержку и терпение во время моей работы над книгой.

Отдельная благодарность моему коллеге в области программирования, к. т. н. — Пышкину Евгению Валерьевичу, за дельные советы и помощь в выборе и связи с издательством.

Хорошая книга не может быть написана в вакууме. Я хотела бы поблагодарить моих студентов Санкт-Петербургского государственного политехнического университета, которые в процессе освоения данного материала задавали много полезных и интересных вопросов. Ответы на эти вопросы повлияли на методику изложения и структуру данной книги.

Также хочется сказать большое спасибо редактору издательства "БХВ-Петербург" Седых Нине Валерьевне за качественную и технически грамотную работу над книгой.

ГЛАВА 1

Создание простого приложения MFC

1.1. Создание проекта

Применение MFC для создания Windows-приложений позволяет использовать целые блоки заранее написанного (компанией Microsoft) и готового к работе кода, что значительно упрощает и ускоряет создание программы. Мастер построения приложения создает основной шаблон приложения, используя настройки, выбираемые программистом при создании проекта.

Для создания приложения MFC надо запустить программу Microsoft Visual Studio 2005 (при стандартной установке это выполняется с помощью системного меню **Пуск | Программы | Microsoft Visual Studio 2005 | Microsoft Visual Studio 2005** (Start | Programs | Microsoft Visual Studio 2005 | Microsoft Visual Studio 2005)). После запуска появится стартовая страница (**Start Page**) (рис. 1.1). Чтобы создать новый проект, надо на стартовой странице в строке **Create** (Создать) нажать ссылку **Project** (Проект) или выполнить команду меню **File | New | Project** (Файл | Новый | Проект), как показано на рис. 1.1.

Затем в появившемся окне **New Project** (Новый проект) надо задать тип проекта, его местоположение и название:

1. В дереве **Project types** (Тип проекта) надо выбрать лист **MFC** и в области **Templates** (Шаблоны) — **MFC Application** (MFC-приложение) (рис. 1.2, *а*).
2. С помощью кнопки **Browse** (Просмотр) выбрать местоположение создаваемого проекта:
   - в открывшемся окне **Project Location** (Расположение проекта) выбрать нужный диск и папку для проекта, причем поле **Folder name** (Имя папки) заполнять не надо (рис. 1.2, *б*);

- при нажатии кнопки **Open** (Открыть) выбранное местоположение появится в поле **Location** (Расположение) (рис. 1.2, *в*).

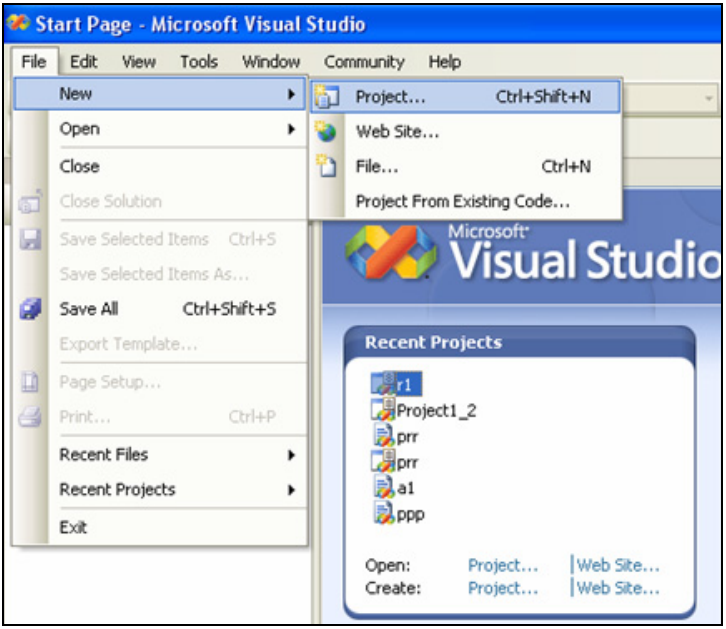


Рис. 1.1. Создание нового проекта

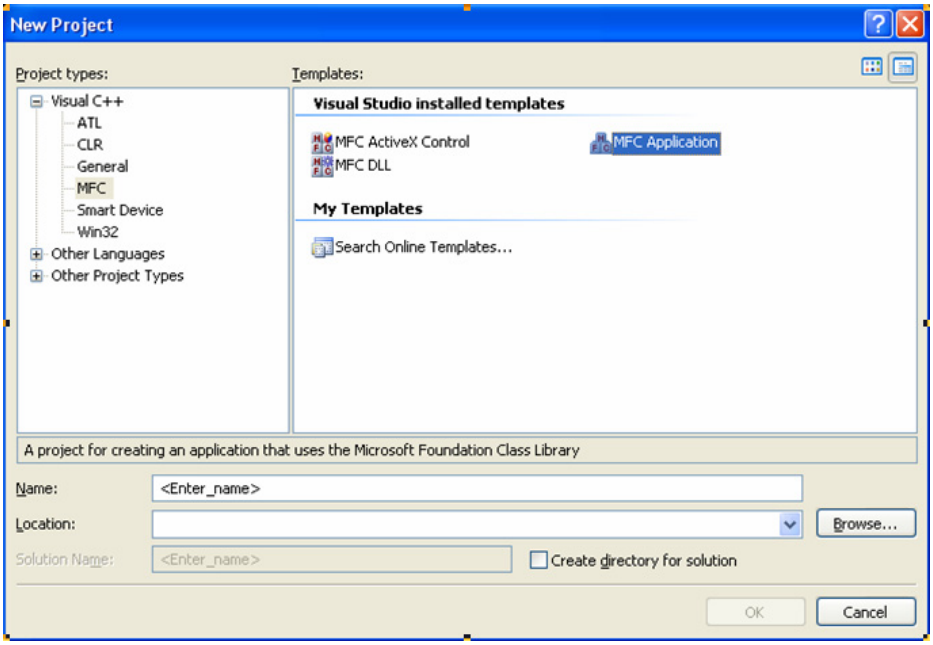
3. Задать в поле **Name** (Имя) имя проекта — **pr1** (рис. 1.2, *в*) и нажать кнопку **OK**.

После этого появится окно для выбора свойств приложения **MFC Application Wizard** (Мастер создания приложения) со следующими вкладками:

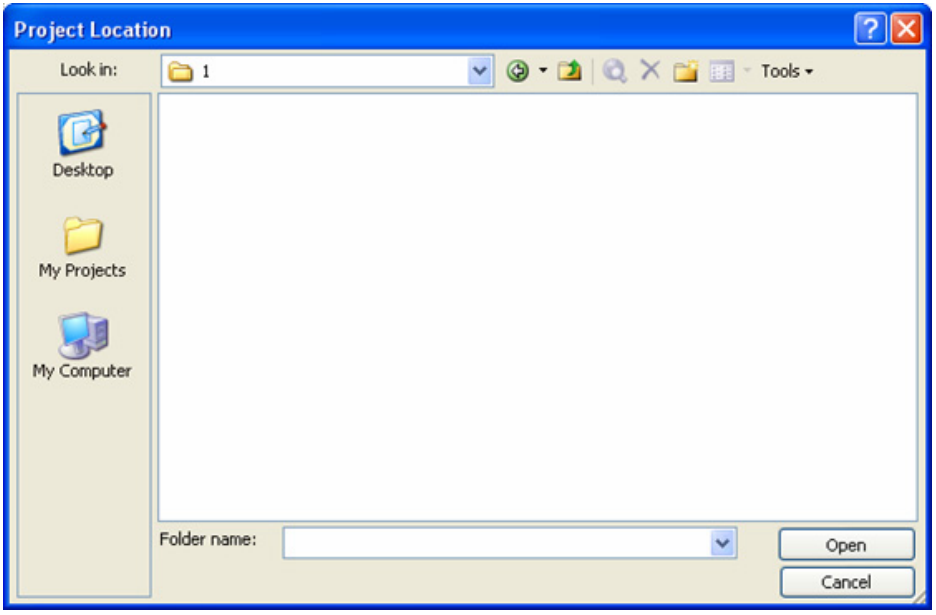
1. **Overview** (Обзор) — здесь перечислены установки, предлагаемые мастером по умолчанию (многодокументное приложение, без поддержки работы с базой данных, без поддержки работы со смешанными документами) (рис. 1.3). Чтобы создать простое однодокументное приложение, эти установки надо изменить.

Примечание

Для выбора нужной вкладки надо щелкнуть по ее названию в левом списке или воспользоваться кнопками последовательного перехода между вкладками **Next** (Следующая) и **Previous** (Предыдущая).

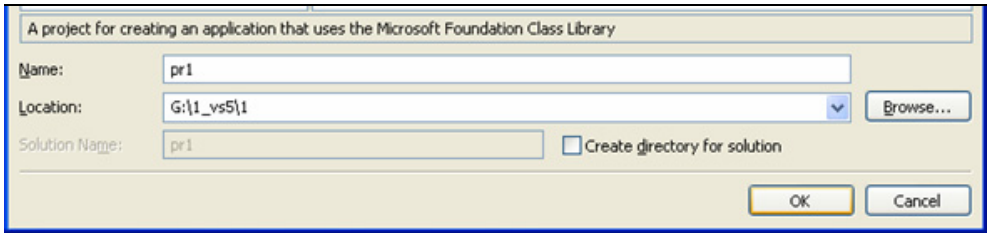


а



б

Рис. 1.2. Создание проекта: выбор шаблона (*а*), местоположения проекта (*б*)



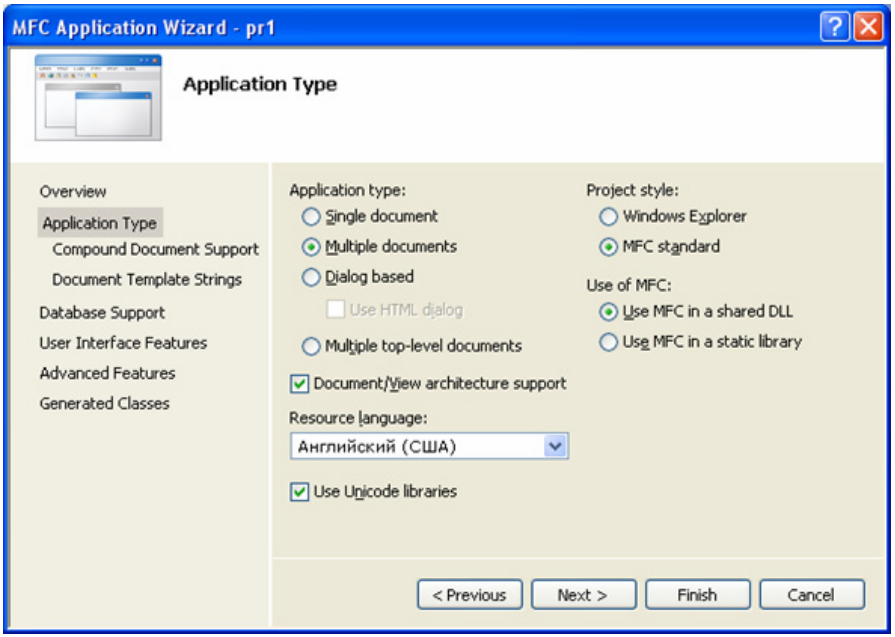
в

Рис. 1.2. Задание имени проекта (*в*)

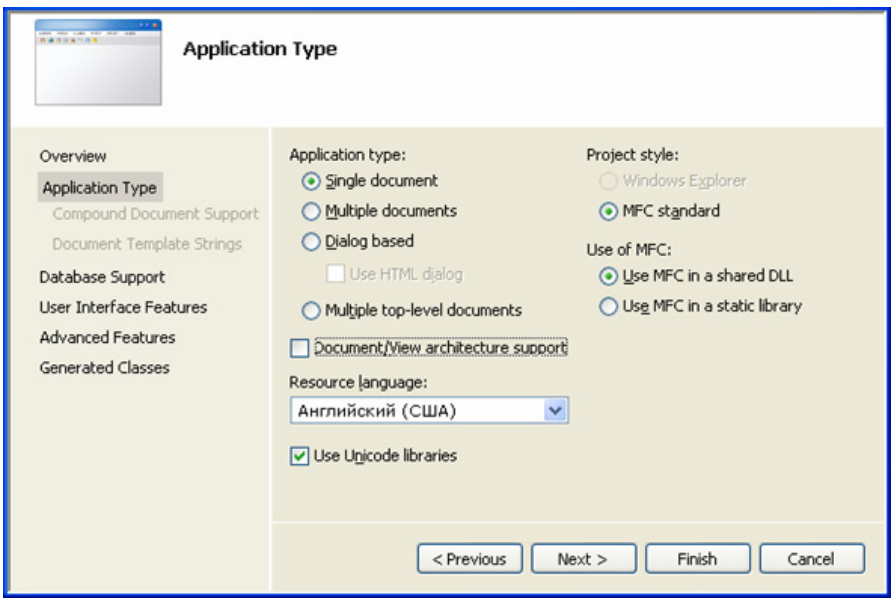
Рис. 1.3. Установки проекта, предлагаемые мастером по умолчанию

2. **Application Type** (Тип приложения) — на этой вкладке задаются следующие настройки (рис. 1.4):

   - положение переключателя **Application type** вместо **Multiple document** (Многодокументное приложение) надо изменить на **Single document** (Однодокументное приложение). Работа с многодокументным приложением будет рассмотрена в *гл. 14*;



а



б

Рис. 1.4. Настройки типа приложения: по умолчанию (*а*) и измененные (*б*)

- флажок **Document/View architecture support** (Поддержка архитектуры документ/представление) для упрощения кода надо снять. Работа с поддержкой архитектуры документ/представление будет рассмотрена в *гл. 10*;
- в раскрывающемся списке **Resource language** (Язык ресурсов приложения) надо оставить значение **Английский (США)** (English (USA)). Хотя выбран английский язык, ресурсы приложения можно будет задавать и на русском языке;
- о флажке **Use Unicode libraries** (Использование расширенного кода) будет подробно рассказано в *гл. 7*;
- переключатель **Project style** (Стиль проекта) оставить в положении **MFC standard** (Стандарт MFC). Этим переключателем задается внешний вид окон приложения: обычный (MFC standard) или в стиле браузера Internet (Windows Explorer). Изменение этой опции возможно только при использовании архитектуры документ/представление;
- переключатель **Use of MFC** (Использование MFC) по умолчанию установлен в положение **Use MFC in a shared DLL** (Использование динамически подключаемых библиотек). В этом случае размер выполняемого файла будет небольшим, но программа не сможет работать на компьютерах, где не установлен пакет Microsoft Visual Studio 2005 из-за отсутствия динамически подключаемых библиотек. Если выбрать опцию **Use MFC in a static library** (Использование статических библиотек), то программа будет легко переносима на другие компьютеры, но размер выполняемого файла будет намного больше, т. к. в него будут включены все необходимые библиотеки. В любом случае, выбор типа библиотек всегда можно изменить в свойствах уже построенного проекта (об этом будет рассказано в *гл. 7*).

3. **Database Support** (Поддержка базы данных) — на этой вкладке надо оставить переключатель в положении **None** (Без поддержки) (рис. 1.5). Наше приложение не будет взаимодействовать с базой данных.
4. **User Interface Features** (Возможности пользовательского интерфейса) — на этой вкладке задается следующее (рис. 1.6, *а*):
   - **Thick frame** (Толстая граница фрейма) — утолщенная рамка окна приложения, позволяющая менять размеры окна;
   - **Minimize box** — окно приложения будет иметь кнопку **Свернуть** (Minimize) в правом верхнем углу;

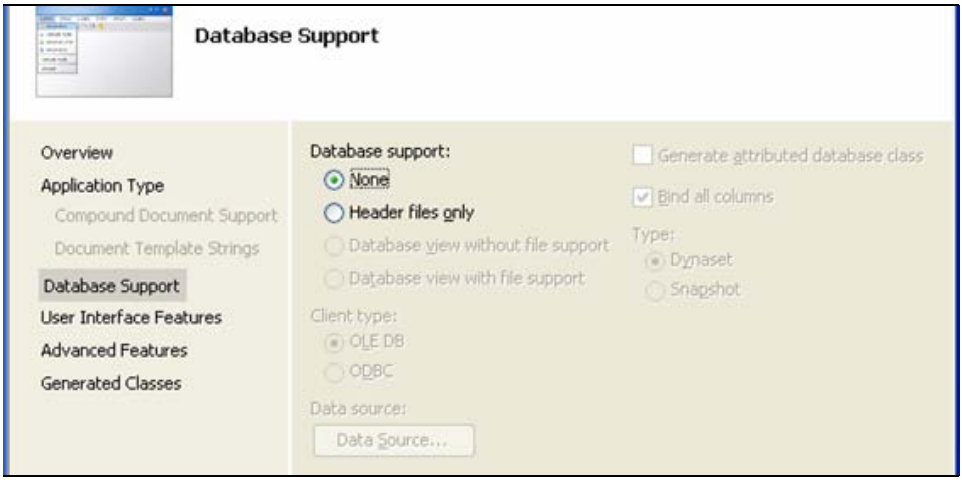
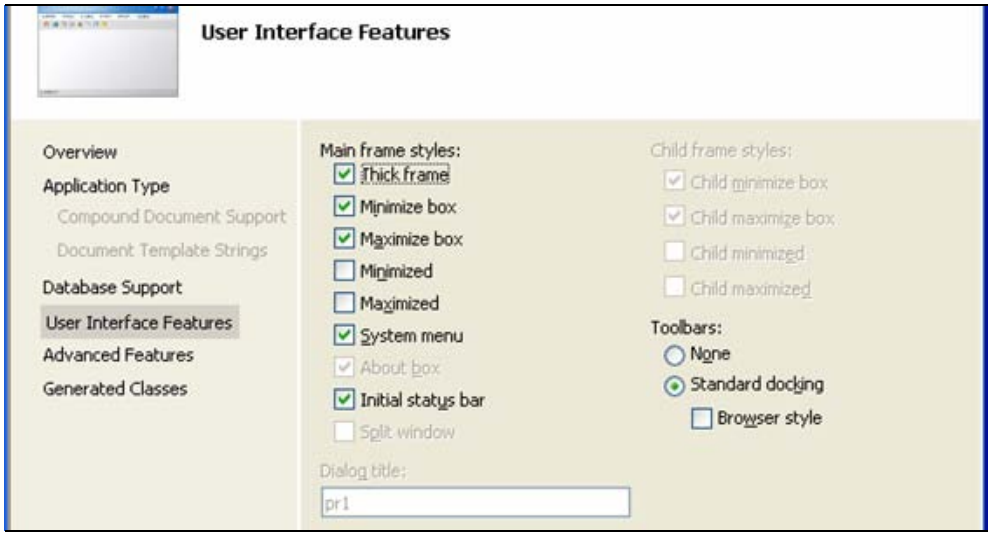
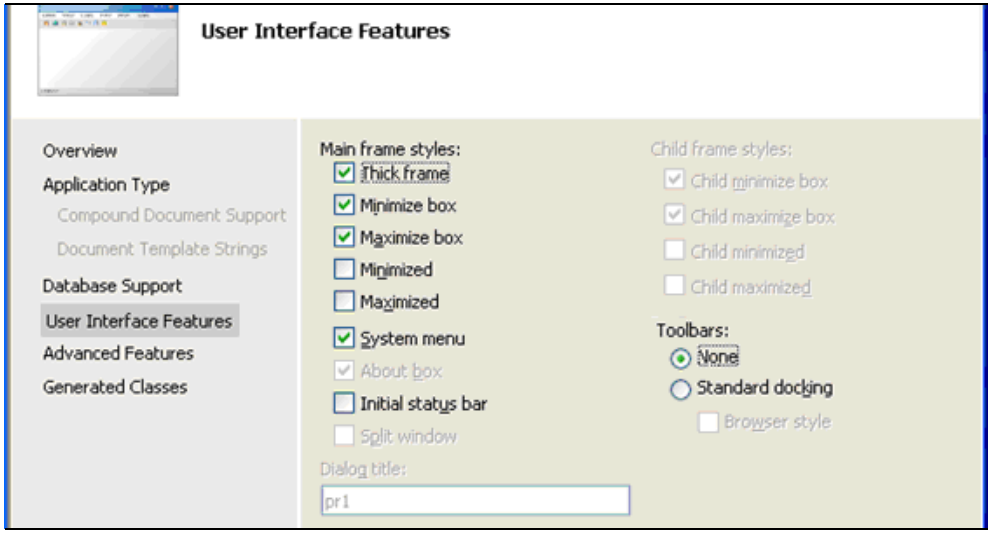


Рис. 1.5. Настройка поддержки базы данных по умолчанию

- **Maximize box** — окно имеет кнопку **Развернуть** (Maximize) в правом верхнем углу;
- **Minimized** (Минимизированное) — при запуске приложения окно будет свернуто в пиктограмму. По умолчанию этот флажок снят;
- **Maximized** (Максимизированное) — при запуске приложения окно будет развернуто во весь экран. По умолчанию этот флажок снят;
- **System menu** (Системное меню) — при нажатии в левый верхний угол окна приложения (на иконку) будет появляться системное меню для окна: (**Восстановить** (Restore), **Переместить** (Move), **Размер** (Size), **Свернуть** (Minimize), **Развернуть** (Maximize), **Закрыть** (Close));
- **Initial status bar** (Инициализация строки статуса) — окно приложения будет иметь строку статуса. Обычно это небольшое (в одну строку) серое поле внизу окна, где появляются подсказки и отражаются состояния клавиш <Caps Lock>, <Ins> и др. О работе со строкой статуса будет рассказано в *гл. 9*. Для упрощения кода этот флажок надо снять (рис. 1.6, *б*);
- переключатель **Toolbars** (Панель инструментов) — положение **Standard docking** (Стандартная привязка) указывает на то, что окно приложения будет иметь панель инструментов со стандартной привязкой к окну. Это панель с кнопками, дублирующими пункты меню, и находящаяся обычно под меню.



а



б

Рис. 1.6. Настройки пользовательского интерфейса: по умолчанию (*а*) и измененные (*б*)

О работе с панелью инструментов будет рассказано в *гл. 9*. Для упрощения кода надо отказаться от панели инструментов, выбрав положение переключателя **None** (Нет) (рис. 1.6, *б*).

5. **Advanced Features** (Дополнительные возможности) — настройки на этой вкладке оставляем по умолчанию (рис. 1.7):

   - **Context-sensitive Help** (Контекстная справка) — при установке этого флажка в приложение будет автоматически включена справочная система. Работа со справочной системой рассмотрена в *гл. 15*. По умолчанию этот флажок снят;
   - **ActiveX controls** (Элементы управления ActiveX) — выбор этого флажка дает возможность использовать элементы управления ActiveX (внедренные элементы управления других приложений, например, таблицы Excel);
   - **Windows sockets** (Оконные сокеты) — при установке этого флажка приложение может иметь непосредственный доступ к Internet, используя протоколы FTP и HTTP. По умолчанию этот флажок снят;
   - **Active Accessibility** (Активная доступность) — установка этого флажка делает доступным работу с объектами COM (Component Object Model, Модель компонентных объектов);

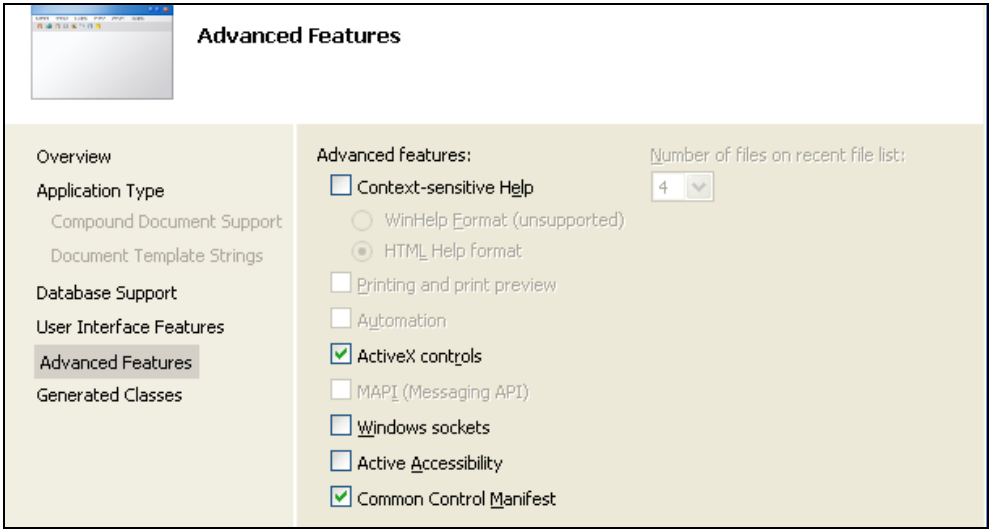


Рис. 1.7. Настройка дополнительных возможностей по умолчанию

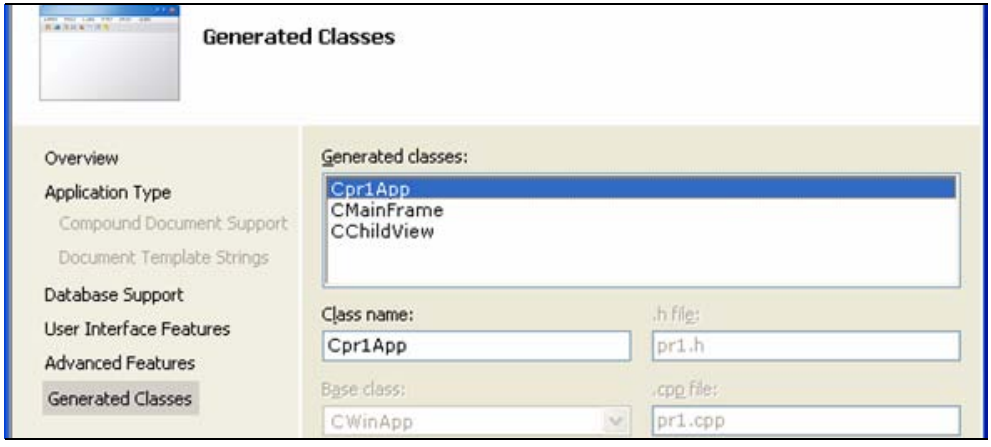
   - **Common Control Manifest** (Объявление общих элементов управления) — установка этого флажка дает возможность программного вы-

бора состава элементов управления, используемых в приложении *(см. разд. 1.6.1)*.

6. **Generated Classes** (Сгенерированные классы) — на этой вкладке можно изменить имена классов приложения и названия файлов для их размещения (при включенной поддержке архитектуры документ/представление здесь можно выбирать типы базовых классов, о чем будет рассказано в *гл. 13*).

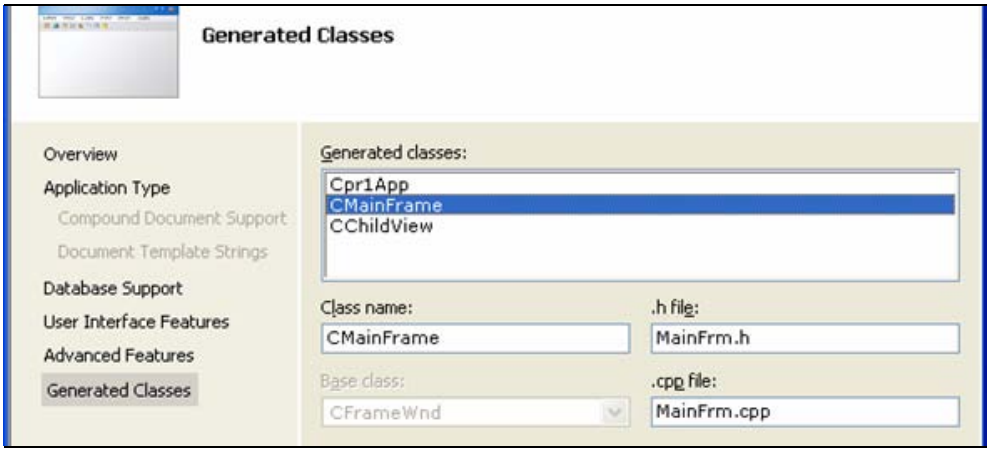
   - **Cpr1App** (производный класс от `CWinApp`) — класс приложения. Файлы: pr1.h — объявление класса, pr1.cpp — определение класса (рис. 1.8, *а*);
   - **CMainFrame** (производный класс от `CFrameWnd`) — класс фрейма (класс главного окна приложения, состоящего из рамки, заголовка окна и меню). Файлы: MainFrm.h — объявление класса, MainFrm.cpp — определение класса (рис. 1.8, *б*);
   - **CChildView** (производный класс от `CWnd`) — класс представления (класс внутреннего содержимого окна фрейма). Файлы: ChildView.h — объявление класса, ChildView.cpp — определение класса (рис. 1.8, *в*).

После просмотра и выбора необходимых настроек надо нажать кнопку **Finish** (Завершение) (рис. 1.8, *в*). Мастер завершит свою работу и сгенерирует все необходимые файлы проекта.

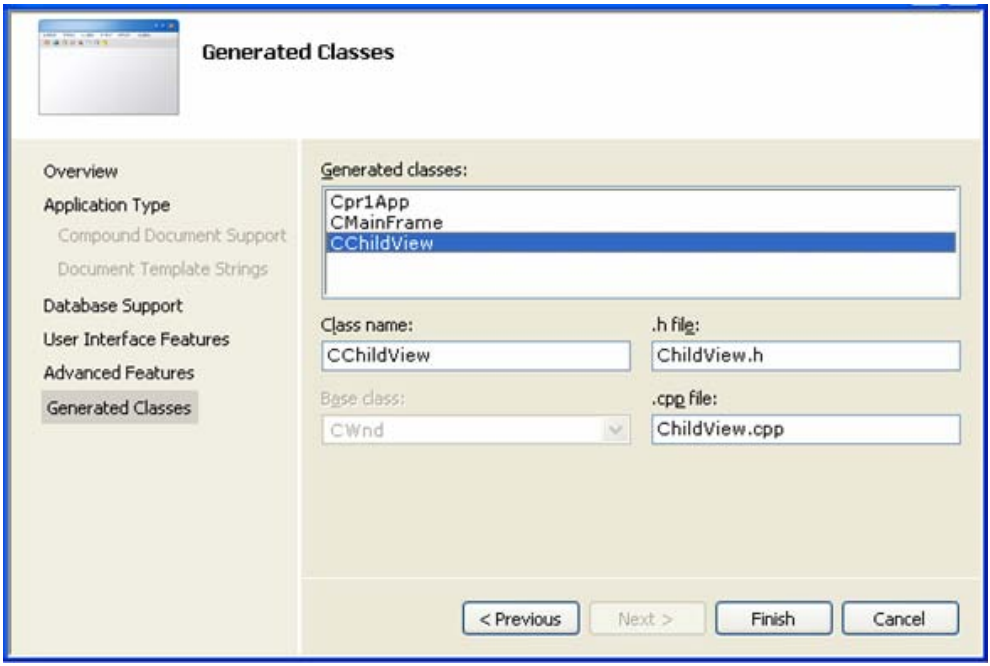


а

Рис. 1.8. Классы, сгенерированные мастером: класс приложения (*а*)



б

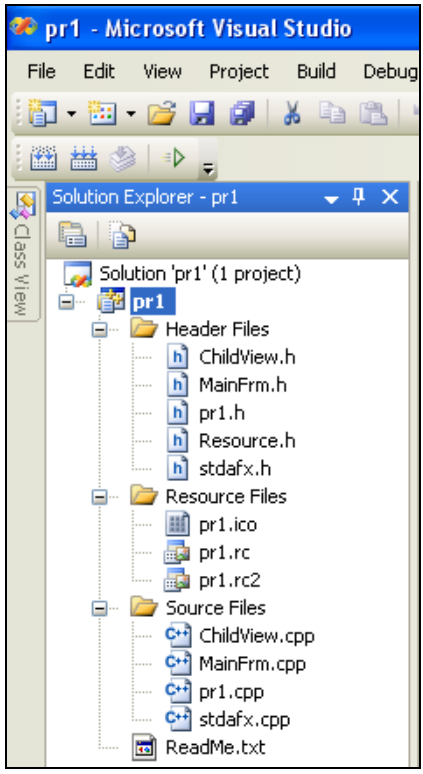


в

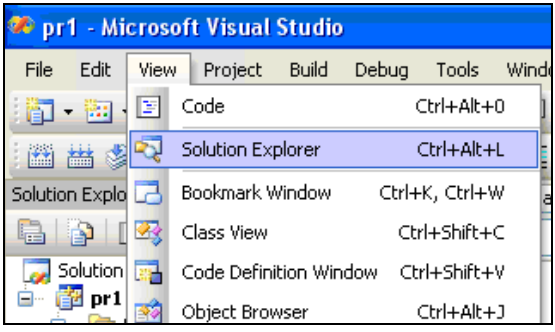
Рис. 1.8. Класс окна фрейма (*б*) и класс окна представления (*в*)

1.2. Файлы проекта

После создания проекта появится окно **Solution Explorer** (Окно файлов проекта), показанное на рис. 1.9, *а*.



а



б

Рис. 1.9. Окно файлов проекта (*а*)
и открытие окна файлов проекта с помощью меню (*б*)

Если окно не появилось, его можно открыть самостоятельно, выполнив команду меню **View | Solution Explorer** (Просмотр | Окно файлов проекта) (рис. 1.9, *б*). Любой файл проекта можно открыть в окне редактирования, дважды щелкнув по нему левой кнопкой мыши. Файл pr1.ico — иконка приложения (см. рис. 1.11, *в*).

Файл pr1.rc — ресурсы приложения. Это обычный текстовый файл, но для удобства работы с ним, он (по умолчанию) открывается в специальном редакторе ресурсов **Resource View** (рис. 1.10). Файл pr1.rc2 является служебным и программистом не используется.

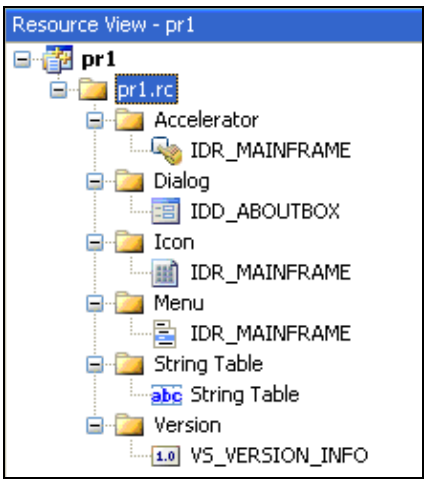
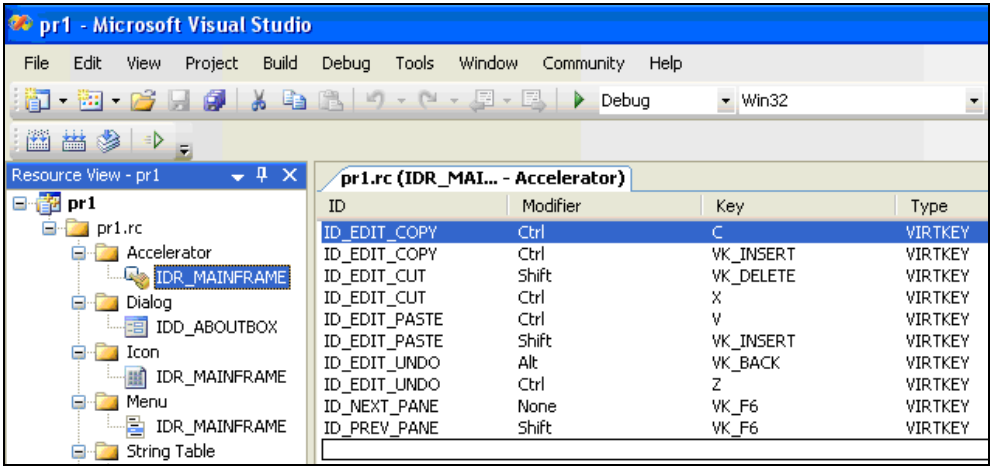


Рис. 1.10. Ресурсы приложения

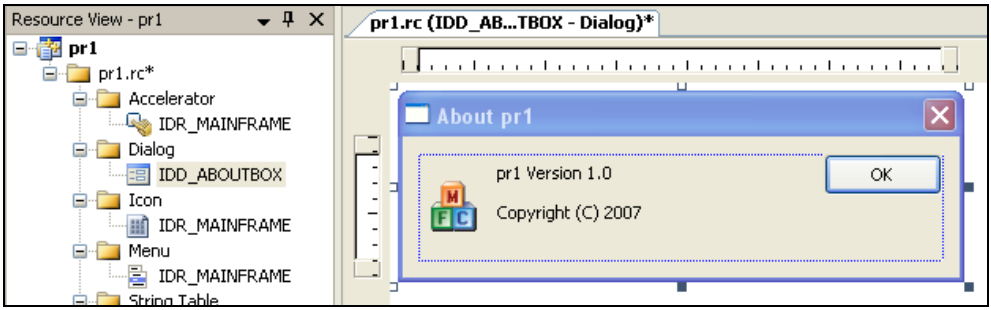
Ресурсы приложения состоят из:

1. **Accelerator** (Акселераторы) — список горячих клавиш для выполнения пунктов меню (рис. 1.11, *а*). Например, для выполнения команды меню **Edit | Copy** (Редактирование | Копировать) можно использовать быструю комбинацию клавиш <Ctrl>+<C>.
2. **Dialog** (Диалог) — окно диалога **About pr1** (О программе), которое будет появляться при выборе команды меню приложения **Help | About pr1** (Помощь | О программе) (рис. 1.11, *б*).

3. **Icon** (Иконка) — иконка приложения. Это не одна, а целый набор иконок разных цветов и разрешений, предназначенных для возможной работы приложения с различными видеокартами (рис. 1.11, *в*).
4. **Menu** (Меню) — главное меню приложения (рис. 1.11, *г*).
5. **String Table** (Таблица строк) — все строковые ресурсы приложения (заголовок окна приложения, подсказки пунктов меню и т. п.) (рис. 1.11, *д*).
6. **Version** (Версия приложения) — служебная информация о версии продукта, его названии, названии компании и т. п. (рис. 1.11, *е*).

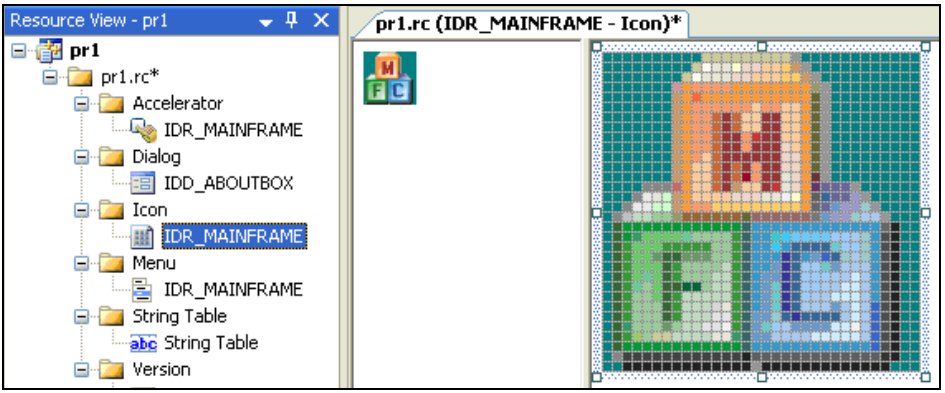


а

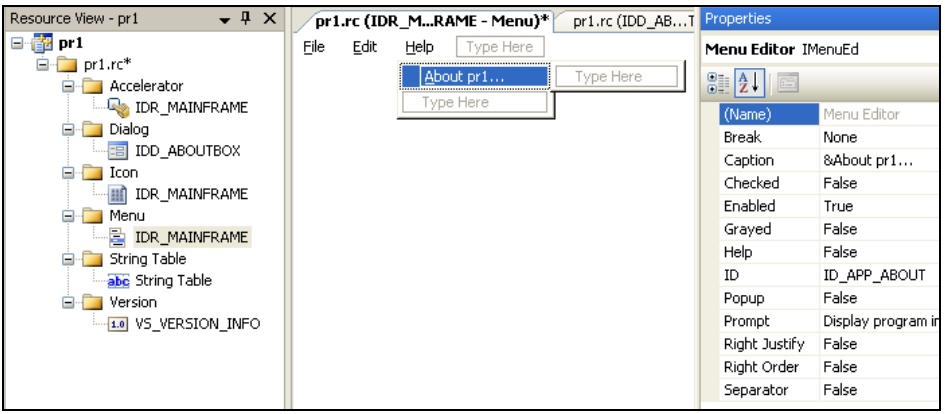


б

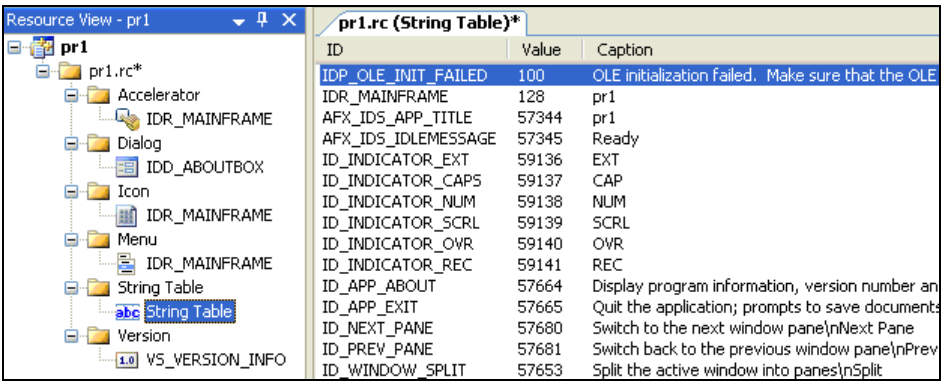
Рис. 1.11. Акселераторы (*а*), диалоговое окно (*б*)



в

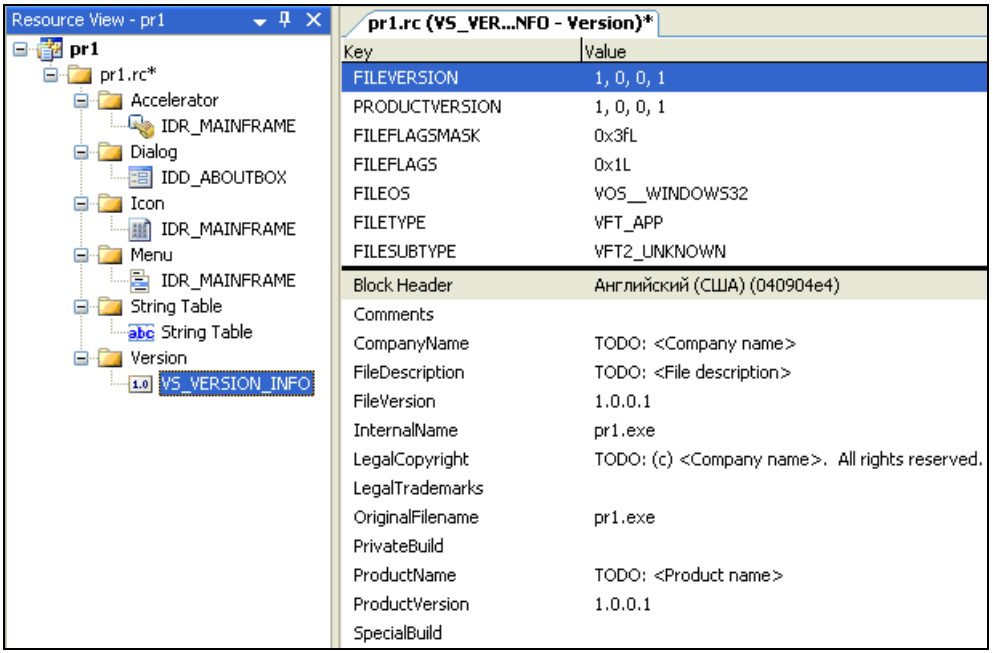


г



д

Рис. 1.11. Иконка (*в*), меню (*г*), таблица строк (*д*)



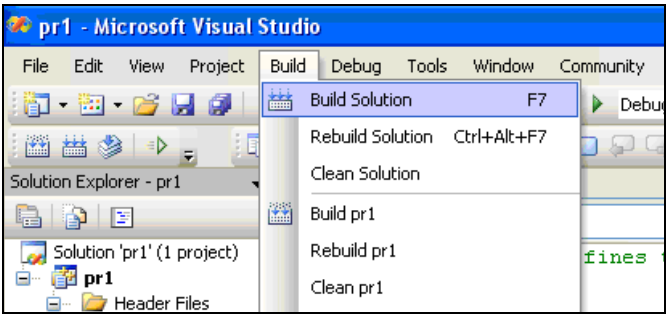
е

Рис. 1.11. Версия (*е*) приложения

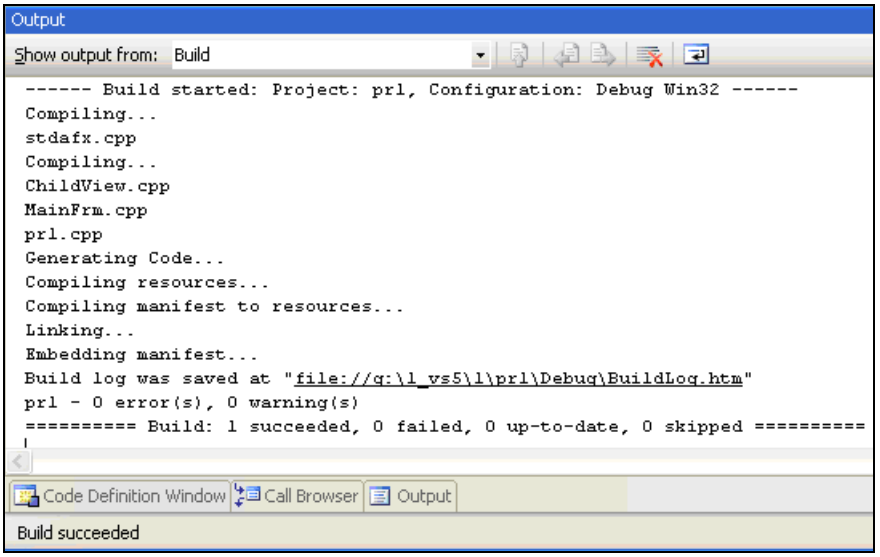
1.3. Создание выполняемого файла и запуск приложения

Для создания выполняемого файла надо вызвать команду меню **Build | Build Solution** (Построение | Построить приложение) (рис. 1.12, *а*) или нажать клавишу <F7>. После этого в окне **Output** (Результат) появятся результаты построения (рис. 1.12, *б*). Если окно **Output** не появилось, его можно открыть самостоятельно, выполнив команду меню **View | Output** (рис. 1.12, *в*).

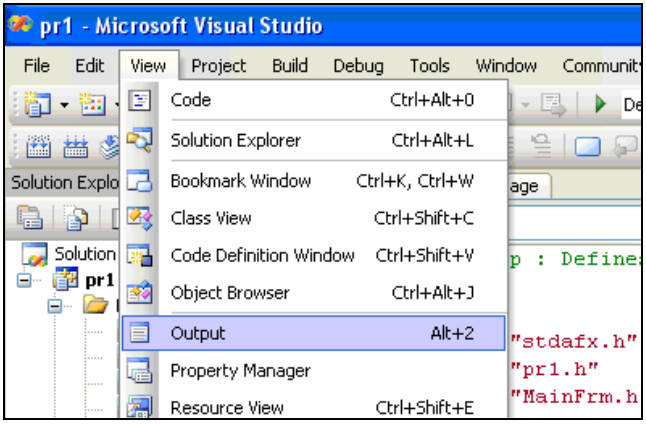
Если в тексте программы были допущены ошибки, они будут отображены в окне **Output** (рис. 1.12, *г*). Щелкнув два раза по тексту ошибки (в окне **Output**), можно попасть в то место программы, где эта ошибка была допущена, и исправить ее. После исправления ошибок надо снова построить программу.



а

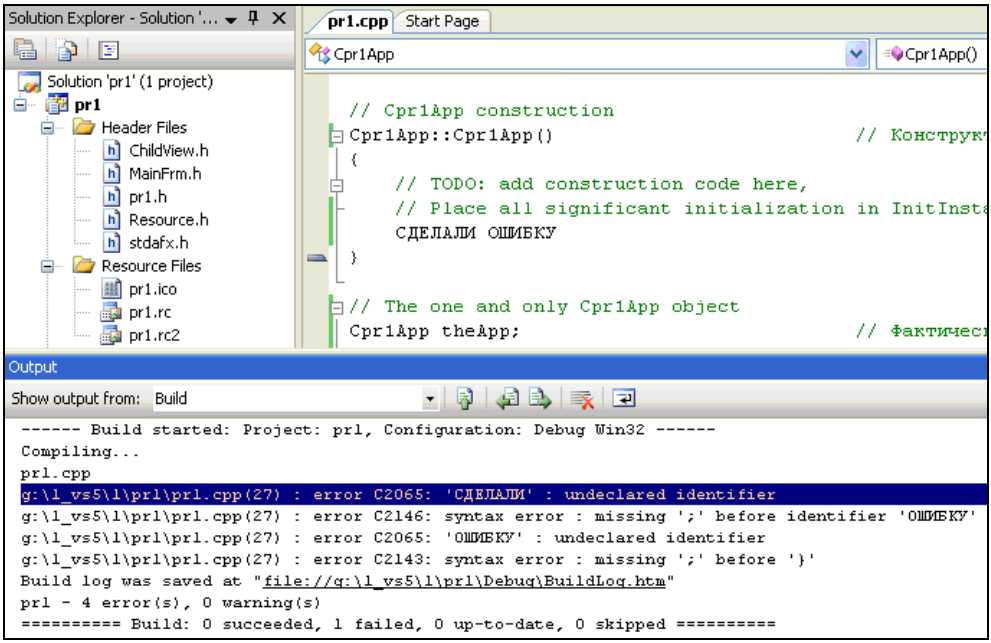


б



в

Рис. 1.12. Построение готового приложения (*а*), результаты построения приложения (*б*) и открытие окна результатов построения приложения с помощью меню (*в*)



г

Рис. 1.12. Исправление ошибок в листинге программы (*г*)

Для запуска построенной программы надо выполнить команду меню **Debug | Start Without Debugging** (Отладка | Запуск без отладчика) (рис. 1.13).

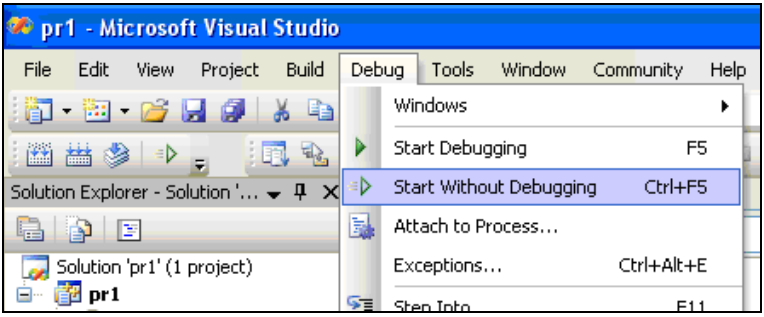


Рис. 1.13. Запуск программы

Результаты работы программы показаны на рис. 1.14.

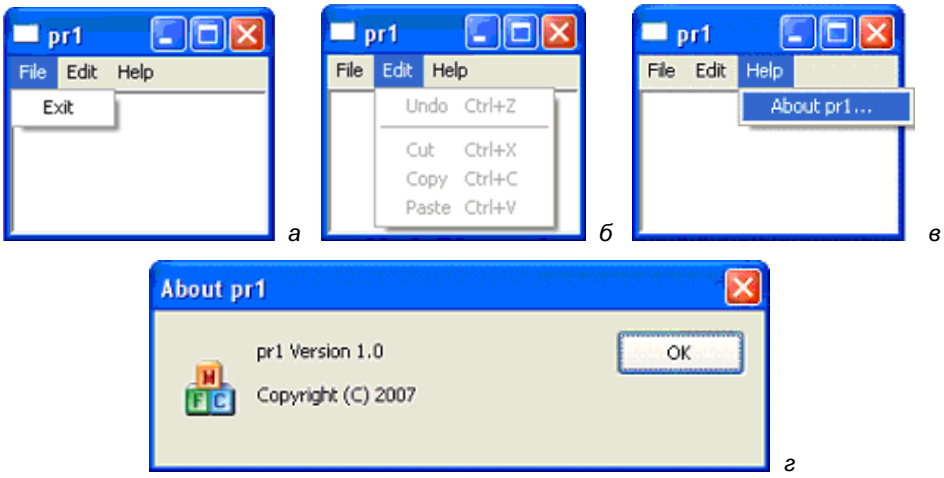


Рис. 1.14. Результат работы программы:
команды меню **File** (*а*), **Edit** (*б*), вызов диалогового окна **About pr1**
с помощью команды **Help** (*в*) и окно **About pr1** (*г*)

1.4. Архитектура приложения

Архитектура простого SDI-приложения (Single Document Interface — однодокументный интерфейс) изображена на рис. 1.15. Структура классов приложения такая:

```
class Cpr1App : public CWinApp     // Объект приложения
{
   CWnd* CWinThread::m_pMainWnd;  // Указатель (наследуемый) на фреймовое
                                  // окно (главное окно приложения)
};
class CMainFrame : public CFrameWnd  // Фреймовое окно
{
   CChildView m_wndView              // Объект окна представления
};
class CChildView : public CWnd       // Окно представления
{
};
```

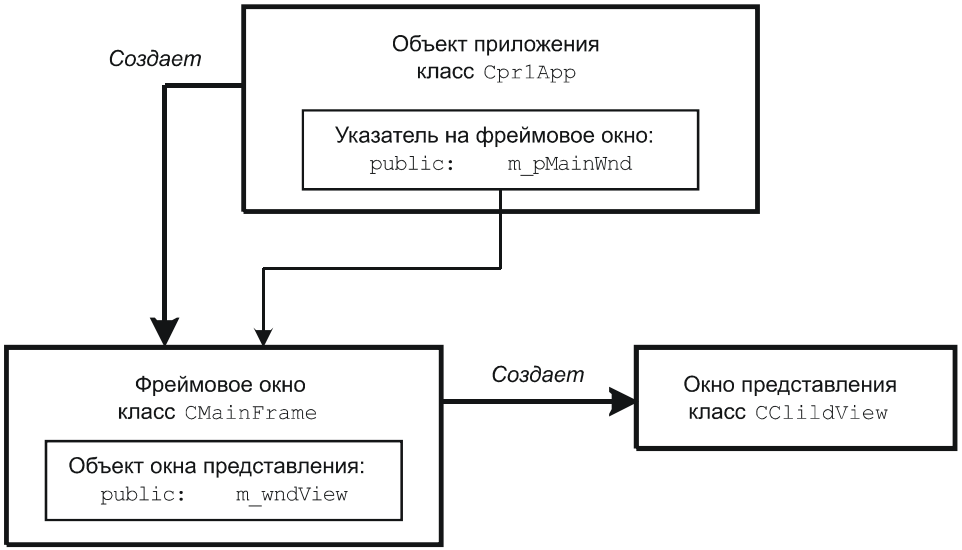


Рис. 1.15. Архитектура простого SDI-приложения

Примечание

Фреймовое окно является только частью объекта, на который указывает `m_pMainWnd`. Поэтому, чтобы получить доступ к членам класса `CMainFrame`, надо воспользоваться явным указателем на этот класс и преобразованием типов:

```
CMainFrame *pframe = (CMainFrame *)m_pMainWnd;
```

1.5. Листинги программы

Здесь приведены листинги программы, которые были сгенерированы мастером создания приложения: объявление класса приложения (листинг 1.1), определение класса приложения (листинг 1.2), объявление класса окна фрейма (листинг 1.3), определение класса окна фрейма (листинг 1.4), объявление класса окна представления (листинг 1.5), определение класса окна представления (листинг 1.6), идентификаторы приложения (листинг 1.7), ресурсы приложения (листинг 1.8). Для упрощения понимания кода, в ключевых местах листингов были добавлены комментарии на русском языке.

Листинг 1.1. Файл pr1.h (объявление класса приложения)

```
// pr1.h : main header file for the pr1 application
//
#pragma once
#ifndef __AFXWIN_H__
   #error "include 'stdafx.h' before including this file for PCH"
#endif
#include "resource.h"               // main symbols
// Cpr1App:
// See pr1.cpp for the implementation of this class
//
class Cpr1App : public CWinApp      // Класс приложения
{
public:
   Cpr1App();
// Overrides
public:
   virtual BOOL InitInstance();
// Implementation
public:
   afx_msg void OnAppAbout();
   DECLARE_MESSAGE_MAP()
};
extern Cpr1App theApp;
```

Листинг 1.2. Файл pr1.cpp (определение класса приложения)

```
// pr1.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "pr1.h"
#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
// Cpr1App
BEGIN_MESSAGE_MAP(Cpr1App, CWinApp)      // Карта сообщений Cpr1App
                                         // Обработка выбора меню "About"
   ON_COMMAND(ID_APP_ABOUT, &Cpr1App::OnAppAbout)
END_MESSAGE_MAP()

Cpr1App::Cpr1App()               // Конструктор
{
   // TODO: add construction code here,
   // Place all significant initialization in InitInstance
}

Cpr1App theApp;                  // Создание объекта приложения
                                 // (фактически, начало работы программы)

BOOL Cpr1App::InitInstance()     // Инициализация приложения Cpr1App
{
   // InitCommonControlsEx() is required on Windows XP if an application
   // manifest specifies use of ComCtl32.dll version 6 or later to enable
   // visual styles.  Otherwise, any window creation will fail.
   INITCOMMONCONTROLSEX InitCtrls;
   InitCtrls.dwSize = sizeof(InitCtrls);
   // Set this to include all the common control classes you want to use
   // in your application.
   InitCtrls.dwICC = ICC_WIN95_CLASSES;
   InitCommonControlsEx(&InitCtrls);  // Загрузка динамической библиотеки
                                      // элементов управления
                                      // (кнопки и т. д.)

   CWinApp::InitInstance();           // Вызов функции инициализации
                                      // приложения базового класса

   // Инициализация OLE-библиотеки
   if (!AfxOleInit())
   {
```

```
      AfxMessageBox(IDP_OLE_INIT_FAILED);
      return FALSE;
   }
   AfxEnableControlContainer();      // Разрешить в приложении поддержку
                                     // элементов управления OLE
   // Standard initialization
   // If you are not using these features and wish to reduce the size
   // of your final executable, you should remove from the following
   // the specific initialization routines you do not need
   // Change the registry key under which our settings are stored
   // TODO: You should modify this string to be something appropriate
   // such as the name of your company or organization
   // Сохранение параметров начальной установки приложения в системном
   // реестре
   SetRegistryKey(_T("Local AppWizard-Generated Applications"));

   // To create the main window, this code creates a new frame window
   // object and then sets it as the application's main window object
   CMainFrame* pFrame = new CMainFrame;
   if (!pFrame)
      return FALSE;
   m_pMainWnd = pFrame;              // Указатель на главное окно
                                     // приложения
   // create and load the frame with its resources
   pFrame->LoadFrame(                // Создание главного окна приложения
      IDR_MAINFRAME, WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, NULL, NULL);
   pFrame->ShowWindow(SW_SHOW);      // Задание вида отображения окна
   pFrame->UpdateWindow();           // и его перерисовка

   // call DragAcceptFiles only if there's a suffix
   //  In an SDI app, this should occur after ProcessShellCommand
   return TRUE;
}
// Cpr1App message handlers
```

```
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog     // Класс диалогового окна About
{
public:
   CAboutDlg();
// Dialog Data
   enum { IDD = IDD_ABOUTBOX };
protected:
   // Поддержка динамического обмена данными DDX/DDV
   virtual void DoDataExchange(CDataExchange* pDX);
// Implementation
protected:
   DECLARE_MESSAGE_MAP()
};

// Конструктор класса окна диалога (вызывает конструктор базового
// класса и передает ему идентификатор ресурса диалога)
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
   // Вызов функции обмена данными базового класса
   CDialog::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog) // Карта сообщений CAboutDlg
END_MESSAGE_MAP()

// App command to run the dialog
// Функция обработки выбора меню Help | About pr1 (см. карту
// сообщений Cpr1App в начале этого файла)
void Cpr1App::OnAppAbout()
{
   CAboutDlg aboutDlg;
   aboutDlg.DoModal();               // Создание модального диалога
}
// Cpr1App message handlers
```

Листинг 1.3. Файл MainFrm.h (объявление класса окна фрейма)

```
// MainFrm.h : interface of the CMainFrame class
//
#pragma once
#include "ChildView.h"

class CMainFrame : public CFrameWnd  // Класс фреймового окна
{
public:
   CMainFrame();
protected:
   DECLARE_DYNAMIC(CMainFrame)
// Attributes
public:
// Operations
public:
// Overrides
public:
   virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
   virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
                         AFX_CMDHANDLERINFO* pHandlerInfo);
// Implementation
public:
   virtual ~CMainFrame();
#ifdef _DEBUG
   virtual void AssertValid() const;
   virtual void Dump(CDumpContext& dc) const;
#endif

   CChildView m_wndView;        // Окно представления (находится внутри
                                // клиентской области фреймового окна)

// Generated message map functions
protected:
```

```
   afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
   afx_msg void OnSetFocus(CWnd *pOldWnd);
   DECLARE_MESSAGE_MAP()
};
```

Листинг 1.4. Файл MainFrm.cpp (определение класса окна фрейма)

```
// MainFrm.cpp : implementation of the CMainFrame class
//
#include "stdafx.h"
#include "pr1.h"
#include "MainFrm.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame
IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)  // Карта сообщений CMainFrame
   ON_WM_CREATE()                         // Создание окна
   ON_WM_SETFOCUS()                       // Получение фокуса ввода
END_MESSAGE_MAP()

CMainFrame::CMainFrame()                  // Конструктор
{
   // TODO: add member initialization code here
}
CMainFrame::~CMainFrame()                 // Деструктор
{
}

// Обработка сообщения о создании фреймового окна
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // Вызов обработки сообщения базового класса
```

```
   if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;

   // Создание окна представления в клиентской (внутренней) области
   // фреймового окна
   if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
                         CRect(0, 0, 0, 0), this,
                         AFX_IDW_PANE_FIRST,NULL))
   {
      TRACE0("Failed to create view window\n");
      return -1;
   }
   return 0;
}
// Регистрация класса фреймового окна
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
   // Вызов функции регистрации базового класса
   if( !CFrameWnd::PreCreateWindow(cs) )
      return FALSE;
   // TODO: Modify the Window class or styles here by modifying
   //  the CREATESTRUCT cs

   // Задание стилей класса фреймового окна
   cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
   cs.lpszClass = AfxRegisterWndClass(0);    // Регистрация класса
   return TRUE;
}

// CMainFrame diagnostics
// Две отладочные функции - для отладки и проверки данного объекта
// на допустимость значений хранимых в нем величин
#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
```

```
   CFrameWnd::AssertValid();
}
void CMainFrame::Dump(CDumpContext& dc) const
{
   CFrameWnd::Dump(dc);
}
#endif    //_DEBUG

// CMainFrame message handlers

// Установка фокуса ввода
void CMainFrame::OnSetFocus(CWnd* /*pOldWnd*/)
{
   // forward focus to the view window
   m_wndView.SetFocus();              // Установить фокус ввода
                                      // на окно представления
}

// Распределение командных сообщений
BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra,
                          AFX_CMDHANDLERINFO* pHandlerInfo)
{
   // Если это сообщение обрабатывает окно представления - вернуть
   // управление
   if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
      return TRUE;

   // Иначе - обработка сообщения по умолчанию
   return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```

Листинг 1.5. Файл ChildView.h (объявление класса окна представления)

```
// ChildView.h : interface of the CChildView class
//
#pragma once
```

```
// CChildView window
class CChildView : public CWnd      // Класс окна представления
{
// Construction
public:
   CChildView();
// Attributes
public:
// Operations
public:
// Overrides
   protected:
   virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
// Implementation
public:
   virtual ~CChildView();
   // Generated message map functions
protected:
   afx_msg void OnPaint();
   DECLARE_MESSAGE_MAP()
};
```

Листинг 1.6. Файл ChildView.cpp (определение класса окна представления)

```
// ChildView.cpp : implementation of the CChildView class
//
#include "stdafx.h"
#include "pr1.h"
#include "ChildView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChildView
CChildView::CChildView()            // Конструктор
```

```
{
}
CChildView::~CChildView()            // Деструктор
{
}

BEGIN_MESSAGE_MAP(CChildView, CWnd)  // Карта сообщений CChildView
   ON_WM_PAINT()                     // Перерисовка окна
END_MESSAGE_MAP()

// CChildView message handlers

// Регистрация класса окна
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
   if (!CWnd::PreCreateWindow(cs))
      return FALSE;

   cs.dwExStyle |= WS_EX_CLIENTEDGE;
   cs.style &= ~WS_BORDER;
   cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW| CS_DBLCLKS,
                              ::LoadCursor(NULL, IDC_ARROW),
                              reinterpret_cast<HBRUSH>(COLOR_WINDOW+1),
                              NULL);
   return TRUE;
}

void CChildView::OnPaint()           // Перерисовка окна
{
   CPaintDC dc(this);                // Контекст графического устройства

   // TODO: Add your message handler code here
   // Do not call CWnd::OnPaint() for painting messages
}
```

Листинг 1.7. Файл Resourse.h (идентификаторы приложения)

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by pr1.rc
//
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDR_MAINFRAME                   128
#define IDR_pr1TYPE                     129

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        130
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Листинг 1.8. Файл pr1.rc (ресурсы приложения)

```
// Microsoft Visual C++ generated resource script.
//
#include "resource.h"
#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////
// Generated from the TEXTINCLUDE 2 resource.
#include "afxres.h"
////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////
// Russian resources
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_RUS)
```

```
#ifdef _WIN32
LANGUAGE LANG_RUSSIAN, SUBLANG_DEFAULT
#pragma code_page(1251)
#endif //_WIN32
#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////
// TEXTINCLUDE
1 TEXTINCLUDE
BEGIN
   "resource.h\0"
END

2 TEXTINCLUDE
BEGIN
   "#include ""afxres.h""\r\n"
   "\0"
END

3 TEXTINCLUDE
BEGIN
   "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
   "#define _AFX_NO_OLE_RESOURCES\r\n"
   "#define _AFX_NO_TRACKER_RESOURCES\r\n"
   "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
   "\r\n"
   "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_RUS)\r\n"
   "LANGUAGE 25, 1\r\n"
   "#pragma code_page(1251)\r\n"
   "#include ""res\\pr1.rc2""   // non-Microsoft Visual C++ edited
                               // resources\r\n"
   "#include ""afxres.rc""      // Standard components\r\n"
   "#endif\r\n"
   "\0"
END
```

```
#endif                                  // APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
// Icon
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON                    "res\\pr1.ico"
//////////////////////////////////////////////////////////////////////////////
// Menu
IDR_MAINFRAME MENU
BEGIN
   POPUP "&File"
   BEGIN
      MENUITEM "E&xit",                 ID_APP_EXIT
   END
   POPUP "&Edit"
   BEGIN
      MENUITEM "&Undo\tCtrl+Z",         ID_EDIT_UNDO
      MENUITEM SEPARATOR
      MENUITEM "Cu&t\tCtrl+X",          ID_EDIT_CUT
      MENUITEM "&Copy\tCtrl+C",         ID_EDIT_COPY
      MENUITEM "&Paste\tCtrl+V",        ID_EDIT_PASTE
   END
   POPUP "&Help"
   BEGIN
      MENUITEM "&About pr1...",         ID_APP_ABOUT
   END
END

//////////////////////////////////////////////////////////////////////////////
// Accelerator
IDR_MAINFRAME ACCELERATORS
BEGIN
   "C",            ID_EDIT_COPY,       VIRTKEY, CONTROL,   NOINVERT
   "V",            ID_EDIT_PASTE,      VIRTKEY, CONTROL,   NOINVERT
   VK_BACK,        ID_EDIT_UNDO,       VIRTKEY, ALT,       NOINVERT
```

```
   VK_DELETE,      ID_EDIT_CUT,        VIRTKEY, SHIFT,     NOINVERT
   VK_F6,          ID_NEXT_PANE,       VIRTKEY,            NOINVERT
   VK_F6,          ID_PREV_PANE,       VIRTKEY, SHIFT,     NOINVERT
   VK_INSERT,      ID_EDIT_COPY,       VIRTKEY, CONTROL,   NOINVERT
   VK_INSERT,      ID_EDIT_PASTE,      VIRTKEY, SHIFT,     NOINVERT
   "X",            ID_EDIT_CUT,        VIRTKEY, CONTROL,   NOINVERT
   "Z",            ID_EDIT_UNDO,       VIRTKEY, CONTROL,   NOINVERT
END
////////////////////////////////////////////////////////////////////////
// Dialog
IDD_ABOUTBOX DIALOGEX 0, 0, 235, 55
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP | WS_CAPTION |
WS_SYSMENU
CAPTION "About pr1"
FONT 8, "MS Shell Dlg", 0, 0, 0x1
BEGIN
   ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
   LTEXT           "pr1 Version 1.0",IDC_STATIC,40,10,119,8,SS_NOPREFIX
   LTEXT           "Copyright (C) 2007",IDC_STATIC,40,25,119,8
   DEFPUSHBUTTON   "OK",IDOK,178,7,50,16,WS_GROUP
END
////////////////////////////////////////////////////////////////////////
// Version
VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
```

```
BEGIN
   BLOCK "StringFileInfo"
   BEGIN
      BLOCK "040904e4"
      BEGIN
         VALUE "CompanyName", "TODO: <Company name>"
         VALUE "FileDescription", "TODO: <File description>"
         VALUE "FileVersion", "1.0.0.1"
         VALUE "InternalName", "pr1.exe"
         VALUE "LegalCopyright",
                     "TODO: (c) <Company name>.  All rights reserved."
         VALUE "OriginalFilename", "pr1.exe"
         VALUE "ProductName", "TODO: <Product name>"
         VALUE "ProductVersion", "1.0.0.1"
      END
   END
   BLOCK "VarFileInfo"
   BEGIN
      VALUE "Translation", 0x409, 1252
   END
END
////////////////////////////////////////////////////////////////////////////
// DESIGNINFO
#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO
BEGIN
   IDD_ABOUTBOX, DIALOG
   BEGIN
      LEFTMARGIN, 7
      RIGHTMARGIN, 228
      TOPMARGIN, 7
      BOTTOMMARGIN, 48
   END
END
#endif                    // APSTUDIO_INVOKED
```

```
//////////////////////////////////////////////////////////////////////////
// String Table
STRINGTABLE
BEGIN
   IDP_OLE_INIT_FAILED   "OLE initialization failed.  Make sure that "
                         "the OLE libraries are the correct version."
END

STRINGTABLE
BEGIN
   IDR_MAINFRAME            "pr1"
END

STRINGTABLE
BEGIN
   AFX_IDS_APP_TITLE        "pr1"
   AFX_IDS_IDLEMESSAGE      "Ready"
END

STRINGTABLE
BEGIN
   ID_INDICATOR_EXT      "EXT"
   ID_INDICATOR_CAPS     "CAP"
   ID_INDICATOR_NUM      "NUM"
   ID_INDICATOR_SCRL     "SCRL"
   ID_INDICATOR_OVR      "OVR"
   ID_INDICATOR_REC      "REC"
END

STRINGTABLE
BEGIN
   ID_APP_ABOUT   "Display program information, version number and "
                  "copyright\nAbout"
   ID_APP_EXIT    "Quit the application; prompts to save "
                  "documents\nExit"
END
```

```
STRINGTABLE
BEGIN
   ID_NEXT_PANE    "Switch to the next window pane\nNext Pane"
   ID_PREV_PANE    "Switch back to the previous window "
                   "pane\nPrevious Pane"
END

STRINGTABLE
BEGIN
   ID_WINDOW_SPLIT  "Split the active window into panes\nSplit"
END

STRINGTABLE
BEGIN
   ID_EDIT_CLEAR    "Erase the selection\nErase"
   ID_EDIT_CLEAR_ALL "Erase everything\nErase All"
   ID_EDIT_COPY    "Copy the selection and put it on the Clipboard\nCopy"
   ID_EDIT_CUT     "Cut the selection and put it on the Clipboard\nCut"
   ID_EDIT_FIND    "Find the specified text\nFind"
   ID_EDIT_PASTE   "Insert Clipboard contents\nPaste"
   ID_EDIT_REPEAT   "Repeat the last action\nRepeat"
   ID_EDIT_REPLACE  "Replace specific text with different text\nReplace"
   ID_EDIT_SELECT_ALL "Select the entire document\nSelect All"
   ID_EDIT_UNDO     "Undo the last action\nUndo"
   ID_EDIT_REDO     "Redo the previously undone action\nRedo"
END

STRINGTABLE
BEGIN
   AFX_IDS_SCSIZE       "Change the window size"
   AFX_IDS_SCMOVE       "Change the window position"
   AFX_IDS_SCMINIMIZE    "Reduce the window to an icon"
   AFX_IDS_SCMAXIMIZE    "Enlarge the window to full size"
   AFX_IDS_SCNEXTWINDOW  "Switch to the next document window"
   AFX_IDS_SCPREVWINDOW  "Switch to the previous document window"
```

```
    AFX_IDS_SCCLOSE       "Close the active window and prompts to save "
                          "the documents"
END

STRINGTABLE
BEGIN
    AFX_IDS_SCRESTORE     "Restore the window to normal size"
    AFX_IDS_SCTASKLIST    "Activate Task List"
END

#endif                   // Russian resources
/////////////////////////////////////////////////////////////////////////////
#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
// Generated from the TEXTINCLUDE 3 resource.
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_RUS)
LANGUAGE 25, 1
#pragma code_page(1251)
#include "res\pr1.rc2"      // non-Microsoft Visual C++ edited resources
#include "afxres.rc"        // Standard components
#endif
/////////////////////////////////////////////////////////////////////////////
#endif                      // not APSTUDIO_INVOKED
```

1.6. Описание программы

Работа Windows-приложения начинается с функции `WinMain()`. Но в листингах ее нет, т. к. она скрыта в файлах библиотеки MFC. Фактически, работа программы начинается с создания объекта приложения `Cpr1App theApp`.

Функция `WinMain()` должна выполнять следующие действия:

- инициализация приложения (`Cpr1App::InitInstance()`, листинг 1.2):
  - определение и регистрация оконных классов приложения (`CMainFrame::PreCreateWindow()`, листинг 1.4; `CChildView::PreCreateWindow()`, листинг 1.6);
  - создание главного окна приложения (вызов функции `LoadFrame()`, листинг 1.2);
  - отображение главного окна (вызов функций `ShowWindow()`, `UpdateWindow()`, листинг 1.2);

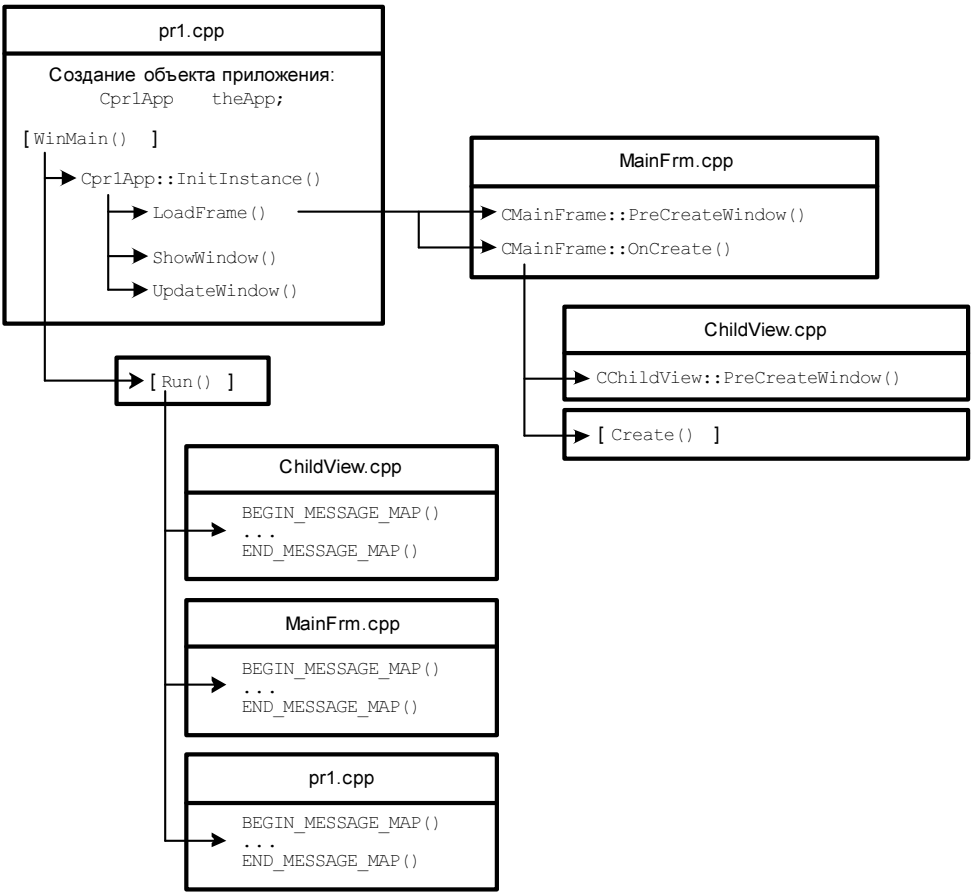


Рис. 1.16. Схема работы приложения

- запуск цикла обработки сообщений (вызов фунции `Run()`, которая тоже скрыта в файлах библиотеки MFC).

Цикл обработки сообщений нужен для получения и обработки сообщений, передаваемых операционной системой Windows. Эти сообщения ставятся в очередь, откуда они потом (по мере готовности программы) передаются в функции, занимающиеся их обработкой. При закрытии главного окна приложения посылается сообщение `WM_DESTROY`. Обработчик этого сообщения (скрытый в файлах библиотеки MFC) посылает в цикл обработки сообщений сообщение `WM_QUIT`, которое является условием прекращения работы цикла, и программа завершает свою работу.

Схема работы Windows-приложения показана на рис. 1.16.

1.6.1. Описание класса приложения *Cpr1App*

Класс `class Cpr1App:public CWinApp` — это класс приложения, который отвечает за управление программой в целом. Он вызывает функции `InitInstance()`, `Run()` и т. д.

Карта сообщений — это специальная структура MFC, которая связывает функции с обрабатываемыми сообщениями и позволяет вызывать для каждого типа сообщения соответствующий обработчик. Карта сообщений объявляется в интерфейсе класса (обычно в файле с расширением h, см. листинг 1.1) с помощью макроса `DECLARE_MESSAGE_MAP()` и определяется в файле реализации класса (обычно в файле с расширением cpp, см. листинг 1.2) с помощью макросов `BEGIN_MESSAGE_MAP()` и `END_MESSAGE_MAP()`, отмечающих соответственно начало и конец карты сообщений.

Синтаксис макроса начала карты сообщения такой:

```
BEGIN_MESSAGE_MAP(Cpr1App, CWinApp)
```

где параметр `Cpr1App` — имя класса, сообщения которого будут обрабатываться данной картой сообщений, `CWinApp` — имя базового класса.

При выборе меню **Help | About pr1** генерируется сообщение, которое затем обрабатывается приложением:

```
ON_COMMAND(ID_APP_ABOUT, &Cpr1App::OnAppAbout)
```

`ON_COMMAND` — стандартный макрос, определяющий функцию, которая будет обрабатывать сообщение `WM_COMMAND`. Параметр `ID_APP_ABOUT` — идентификатор команды (см. рис. 1.11, *д*), `Cpr1App::OnAppAbout` — функция, обрабатывающая данное сообщение. Функция должна быть объявлена в классе

(см. листинг 1.1): `afx_msg void OnAppAbout()` и определена (см. листинг 1.2).

Сначала сообщение посылается объекту окна представления (`CChildView`). Если там нет нужного обработчика, то посылается объекту главного окна программы (`CMainFrame`). Если там нет нужного обработчика — посылается объекту приложения (`Cpr1App`). Если и там нет нужного обработчика, то сообщение обрабатывается стандартным способом.

При создании объекта приложения (`Cpr1App theApp`) автоматически вызывается конструктор базового класса. Конструктор базового класса имеет вид:

```
CWinApp::CWinApp(LPCTSTR lpszAppName = NULL);
```

где `lpszAppName` — указатель на текстовую строку с именем приложения. Если он равен `NULL`, то используется строка из файла ресурсов с идентификатором `IDR_MAINFRAME` — в ней можно указать имя главного окна приложения (см. рис. 1.11, *e*). Этот параметр заносится в член класса `m_pszAppName`.

В функции инициализации приложения (`Cpr1App::InitInstance()`) используется структура для хранения данных о доступных элементах управления:

```
typedef struct tagINITCOMMONCONTROLSEX
{
   DWORD dwSize;                              // Размер структуры в байтах
   DWORD dwICC;                               // Битовый флаг
} INITCOMMONCONTROLSEX, *LPINITCOMMONCONTROLSEX;
```

Параметр `dwICC` задает, какой класс общих элементов управления (common controls) должен быть загружен для использования.

Флаг `dwICC` может принимать следующие значения:

- `ICC_ANIMATE_CLASS`, который включает следующий набор элементов управления:
  - animate (анимация) — просмотр видеоклипов в формате AVI;
- `ICC_BAR_CLASSES`:
  - toolbar (панель инструментов) — меню в виде кнопок с иконками;
  - status bar (окно состояния) — линейка, содержащая информацию о состоянии приложения или подсказки;
  - trackbar (ползунок) — работает по принципу линейки прокрутки;

- ToolTip (подсказка) — небольшой всплывающий прямоугольник со справочной информацией о назначении кнопок панели инструментов;

- ICC_COOL_CLASSES:
  - rebar (перемещаемая панель) — в начале перемещаемой панели находится небольшая полоска, захватив которую мышью можно перемещать панель, а также прикреплять ее к границам окна;
- ICC_DATE_CLASSES:
  - date (дата) — поле или календарь для выбора даты;
  - time picker (таймер) — поле для выбора времени;
- ICC_HOTKEY_CLASS:
  - hot key (горячая клавиша) — комбинация клавиш для быстрого доступа к определенным действиям;
- ICC_LISTVIEW_CLASSES:
  - list-view (список) — может содержать не только текст, но и иконки, связанные с элементами списка;
  - header (дочернее окно над колонками с заголовками каждого столбца, позволяющее менять ширину столбца);
- ICC_PROGRESS_CLASS:
  - progress bar (индикатор) — полоса прогресса, заполняемая со временем;
- ICC_STANDARD_CLASSES (стандартные элементы управления):
  - button (кнопки);
  - edit (поле редактирования);
  - static (простой текст);
  - listbox (список);
  - combobox (комбинированный список) — комбинация окна редактирования и списка;
- ICC_TAB_CLASSES:
  - tab (закладка) — меню в виде закладок в записной книжке, для смены страниц в одной и той же области диалогового окна;
  - ToolTip;

- `ICC_TREEVIEW_CLASSES`:
  - tree-view (окно просмотра деревьев) — специальное окно для просмотра древовидных структур данных;
  - ToolTip;
- `ICC_UPDOWN_CLASS`:
  - up-down (спин) — небольшой элемент управления с возможностью выбора значения с помощью стрелок вверх и вниз;
- `ICC_WIN95_CLASSES`:
  - animate, control, header, hot key, list-view, progress bar, status bar, tab, ToolTip, toolbar, trackbar, tree-view, up-down.

После выбора набора элементов управления загружается динамическая библиотека общих элементов управления:

```
BOOL InitCommonControlsEx(                          // 0 - Ошибка
   LPINITCOMMONCONTROLSEX lpInitCtrls);
```

Параметр `lpInitCtrls` — указатель на структуру с данными об используемых элементах управления.

Основные действия по инициализации приложения выполняются в функции базового класса, которую надо вызвать:

```
virtual BOOL CWinApp::InitInstance();           // 0 - Ошибка
```

Инициализация OLE-библиотеки выполняется с помощью функции:

```
BOOL AFXAPI AfxOleInit();                       // 0 - Ошибка
```

> ***Примечание***
>
> Технология OLE (Object Linking and Embedding — связывание и внедрение объектов) представляет собой форму взаимодействия процессов. В частности, она позволяет одному приложению внедрять или связывать информацию, создаваемую другими приложениями. В OLE имеются два типа приложений: контейнеры (клиенты) и сервер (объект).

Для вывода окна сообщения используется функция:

```
int AfxMessageBox(                  // Код нажатой в окне сообщения кнопки
   LPCTSTR lpszText,                // Выводимый текст
   UINT nType = MB_OK,              // Стиль окна
   UINT nIDHelp = 0);               // Идентификатор справки
```

Параметр `lpszText` может передаваться по-разному:

- с использованием Unicode:
  - `CString s("Сообщение");  AfxMessageBox(s);`
  - `AfxMessageBox((CString) "Сообщение");`
  - `AfxMessageBox(_T("Сообщение"));`
  - `AfxMessageBox(L"Сообщение");`
- без использования Unicode:
  - `CString s("Сообщение");  AfxMessageBox(s);`
  - `AfxMessageBox("Сообщение");`

При использовании параметра `nType` можно комбинировать битовые флаги (с помощью символа `'|'`), но не более одного флага из каждой группы. Битовые флаги делятся на следующие группы:

1. Какие кнопки будут в окне:
   - `MB_ABORTRETRYIGNORE` — **Прервать** (Abort), **Повтор** (Retry), **Пропустить** (Ignore);
   - `MB_OK` — **OK** (по умолчанию);
   - `MB_OKCANCEL` — **OK**, **Отмена** (Cancel);
   - `MB_RETRYCANCEL` — **Повтор** (Retry), **Отмена** (Cancel);
   - `MB_YESNO` — **Да** (Yes), **Нет** (No);
   - `MB_YESNOCANCEL` — **Да** (Yes), **Нет** (No), **Отмена** (Cancel).
2. Вид иконки в окне сообщения:
   - `MB_ICONEXCLAMATION` —  ;
   - `MB_ICONINFORMATION` —  ;
   - `MB_ICONQUESTION` —  ;
   - `MB_ICONSTOP` —  .
3. Кнопка, используемая по умолчанию (указывается при использовании в окне более одной кнопки):
   - `MB_DEFBUTTON1` — первая из списка кнопок (по умолчанию);
   - `MB_DEFBUTTON2` — вторая из списка кнопок;
   - `MB_DEFBUTTON3` — третья из списка кнопок.

Возвращаемое значение функции `AfxMessageBox()` может быть таким:

- `IDABORT` — была нажата кнопка **Прервать** (Abort);
- `IDCANCEL` — была нажата кнопка **Отмена** (Cancel) или клавиша <Esc> (если в окне нет кнопки **Отмена** (Cancel), то нажатие клавиши <Esc> ни к чему не приводит);
- `IDIGNORE` — была нажата кнопка **Пропустить** (Ignore);
- `IDNO` — была нажата кнопка **Нет** (No);
- `IDOK` — была нажата кнопка **OK**;
- `IDRETRY` — была нажата кнопка **Повтор** (Retry);
- `IDYES` — была нажата кнопка **Да** (Yes);

Параметр `nIDHelp` содержит идентификатор для перехода к соответствующей теме справки при нажатии клавиши <F1>.

Еще один вариант функции вывода окна сообщения, работающей с таблицей строк ресурсов приложения:

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

где `nIDPrompt` — идентификатор строки ресурса выводимого текста; `nIDHelp` — если задано значение по умолчанию (`-1`), то идентификатор `nIDPrompt` задает тему справки.

Для добавления строки ресурса надо щелкнуть левой кнопкой мыши по пустому полю внизу таблицы строк (рис. 1.17, *а*) и ввести новые данные (рис. 1.17, *б*).

Строку ресурса можно использовать несколькими способами:

1. Создать простую строку и вывести ее с помощью параметра `nIDPrompt`. Данные новой строки приведены в табл. 1.1.

Таблица 1.1. *Добавление новой строки в таблицу строк*

ID	Value	Caption
IDS_STRING101	101	Новая строка

Вызов функции такой:

```
AfxMessageBox(IDS_STRING101);
```

В окне сообщения будет текст "Новая строка".

2. Создать строку ресурса с одним форматом (`%1`). Данные новой строки приведены в табл. 1.2.

Таблица 1.2. *Добавление новой строки с одним форматом в таблицу строк*

ID	Value	Caption
IDS_STRING102	102	Новая %1 строка

Вывод сообщения может быть таким:

```
CString sinp("ПЕРВАЯ");
CString s;
AfxFormatString1(s, IDS_STRING102, sinp);
AfxMessageBox(s);
```

В окне сообщения будет текст "Новая ПЕРВАЯ строка".

3. Создать строку ресурса с форматами (`%1` и `%2`). Данные новой строки приведены в табл. 1.3.

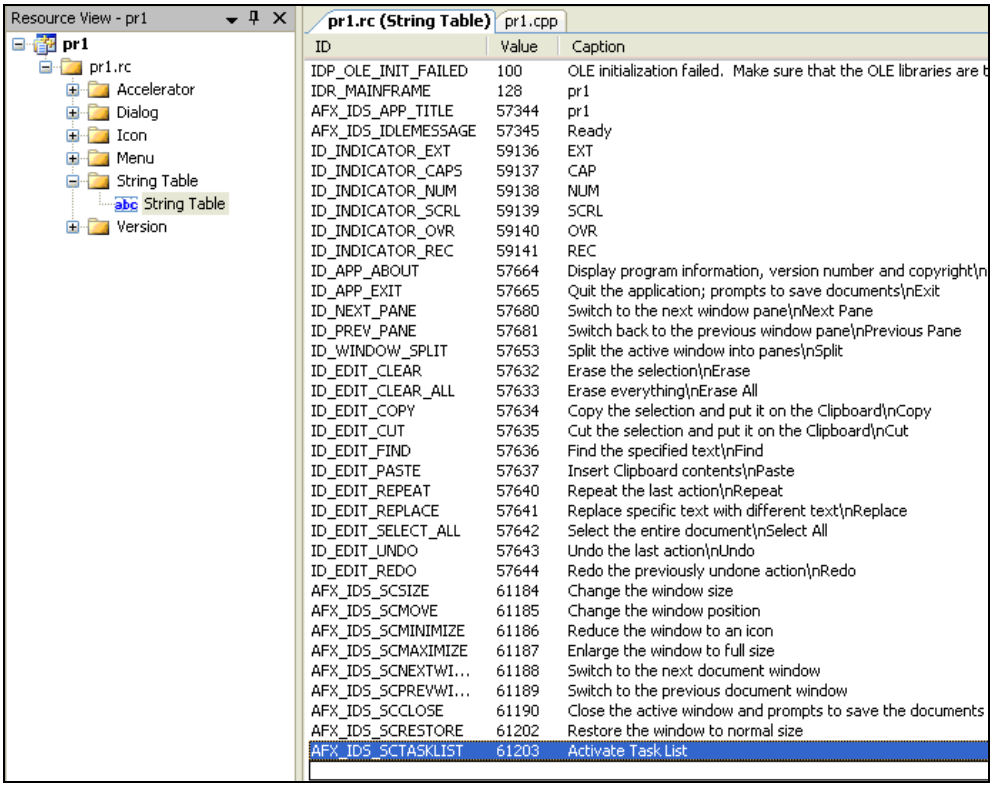
Таблица 1.3. *Добавление новой строки с двумя форматами в таблицу строк*

ID	Value	Caption
IDS_STRING103	103	Новая %1 строка с %2 аргументами

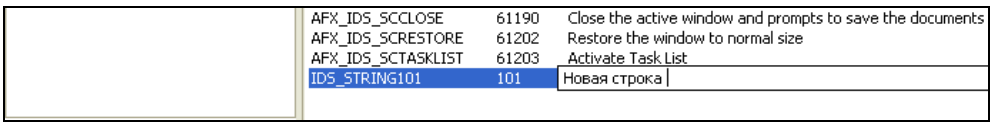
Вывод сообщения такой:

```
CString sinp1("ПЕРВАЯ");
CString sinp2("ДВУМЯ");
CString s;
AfxFormatString2(s, IDS_STRING103, sinp1, sinp2);
AfxMessageBox(s);
```

В окне сообщения будет текст "Новая ПЕРВАЯ строка с ДВУМЯ аргументами".



а



б

Рис. 1.17. Окно ресурсов с таблицей строк (*а*) и добавление новой строки для вывода в окне сообщения (*б*)

Примечание

Для удобства быстрого написания названий функций, не относящихся к классу, можно воспользоваться оператором разрешения области видимости (`::`). Сразу после того, как вы его набрали, появляется контекстная подсказка со списком функций, в котором можно выбрать нужную. Для быстрого перемещения по длинному списку можно набрать первые символы названия функции, и список автоматически перескочит на первую строчку с

таким названием. Получить контекстную подсказку можно также с помощью комбинации клавиш <Ctrl>+<Пробел>. Например:

```
::AfxMessageBox("Сообщение");
```

В предыдущих примерах кода использовались функции форматирования строк с форматом:

- ```
  void AfxFormatString1(
    CString& rString,
    UINT nIDS,
    LPCTSTR lpsz1);
  ```

  Эта функция загружает строку из ресурсов, определяемую идентификатором `nIDS` в объект `rString` с одновременной заменой шаблона `%1` на строку, указанную в `lpsz1`;

- ```
  void AfxFormatString2(
    CString& rString,
    UINT nIDS,
    LPCTSTR lpsz1
    LPCTSTR lpsz2);
  ```

  Аналогично `AfxFormatString1()`, но замена выполняется для шаблона `%1 %2` на строки `lpsz1` и `lpsz2` соответственно.

Для `AfxMessageBox()` в качестве заголовка окна сообщения используется имя программы. Если надо задать свой заголовок окна сообщения, то можно воспользоваться функцией:

```
int MessageBox(              // Код нажатой кнопки или 0 - если ошибка
                             // при создании окна
   HWND hWnd,                // Дескриптор родителя окна-сообщения
   LPCTSTR lpText,           // Выводимый текст
   LPCTSTR lpCaption,        // Строка - заголовок окна
   UINT nType = MB_OK);      // Стиль окна
```

Если `hWnd` задать `NULL`, то родителя у окна сообщения не будет. Если надо сделать родителем главное окно, то можно получить его дескриптор следующим образом:

```
theApp.m_pMainWnd->operator HWND()
```

или:

```
AfxGetMainWnd()->operator HWND()
```

Если параметр `lpCaption` задать `NULL`, то у сообщения будет стандартный заголовок окна — "Ошибка".

Создание главного окна приложения выполняется с помощью функции:

```
virtual BOOL CFrameWnd::LoadFrame(  // false - ошибка создании
   UINT nIDResource,                // Идентификатор ресурса данного окна
   DWORD dwDefaultStyle =           // Стиль окна
      WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE,
   CWnd* pParentWnd = NULL,          // Указатель на класс родительского
                                     // окна
   CCreateContext* pContext = NULL); // Указатель на объект структуры
                                     // CCreateContext
```

Функция `LoadFrame()` загружает главное окно приложения и связанные с ним ресурсы. В параметре `nIDResource` определены меню, таблица назначения клавиш, значки и строковый ресурс заголовка окна. Параметр `pContext` задает объекты классов, связанные с данным классом окна, включая все объекты класса представления.

Стили окна (параметр `dwDefaultStyle`) могут быть следующими:

- `WS_BORDER` — окно имеет рамку;
- `WS_CAPTION` — окно имеет титульную полосу, в которой отображается заголовок;
- `WS_CHILD` — окно является дочерним (порождено другим окном);
- `WS_CHILDWINDOW` — то же, что и `WS_CHILD`;
- `WS_CLIPCHILDREN` — в процессе рисования родительского окна не разрешается рисование в порожденных окнах;
- `WS_CLIPSIBLINGS` — перерисовка одного из порожденных окон не влечет за собой перерисовку других. Используется только вместе со стилем `WS_CHILD`;
- `WS_DISABLED` — неактивное окно, не получающее фокус ввода;
- `WS_DLGFRAME` — рисуется окно с рамкой, как у обычного диалога; заголовок отсутствует;
- `WS_GROUP` — определяет первый элемент в группе элементов управления;
- `WS_HSCROLL` — окно имеет горизонтальную линейку прокрутки;
- `WS_MAXIMIZE` — отображение окна в полноэкранном виде;

- WS_MAXIMIZEBOX — окно имеет кнопку максимизации в правом верхнем углу;
- WS_MINIMIZE — окно отображается свернутым до иконки на панели задач;
- WS_MINIMIZEBOX — окно имеет кнопку минимизации в правом верхнем углу;
- WS_OVERLAPPED — окно с рамкой и заголовком;
- WS_OVERLAPPEDWINDOW — комбинация стилей WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX и WS_MAXIMIZEBOX;
- WS_POPUP — всплывающее окно;
- WS_POPUPWINDOW — создание всплывающего окна с комбинацией стилей WS_BORDER, WS_POPUP и WS_SYSMENU;
- WS_SIZEBOX — окно имеет толстую рамку, используемую для изменения размеров окна;
- WS_SYSMENU — окно имеет кнопку системного меню в левом верхнем углу;
- WS_TABSTOP — используется для элементов управления, которые можно выбирать клавишей <Tab>;
- WS_THICKFRAME — окно имеет толстую рамку, используемую для изменения размеров окна;
- WS_VISIBLE — окно изначально видимо;
- WS_VSCROLL — окно имеет вертикальную линейку прокрутки.

Дополнительные стили, которые могут комбинироваться с перечисленными основными стилями с помощью операции '|':

- FWS_ADDTOTITLE — в конец заголовка окна добавляет имя отображаемого в нем документа;
- FWS_PREFIXTITLE — перед заголовком окна помещается имя документа;
- FWS_SNAPTOBARS — позволяет подогнать размеры окна под размеры панели элементов управления.

Для задания *всех* параметров создания окна следует вместо LoadFrame() использовать функцию Create():

```
virtual BOOL CFrameWnd::Create(  // false - ошибка создания
   LPCTSTR lpszClassName,        // Указатель на строку - имя класса окна
   LPCTSTR lpszWindowName,       // Указатель на строку - заголовок окна
```

```
   DWORD dwStyle = WS_OVERLAPPEDWINDOW, // Стиль окна
   const RECT& rect = rectDefault,      // Ссылка на структуру,
                                // содержащую размеры и координаты окна
                                // относительно его родителя
   CWnd* pParentWnd = NULL,     // Указатель на класс родительского окна
   LPCTSTR lpszMenuName = NULL, // Адрес строки, содержащей
                                // имя ресурса меню
   DWORD dwExStyle = 0,         // Атрибуты расширенного стиля
   CCreateContext* pContext = NULL);   // Указатель на структуру
                                       // CCreateContext
```

Если `lpszClassName` задать `NULL`, то будут использоваться атрибуты предопределенного класса `CFrameWnd`.

Если меню задано числовым идентификатором, то для получения строки `lpszMenuName` надо использовать макрос `MAKEINTRESOURCE`:

```
   LPCTSTR MAKEINTRESOURCE(WORD wInteger);
```

Структура с размерами и координатами окна определена как:

```
typedef struct tagRECT
{
   LONG left;     // X координата верхнего левого угла
   LONG top;      // Y координата верхнего левого угла
   LONG right;    // X координата нижнего правого угла
   LONG bottom;   // Y координата нижнего правого угла
} RECT;
```

После создания окна задается вид его отображения, и окно перерисовывается (в данном случае отображается).

Вид отображения окна выполняется с помощью функции:

```
BOOL CWnd::ShowWindow(     // true - если окно до этого отображалось,
                           // false - если оно до этого было скрыто
   int nCmdShow);          // Режим отображения
```

Режимы отображения (`nCmdShow`) могут быть заданы такие:

- `SW_HIDE` — скрывается текущее окно и активизируется другое;
- `SW_MINIMIZE` — активное окно свертывается в иконку на панели задач;
- `SW_RESTORE` — окно активизируется и отображается (если окно было свернуто, то восстанавливаются его прежние размеры);

- SW_SHOW — окно активизируется и отображается с текущими размерами и положением;
- SW_SHOWMAXIMIZED — окно активизируется и отображается развернутым на весь экран;
- SW_SHOWMINIMIZED — окно активизируется и отображается свернутым в иконку на панели задач;
- SW_SHOWMINNOACTIVE — окно отображается свернутым в иконку на панели задач (при этом окно не меняет свою активность);
- SW_SHOWNA — отображает окно в его текущем состоянии (при этом окно не меняет свою активность);
- SW_SHOWNOACTIVATE — аналогично SW_SHOWNA;
- SW_SHOWNORMAL — аналогично SW_RESTORE.

Перерисовка окна выполняется функцией:

```
void CWnd::UpdateWindow();
```

Класс диалогового окна, которое появляется при выборе пункта меню **Help | About pr1**:

```
class CAboutDlg : public CDialog
```

Идентификатор ресурса этого диалогового окна (см. рис. 1.11, *б*):

```
enum { IDD = IDD_ABOUTBOX };
```

Функция DoDataExchange() вызывается для обмена данными (и проверки корректности данных), передаваемых между объектами классов диалогового окна и включенных в него объектов классов элементов управления (переключателей, полей редактирования, списков и т. п.):

```
virtual void CWnd::DoDataExchange(
   CDataExchange* pDX);          // Указатель на изменяемый объект
```

Создание модального диалогового окна (приложение не будет ни на что реагировать, пока это окно не будет закрыто) выполняется с помощью функции:

```
virtual INT_PTR CDialog::DoModal();
```

1.6.2. Описание класса окна фрейма *MainFrm*

Класс фрейма (главного окна приложения) отвечает только за рамочное окно программы (т. е. может иметь рамку, строку заголовка, строку меню, систем-

ное меню, кнопки (минимизации, максимизации и закрытия окна), панель инструментов, строку статуса):

```
class CMainFrame : public CFrameWnd;
```

Этот класс *не может* управлять клиентской (внутренней) областью главного окна.

Иерархия классов: `CObject -> CCmdTarget -> CWnd -> CFrameWnd`.

Класс `CObject` не поддерживает множественное наследование. Создаваемые на его основе классы могут иметь в качестве базового класса только `CObject` (и он должен быть самым старшим в иерархии). Чтобы получить доступ ко всем возможностям класса `CObject` при объявлении и реализации производных от него классов, необходимо использовать макросы `DECLARE_DYNAMIC` (в объявлении класса, листинг 1.9) и `IMPLEMENT_DYNAMIC` (в файле определения класса, листинг 1.10).

Листинг 1.9. Фрагмент файла MainFrm.h, в котором используется макрос DECLARE_DYNAMIC

```
class CMainFrame : public CFrameWnd
{
// ...
protected:
    // CMainFrame - имя класса фреймового окна
    DECLARE_DYNAMIC(CMainFrame)
};
```

Листинг 1.10. Фрагмент файла MainFrm.cpp, в котором используется макрос IMPLEMENT_DYNAMIC

```
// ...
   // CMainFrame — имя класса фреймового окна
   // CFrameWnd — имя базового класса
   IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)
```

В карте сообщений фреймового окна задана обработка двух сообщений: `WM_CREATE` и `WM_SETFOCUS`. Сообщение `WM_CREATE` генерируется при создании

окна (при вызове функции `Create()` или `LoadFrame()`), до того как окно станет видимым. Обработка сообщения о создании окна происходит в функции:

```
afx_msg int CWnd::OnCreate(
   LPCREATESTRUCT lpCreateStruct);
```

Если функция возвращает 0, то процесс создания объекта `CWnd` продолжается. Если возвращает `-1` — происходит уничтожение окна.

Параметр `lpCreateStruct` — указатель на структуру, которая содержит информацию о создаваемом объекте окна:

```
typedef struct tagCREATESTRUCT
{
   LPVOID lpCreateParams;   // Данные для создания окна
   HANDLE hInstance;        // Дескриптор приложения
   HMENU hMenu;             // Дескриптор меню
   HWND hwndParent;         // Дескриптор родительского окна
   int cy;                  // Высота окна
   int cx;                  // Ширина окна
   int y;                   // Y координата левого верхнего угла
   int x;                   // X координата левого верхнего угла
   LONG style;              // Стиль окна
   LPCSTR lpszName;         // Указатель на строку - заголовок окна
   LPCSTR lpszClass;        // Указатель на строку - имя класса окна
   DWORD dwExStyle;         // Расширенный стиль окна
} CREATESTRUCT;
```

Далее создается окно представления (дочернее) в клиентской (внутренней) области фреймового окна (листинг 1.11).

Листинг 1.11. Фрагмент файла MainFrm.cpp, в котором создается окно представления

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
// ...
   m_wndView.Create(
      NULL,                    // Предопределенный тип
      NULL,                    // Без заголовка
```

```
      AFX_WS_DEFAULT_VIEW,     // Стиль по умолчанию
      CRect(0, 0, 0, 0),       // Размеры - вся клиентская область
      this,                    // Родитель – фреймовое окно (CMainFrame)
      AFX_IDW_PANE_FIRST,      // Это первое дочернее окно
      NULL);                   // Без контекстных объектов
}
```

Здесь используется функция создания окна, которая выглядит следующим образом:

```
virtual BOOL CWnd::Create(      // false – ошибка создания
   LPCTSTR lpszClassName,       // Указывает на строку – имя класса окна
   LPCTSTR lpszWindowName,      // Указывает на строку – заголовок окна
   DWORD dwStyle,               // Стиль окна
   const RECT& rect,            // Координаты окна
   CWnd* pParentWnd,            // Указатель на родительское окно
   UINT nID,                    // Идентификатор дочернего окна
   CCreateContext* pContext = NULL);  // Указатель на структуру
                                      // CCreateContext
```

В качестве параметра `dwStyle` передается стиль `AFX_WS_DEFAULT_VIEW` — это комбинация стилей `WS_CHILD | WS_VISIBLE | WS_BORDER`.

В параметре `nID` передается значение `AFX_IDW_PANE_FIRST` — идентификатор первого дочернего окна.

`AFX_WS_DEFAULT_VIEW` и `AFX_IDW_PANE_FIRST` определены в стандартном файле afxres.h, приведенном в листинге 1.12.

Листинг 1.12. Фрагмент файла afxres.h

```
// ...
// parts of Main Frame
#define AFX_IDW_PANE_FIRST         0xE900     // first pane (256 max)
#define AFX_IDW_PANE_LAST          0xE9ff
// common style for form views
#define AFX_WS_DEFAULT_VIEW  (WS_CHILD | WS_VISIBLE | WS_BORDER)
```

Координаты создаваемого окна (параметр `rect`) задаются с помощью класса `CRect`. `CRect` — класс, определяющий размеры прямоугольной области (в

данном случае размеры окна; если задать все 0, то будут использованы размеры по умолчанию):

```
class CRect:public tagRECT
```

Конструктор класса определен как:

```
CRect::CRect(
   int l, int t,      // X, Y координата верхнего левого угла
   int r, int b);     // X, Y координата нижнего правого угла
CRect::CRect(const RECT &srcRect);
CRect::CRect(LPCRECT lpSrcRect);
CRect::CRect(POINT point, SIZE size);
```

Использовать `CRect` можно несколькими способами, например:

1.
```
CRect rect(0, 0, 100, 50);
```

2.
```
RECT Rect;
   Rect.left = 0;
   Rect.top = 0;
   Rect.right = 100;
   Rect.bottom = 50;
   CRect rect(Rect);
```

3.
```
RECT Rect;
   Rect.left = 0;
   Rect.top = 0;
   Rect.right = 100;
   Rect.bottom = 50;
   CRect rect(&Rect);
```

4.
```
CPoint pt(0, 0);
   CSize sz(100, 50);
   CRect rect(pt, sz);
```

Класс, задающий координаты точки, определен как:

```
class CPoint:public tagPOINT
typedef struct tagPOINT
{
   LONG x;                 // X, Y координаты точки
   LONG y;
} POINT;
```

Класс, задающий размеры, определен следующим образом:

```
class CSize:public tagSIZE
typedef struct tagSIZE
{
   LONG cx;          // Ширина
   LONG cy;          // Высота
} SIZE, *PSIZE;
```

Если окно представления создать не удалось, то с помощью макроса `TRACE0` на экран выводится строка сообщения.

При создании окна используется указатель на строку с именем класса окна `LPCTSTR lpszClassName`. Если в функции `Create()` задать `lpszClassName = NULL`, то будут использованы предопределенные настройки окна класса `CWnd`.

Если надо изменить параметры оконного класса, то необходимо переопределить функцию `PreCreateWindow()` (в ней задаются стили класса окна и происходит его регистрация. Эта функция вызывается перед `OnCreate()`):

```
virtual BOOL CWnd::PreCreateWindow(
   CREATESTRUCT &cs);
```

Параметр `cs` — ссылка на структуру `CREATESTRUCT`, которая определяет параметры инициализации.

Использовать `PreCreateWindow()` можно, например, так:

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
   // Задание стиля окна
   cs.style = WS_OVERLAPPED | WS_SYSMENU | WS_BORDER;

   // Размер окна = 1/3 размера экрана, и окно будет расположено в центре
   cs.cy = ::GetSystemMetrics(SM_CYSCREEN)/3;
   cs.cx = ::GetSystemMetrics(SM_CXSCREEN)/3;
   cs.y = ((cs.cy * 3) - cs.cy)/2;
   cs.x = ((cs.cx * 3) - cs.cx)/2;
   // Передача установленных значений для регистрации базовому классу
   return CFrameWnd::PreCreateWindow(cs);
}
```

Можно также задать, добавить или исключить опции (стили):

```
cs.style = WS_OVERLAPPED | WS_SYSMENU | WS_BORDER;      // Задать
cs.lpszName = L"Заголовок";
cs.dwExStyle |= WS_EX_CLIENTEDGE;                       // Добавить
cs.dwExStyle &= ~WS_EX_CLIENTEDGE;                      // Исключить
```

Расширенные стили окна:

- `WS_EX_ACCEPTFILES` — окно поддерживает "перетаскивание файлов" (drag-and-drop);
- `WS_EX_CLIENTEDGE` — придает трехмерный эффект границе окна (клиентская область как бы вдавлена в окно);
- `WS_EX_CONTEXTHELP` — включает знак вопроса (для быстрой контекстной справки) в заголовок окна;
- `WS_EX_CONTROLPARENT` — позволяет перемещаться по дочерним окнам с помощью клавиши <Tab>;
- `WS_EX_DLGMODALFRAME` — окно имеет удвоенную границу;
- `WS_EX_LEFT` — для окна устанавливается свойство выравнивания влево (по умолчанию). Это свойство зависит от класса окна, например, для поля редактирования это выравнивание текста в окне;
- `WS_EX_LEFTSCROLLBAR` — если есть полоса прокрутки, то она находится слева в окне;
- `WS_EX_MDICHILD` — дочернее окно MDI (Multiple Document Interface, многодокументный интерфейс);
- `WS_EX_RIGHTSCROLLBAR` — если есть полоса прокрутки, то она находится справа от окна (по умолчанию);
- `WS_EX_RTLREADING` — определяет порядок ввода и чтения текста справа налево;
- `WS_EX_STATICEDGE` — окно имеет границы с трехмерным стилем для статических элементов управления;
- `WS_EX_TOOLWINDOW` — стиль окна, которое используется для создания плавающей панели инструментов;
- `WS_EX_TOPMOST` — окно будет располагаться поверх всех окон. Для установки или снятия этого бита можно использовать функцию `SetWindowPos()`;

- `WS_EX_TRANSPARENT` — окно является прозрачным;
- `WS_EX_WINDOWEDGE` — окно имеет приподнятую границу для создания трехмерного эффекта.

Далее идет регистрация класса окна и получение имени зарегистрированного класса окна:

```
LPCTSTR AFXAPI AfxRegisterWndClass( // Указатель на строку с именем
                                    // класса
   UINT nClassStyle,                // Стиль класса окна (комбинация с | )
   HCURSOR hCursor = 0,             // Дескриптор курсора
   HBRUSH hbrBackground = 0,        // Дескриптор фоновой кисти (цвет фона
                                    // окна)
   HICON hIcon = 0);                // Дескриптор иконки
```

Стили класса окна могут быть следующими:

- `CS_CLASSDC` — будет создан единый контекст отображения, для использования всеми окнами, созданными на базе этого класса;
- `CS_DBLCLKS` — функция окна будет получать сообщения при двойном щелчке мыши. При этом в функцию окна посылаются сообщения `WM_LBUTTONDBLCLK` и `WM_LBUTTONDBLCLK`. Если этот стиль не будет задан, функция окна получит только идущие парами сообщения об одинарном нажатии кнопки мыши (как бы быстро не щелкать мышью);
- `CS_GLOBALCLASS` — данный класс является глобальным и доступным другим приложениям (другие приложения могут создавать окна на базе этого класса);
- `CS_HREDRAW` — будет перерисовываться внутренняя область окна при изменении его горизонтального размера;
- `CS_NOCLOSE` — будет запрещен выбор закрытия окна;
- `CS_OWNDC` — для каждого окна, созданного на базе данного класса, будет создаваться отдельный контекст отображения;
- `CS_PARENTDC` — окно будет пользоваться родительским контекстом отображения (а не своим собственным);
- `CS_SAVEBITS` — для данного окна система Windows должна сохранять изображение в виде битового образа (bmp). Если такое окно будет перекрыто другим окном, то после уничтожения перекрывшего окна изобра-

жение первого будет восстановлено Windows на основании ранее сохраненного образа;

- CS_VREDRAW — внутренняя область окна будет перерисовываться при изменении его вертикального размера.

Выбор курсора осуществляется с помощью функции:

```
HCURSOR LoadCursor(              // Дескриптор курсора
   HINSTANCE hInstance,          // Дескриптор приложения
   LPCTSTR lpCursorName);        // Указатель на строку с именем ресурса
                                 // курсора
```

Если hInstance = NULL, то можно использовать стандартные виды курсора:

- IDC_ARROW — стрелка;
- IDC_CROSS — перекрестье;
- IDC_HAND — рука;
- IDC_HELP — стрелка с вопросом;
- IDC_IBEAM — вертикальная черта (используемая в редакторах);
- IDC_UPARROW — вертикальная стрелка;
- IDC_WAIT — песочные часы.

В качестве цвета фона окна можно использовать стандартные цвета (при их использовании надо обязательно прибавлять 1, т. к. они определены по порядку, начиная с нуля). Для стандартных настроек Windows:

- COLOR_ACTIVEBORDER — цвет границы активного окна (серый);
- COLOR_ACTIVECAPTION — цвет заголовка активного окна (синий);
- COLOR_APPWORKSPACE — цвет фона MDI-среды приложения (темно-серый);
- COLOR_BACKGROUND — цвет рабочего стола (темно-синий);
- COLOR_BTNFACE — цвет кнопок (светло-серый);
- COLOR_BTNSHADOW — цвет кнопок при наведении на них курсором (серый);
- COLOR_BTNTEXT — цвет текста на кнопках (черный);
- COLOR_CAPTIONTEXT — цвет текста в заголовке (белый);
- COLOR_GRAYTEXT — цвет недоступного текста (серый);

- `COLOR_HIGHLIGHT` — цвет выбранного элемента управления (темно-синий);
- `COLOR_HIGHLIGHTTEXT` — цвет текста выбранного элемента управления (белый);
- `COLOR_INACTIVEBORDER` — цвет границы неактивного окна (серый);
- `COLOR_INACTIVECAPTION` — цвет заголовка неактивного окна (голубой);
- `COLOR_MENU` — цвет фона меню (серый);
- `COLOR_MENUTEXT` — цвет текста в меню (черный);
- `COLOR_SCROLLBAR` — цвет линейки прокрутки (серый);
- `COLOR_WINDOW` — цвет фона окна (белый);
- `COLOR_WINDOWFRAME` — цвет рамки окна (черный);
- `COLOR_WINDOWTEXT` — цвет текста в окне (черный).

Другой вариант задания цвета фона — использовать функцию `GetStockObject()` и задать нужный цвет в качестве аргумента:

- `BLACK_BRUSH` — черный;
- `DKGRAY_BRUSH` — темно-серый;
- `HOLLOW_BRUSH` — прозрачный;
- `LTGRAY_BRUSH` — светло-серый;
- `WHITE_BRUSH` — белый.

Функция получения дескрипторов стандартных CDI (Graphics Device Interface, Интерфейс графических устройств) ресурсов (кисти, перья и т. п.):

```
HGDIOBJ GetStockObject(
   int fnObject);
```

Функция возвращает дескриптор объекта, задаваемого аргументом `fnObject`.

Выбор иконки выполняется с помощью функции:

```
HICON LoadIcon(                 // Дескриптор курсора
   HINSTANCE hInstance,         // Дескриптор приложения
   LPCTSTR lpIconName);         // Указатель на строку с именем ресурса
                                // иконки
```

Если `hInstance = NULL`, то можно использовать стандартные виды иконок:

- `IDI_APPLICATION` — стандартная иконка  ;

- `IDI_ASTERISK` — иконка "Информация"  ;
- `IDI_ERROR` — иконка "Ошибка"  ;
- `IDI_EXCLAMATION` — иконка "Предупреждение"  ;
- `IDI_HAND` — аналогично `IDI_ERROR`;
- `IDI_INFORMATION` — аналогично `IDI_ASTERISK`;
- `IDI_QUESTION` — иконка "`Вопрос`"  ;
- `IDI_WARNING` — аналогично `IDI_EXCLAMATION`;
- `IDI_WINLOGO` — аналогично `IDI_APPLICATION`.

Использовать функцию `AfxRegisterWndClass()` можно, например, так:

```
cs.lpszClass = AfxRegisterWndClass(CS_VREDRAW | CS_HREDRAW,
          LoadCursor(NULL,IDC_ARROW),(HBRUSH)GetStockObject(WHITE_BRUSH),
          LoadIcon(NULL, IDI_APPLICATION));
```

или:

```
cs.lpszClass = AfxRegisterWndClass(0,0,0,LoadIcon(NULL,IDI_APPLICATION));
```

Если стиль класса окна задать `nClassStyle = 0`, то по умолчанию используется следующая комбинация:

- `nClassStyle = CS_DBLCLKS`;
- `hCursor = IDC_ARROW`;
- `hbrBackground = HOLLOW_BRUSH` — нулевая кисть (прозрачное окно, "под которым" видно содержимое других окон, открытых до момента создания данного окна);
- `hIcon = IDI_APPLICATION`.

Для фреймового окна менять цвет и курсор нецелесообразно, т. к. оно будет перекрываться окном представления.

После определения функций создания и регистрации фреймового окна в листинге 1.4 описываются две отладочные функции `AssertValid()` и `Dump()`.

Функция `AssertValid()` проверяет внутреннее состояние объекта на истинность. Отрицательный результат проверки позволяет завершить программу с выдачей сообщения о файле и номере строки, где произошла ошибка. При работе со своим классом следует переопределить эту функцию (причем первой должна вызываться функция базового класса):

```
virtual void CObject::AssertValid() const;
```

Функция `Dump()` сохраняет содержимое объекта в специальном объекте `dc` класса `CDumpContext`, позволяющем послать информацию в окно отладки.

```
virtual void CObject::Dump(CDumpContext &dc) const;
```

Когда окно получает фокус ввода (например, при создании окна), то окну посылается сообщение `WM_SETFOCUS`. Его параметром является дескриптор окна, потерявшего фокус. В ответ на это сообщение вызывается функция `OnSetFocus()`, которой передается указатель на объект `CWnd`, потерявший фокус ввода. Функция `OnSetFocus()` вызывается после `OnCreate()`:

```
afx_msg void CWnd::OnSetFocus(CWnd* pOldWnd);
```

С помощью функции `SetFocus()` фокус ввода устанавливается на окно представления (`m_wndView`):

```
CWnd* CWnd::SetFocus();
```

Далее в листинге 1.4 идет функция `OnCmdMsg()`, которая устанавливает маршрут и распределяет командные сообщения по другим объектам. Это может быть сообщение, которое обрабатывает класс окна представления (`m_wndView.OnCmdMsg()`), или сообщение, которое обрабатывается по умолчанию (`CFrameWnd::OnCmdMsg()`):

```
virtual BOOL CCmdTarget::OnCmdMsg(
   UINT nID,                          // Идентификатор команды
   int nCode,                         // Код извещения
   void* pExtra,                      // Дополнительные параметры
   AFX_CMDHANDLERINFO* pHandlerInfo); // Указатель на структуру
                                      // AFX_CMDHANDLERINFO
```

Обычно параметр `pHandlerInfo` равен `NULL`. Если он не нулевой, то функция `OnCmdMsg()` сама заполняет поля структуры `AFX_CMDHANDLERINFO`. Если функция `OnCmdMsg()` нашла объект, который обрабатывает данное сообщение, то она возвращает `true` (иначе — `false`).

1.6.3. Описание класса окна представления *ChildView*

Класс окна представления отвечает за отображение данных программы на экране, за обработку информации вводимой пользователем и т. д.:

```
class CChildView:public CWnd;
```

Этот класс управляет окном, которое накладывается поверх клиентской (внутренней) области фреймового окна. Окно представления является дочерним от главного фреймового окна и всегда располагается поверх его клиентской области.

Регистрация класса окна (`PreCreateWindow()`) происходит аналогично регистрации фреймового окна. А вот функция создания окна (`Create()`) скрыта в файлах библиотеки MFC.

При перерисовке окна (если окно создали, перекрыли другим окном и т. п.) для обработки сообщения `WM_PAINT` вызывается функция `OnPaint()`:

```
afx_msg void CWnd::OnPaint();
```

Внутри этой функции происходит работа с графическими (GDI) объектами (рисование, вывод текста и т. п.). Чтобы работать с графическими объектами, надо получить контекст графического устройства с помощью класса `CPaintDC` (в данном случае это само окно представления `CChildView`):

```
CPaintDC dc(this);
```

Конструктор класса `CPaintDC` определен так:

```
explicit CPaintDC::CPaintDC(CWnd* pWnd);   // Указатель на окно,
                                           // чей контекст надо получить
```

Графические объекты могут быть определены с помощью следующих классов:

- `CPen` — карандаш, который используется для задания параметров рисования линий: толщина, цвет, стиль (пунктир и т. п.). Дескриптор — `HPEN`;
- `CBrush` — кисть, которая применяется для задания параметров заливки (цвет, стиль) замкнутых контуров. Дескриптор — `HBRUSH`;
- `CFont` — шрифт, задающий параметры вывода текста (имя шрифта, размер символов и т. п.). Дескриптор — `HFONT`;
- `CBitmap` — битовый массив. Это прямоугольный массив точек, формирующий растровое изображение. Дескриптор — `HBITMAP`;
- `CPalette` — палитра, которая используется для осуществления интерфейса между приложением и цветным устройством вывода (дисплей). Содержит список цветов, необходимых приложению. Дескриптор — `HPALETTE`;
- `CRgn` — регион. Это область окна, которая может быть ограничена прямоугольником, многоугольником, эллипсом и т. п. Используется для выполнения операции заполнения, заливки, инверсии и т. д., а также для определения местоположения курсора. Дескриптор — `HRGN`.

1.7. Изменение интерфейса приложения, созданного мастером

1.7.1. Изменения в тексте программы

Для того чтобы увидеть разницу между фреймовым окном (`Frame`) и окном представления (`View`), сделаем окно представления меньше клиентской области фреймового окна (укажем размеры 0, 0, 150, 100). Зададим им разные цвета (`Frame` — серый, `View` — голубой) и типы курсоров (`Frame` — песочные часы, `View` — рука). В фреймовом окне изменим заголовок ("Новый заголовок окна") и иконку приложения ("Ошибка"). При создании окна `View` надо в функции `Create()` вместо `AFX_IDW_PANE_FIRST` (параметр `nID`) задать какой-нибудь несуществующий идентификатор (например, 4444444), иначе размеры окна всегда будут задаваться по умолчанию (во всю клиентскую область окна `Frame`, независимо от параметров `CRect`). Изменения в файлах приведены в листингах 1.13 и 1.14.

Листинг 1.13. Изменения в файле MainFrm.cpp для изменения размера окна представления

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
   if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
                         CRect(0,0,150,100), this, 4444444, NULL))
   {
      TRACE0("Failed to create view window\n");
      return -1;
   }
   return 0;
}
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
```

```
   if(!CFrameWnd::PreCreateWindow(cs))
      return FALSE;
   cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
   cs.lpszName = L"Новый заголовок окна";
   cs.lpszClass = AfxRegisterWndClass(NULL, LoadCursor(NULL, IDC_WAIT),
                        reinterpret_cast<HBRUSH>(COLOR_ACTIVEBORDER+1),
                        LoadIcon(NULL, IDI_ERROR));
   return TRUE;
}
```

Листинг 1.14. Изменения в файле ChildView.cpp для изменения цвета окна представления и вида курсора

```
// ...
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
   if (!CWnd::PreCreateWindow(cs))
      return FALSE;
   cs.dwExStyle |= WS_EX_CLIENTEDGE;
   cs.style &= ~WS_BORDER;
   cs.lpszClass = AfxRegisterWndClass(NULL, LoadCursor(NULL, IDC_HAND),
                        reinterpret_cast<HBRUSH>(COLOR_INACTIVECAPTION+1),
                        NULL);
   return TRUE;
}
```

Результат работы с этими изменениями приведен на рис. 1.18.

Примечание

Загрузить стандартный курсор и иконку можно и другим способом: вместо вызова `LoadCursor(NULL,IDC_WAIT)` использовать:

```
theApp.LoadStandardCursor(MAKEINTRESOURCE(IDC_WAIT));
```

или

```
AfxGetApp()->LoadStandardCursor(IDC_WAIT);
```

а вместо `LoadIcon(NULL,IDI_ERROR)` использовать:

```
theApp.LoadStandardIcon(MAKEINTRESOURCE(IDI_ERROR));
```

или

```
AfxGetApp()->LoadStandardIcon(IDI_ERROR);
```

Функция `AfxGetApp()` возвращает указатель на объект приложения:

```
CWinApp* AFXAPI AfxGetApp();
```

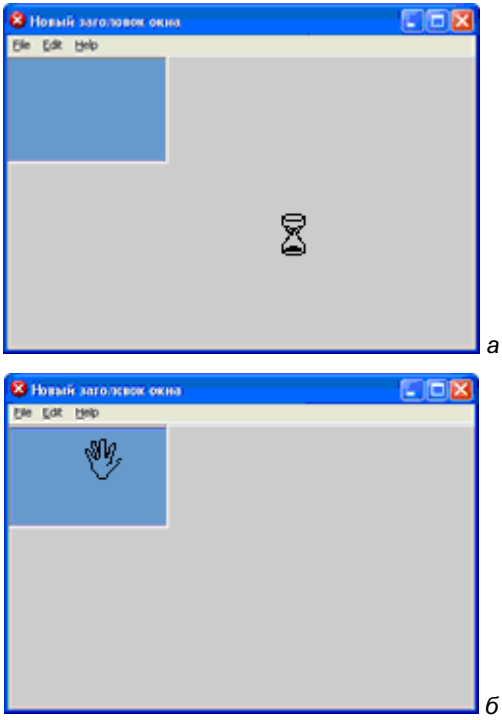


Рис. 1.18. Изменение цвета и размера окон (*а*) и формы курсора (*б*)

1.7.2. Изменения в ресурсах приложения

Таблица акселераторов

Для того чтобы можно было вызывать диалоговое окно **About pr1** не через меню, а с помощью комбинации клавиш, добавим в таблицу акселераторов

идентификатор нужного пункта меню `ID_APP_ABOUT` и комбинацию горячих клавиш <Ctrl>+<A> (рис. 1.19). Идентификатор нужного пункта меню можно посмотреть в карте сообщений (см. листинг 1.2).

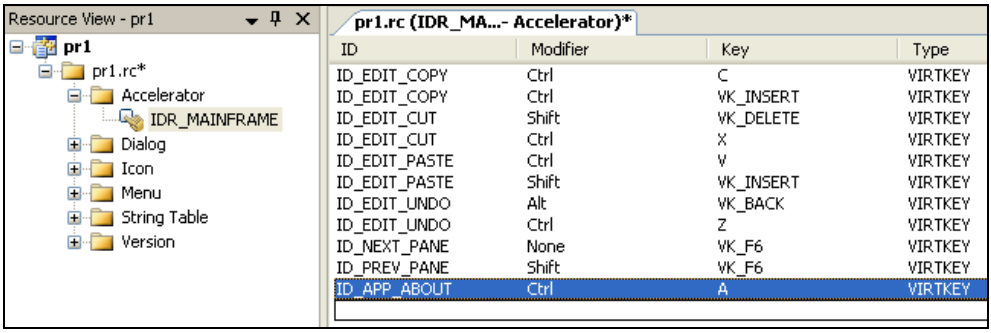


Рис. 1.19. Задание клавиш акселераторов

Окно диалога

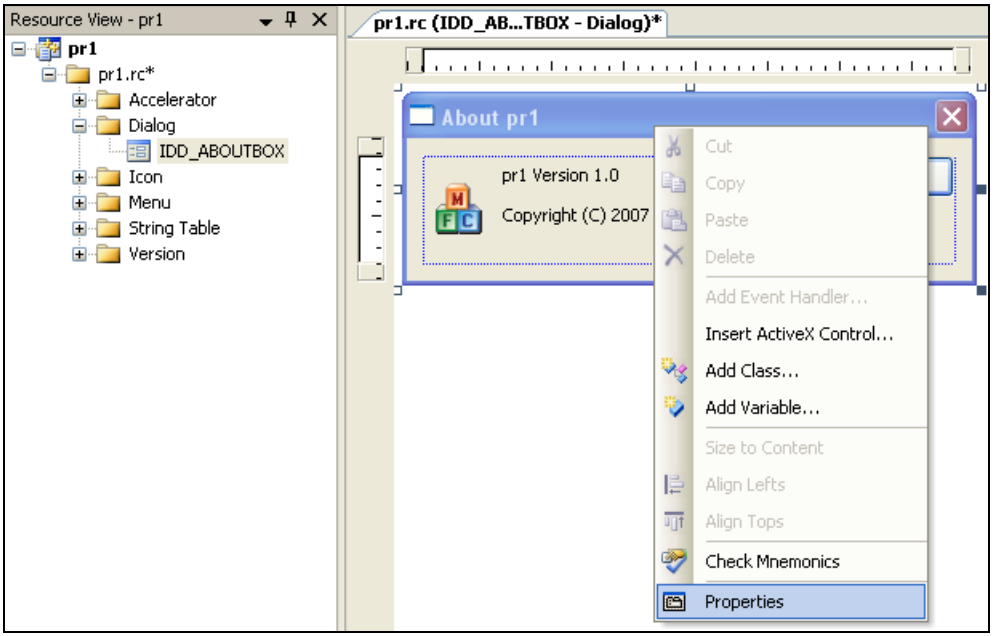
У окна диалога (и у любого элемента в окне) можно посмотреть и изменить свойства. Для вызова окна свойств надо щелкнуть по диалоговому окну (или элементу в окне) правой кнопкой мыши и в контекстном меню выбрать команду **Properties** (Свойства) (рис. 1.20, *а*). Меняем заголовок окна диалога в поле **Caption** с "About pr1" на "О Программе" и нажимаем клавишу <Enter> (рис. 1.20, *б*). Меняем размер окна диалога (так же, как и у стандартных окон Windows — подцепив левой кнопкой мыши рамку окна).

Переносим кнопку **OK** (захватив ее левой кнопкой мыши и перетащив на нужное место), меняем ее размеры (так же, как и у окон Windows). Для изменения текста на кнопке щелкаем по кнопке правой кнопкой мыши и в контекстном меню выбираем команду **Properties**. В окне **Properties** надо изменить заголовок кнопки в поле **Caption** с "OK" на "Кнопка ДА" и нажать клавишу <Enter> (рис. 1.20, *в*).

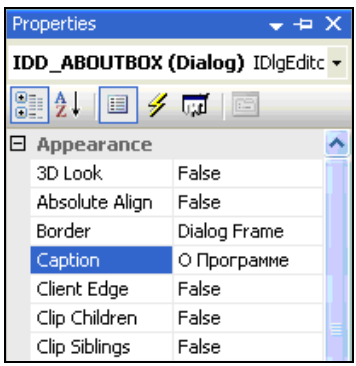
Переносим иконку (захватив ее левой кнопкой мыши и перетаскивая на нужное место). Размеры иконки изменить нельзя (рис. 1.20, *г*).

В диалоговом окне есть два статических элемента (текст "pr1 Version 1.0" и "Copyright (C) 2007"). Меняем их размер, положение и текст. Текст можно

менять в поле **Caption** окна **Properties** или прямо в рамке элемента (рис. 1.20, *д*).

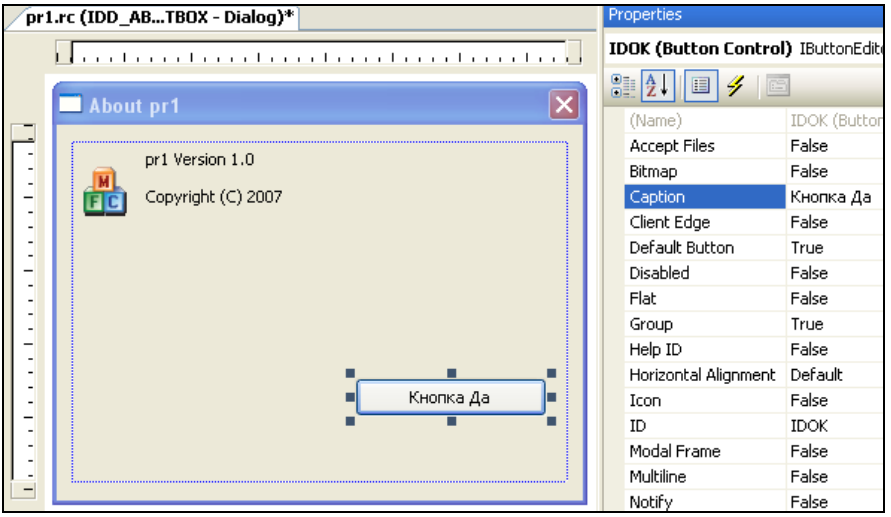


а

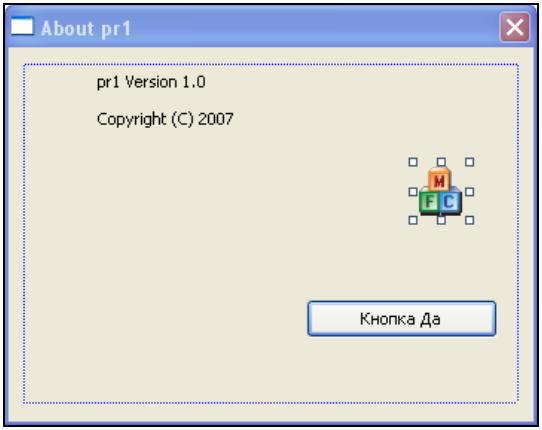


б

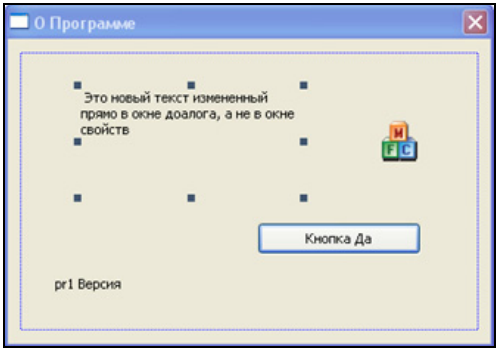
Рис. 1.20. Открытие окна свойств диалога с помощью контекстного меню (*а*), изменение заголовка диалогового окна (*б*)



в



г



д

Рис. 1.20. Изменение положения, размера и названия кнопки (*в*), положения иконки (*г*) и изменение положения и текста статических элементов (*д*)

Иконки приложения

В ресурсах приложения заложено много иконок. Нас интересуют две — иконка окна **About pr1** и иконка приложения. Выберем иконку окна **About pr1**. Для этого надо выполнить команду меню **Image | Current Icon Image Type | 32×32, 16777216** (Изображение | Текущий тип иконки | Размер и количество цветов) (рис. 1.21, *а*). Изменим рисунок иконки, используя панель инструментов (рис. 1.21, *б*). После этого выберем и изменим маленькую иконку приложения **16×16, 16777216** (рис. 1.21, *в* и *г*).

> ***Примечание***
>
> Для старых видеокарт используются иконки с количеством цветов 256.

При изменении иконки приложения надо изменить текст в регистрации класса фреймового окна. Изменения в файле приведены в листинге 1.15.

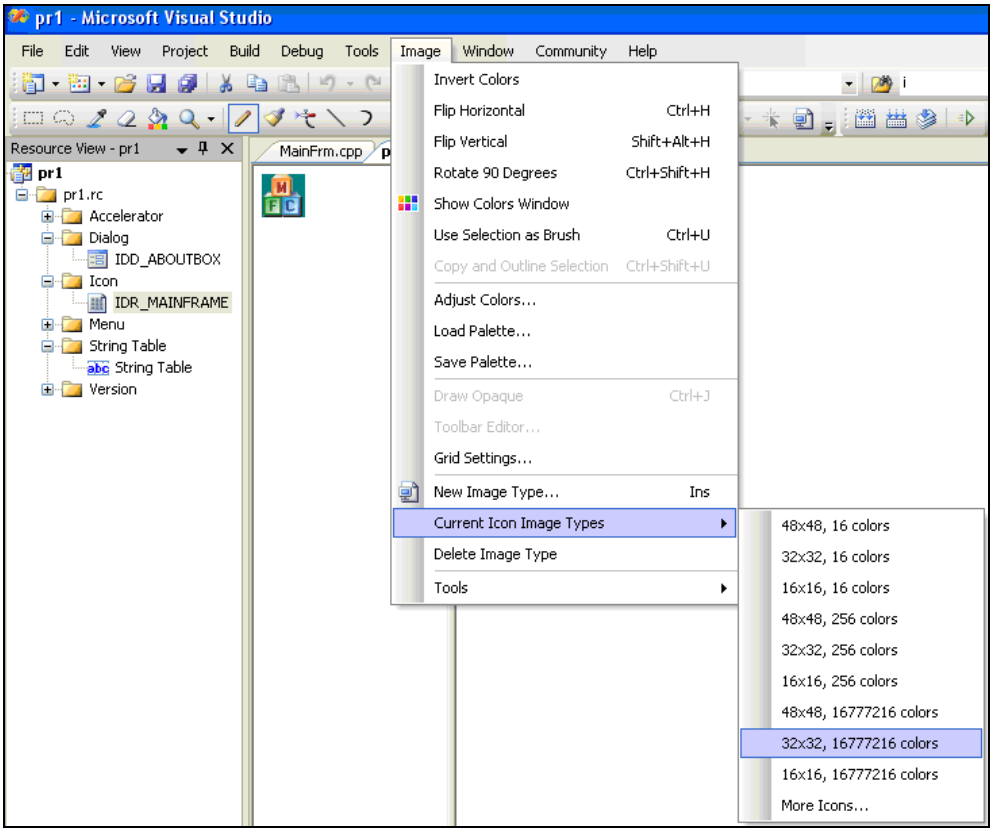
Листинг 1.15. Изменения в файле MainFrm.cpp для изменения вида курсора в фреймовом окне

```
// ...
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
   // ...
   cs.lpszClass = AfxRegisterWndClass(NULL,LoadCursor(NULL,IDC_WAIT),
                     reinterpret_cast<HBRUSH>(COLOR_ACTIVEBORDER+1),
                     LoadIcon(AfxGetInstanceHandle(),
                              MAKEINTRESOURCE(IDR_MAINFRAME)));
   return TRUE;
}
```

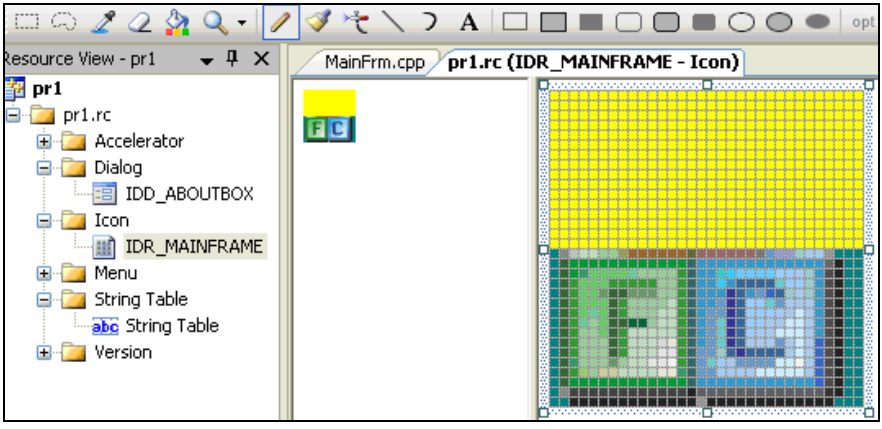
В листинге 1.15 функция `AfxGetInstanceHandle()` позволяет получить дескриптор приложения:

```
HINSTANCE AFXAPI AfxGetInstanceHandle();
```

Макрос `MAKEINTRESOURCE` возвращает указатель на строку с именем ресурса по его заданному идентификатору (`IDR_MAINFRAME`).

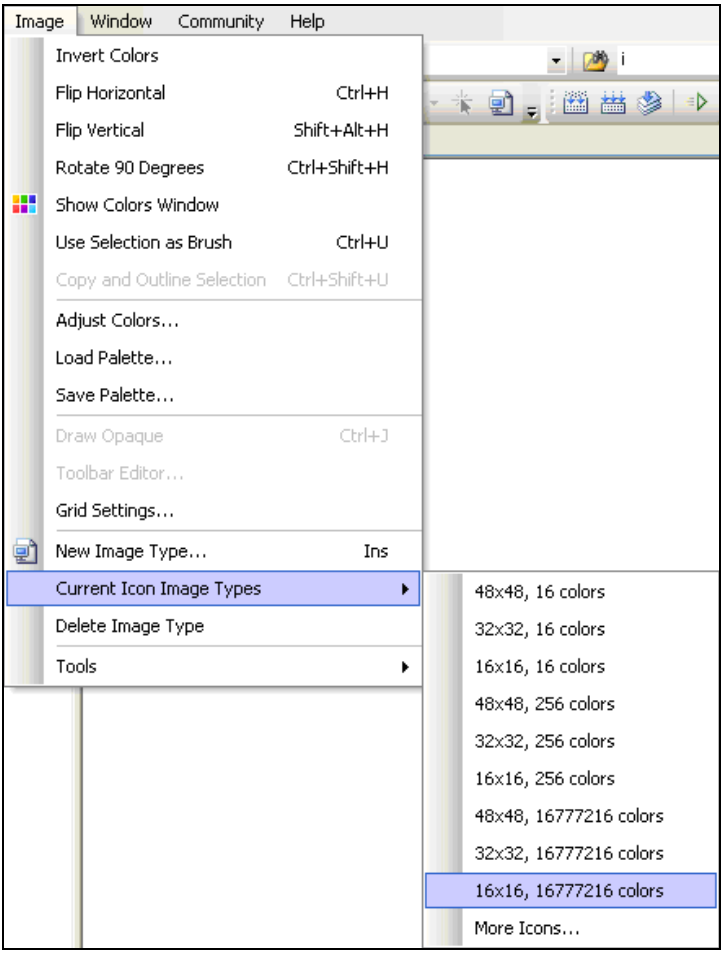


а

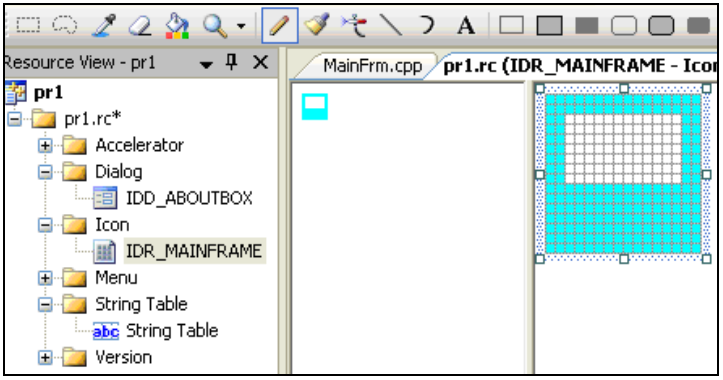


б

Рис. 1.21. Выбор иконки окна **About pr1** (*а*), изменение иконки окна **About pr1** (*б*)



в



г

Рис. 1.21. Выбор иконки приложения (*в*), изменение иконки приложения (*г*)

Примечание

Загрузить иконку можно и другим способом: вместо

```
LoadIcon(AfxGetInstanceHandle(), MAKEINTRESOURCE(IDR_MAINFRAME))
```

использовать

```
LoadIcon(theApp.m_hInstance,MAKEINTRESOURCE(IDR_MAINFRAME))
```

или

```
LoadIcon(AfxGetApp()->m_hInstance, MAKEINTRESOURCE(IDR_MAINFRAME))
```

Меню приложения

Изменим названия пунктов меню для **File | Exit** (Файл | Выход) на русские названия. Для этого надо щелкнуть левой кнопкой мыши в поле меню **File**, в появившемся окне редактирования изменить название пункта и нажать клавишу <Enter>. Аналогичным образом нужно изменить меню **Exit** (рис. 1.22). Это можно было сделать и через поле **Caption** окна **Properties**. Символ '&' в названии меню задает, какая буква будет использоваться для быстрой комбинации выбора пункта меню с помощью клавиатуры (эта буква будет подчеркнута в названии меню, быстрый доступ будет выполняться с помощью комбинации клавиш <Alt>+<*подчеркнутая буква*>). Символ '&' может стоять перед любой буквой (необязательно первой). Регистр не имеет значения, но имеет значение раскладка клавиатуры (русская или английская). Для доступа к меню **ФАЙЛ** — комбинация клавиш <Alt>+<ф>, для доступа к меню **Выход** — последовательное нажатие комбинаций клавиш <Alt>+<ф> и <Alt>+<в>.

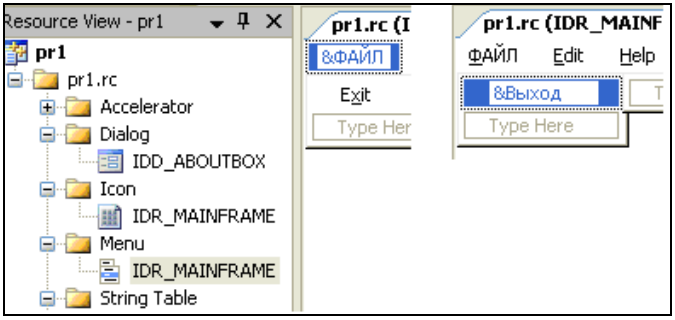


Рис. 1.22. Изменение названий пунктов меню

Заголовок окна приложения

Поменяем заголовок окна приложения в таблице строк на "ГЛАВНОЕ ОКНО ПРИЛОЖЕНИЯ" (рис. 1.23). Но чтобы брался этот заголовок, надо закомментировать заголовок в регистрации класса. Изменения в файле приведены в листинге 1.16.

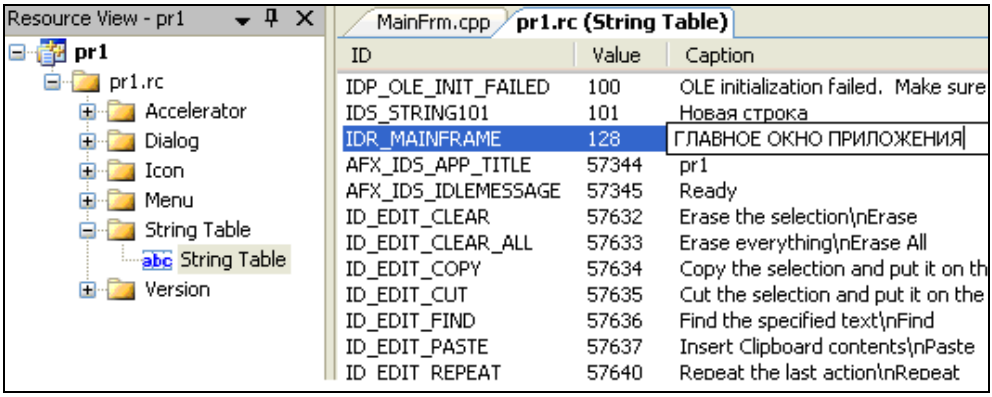


Рис. 1.23. Изменение заголовка окна приложения

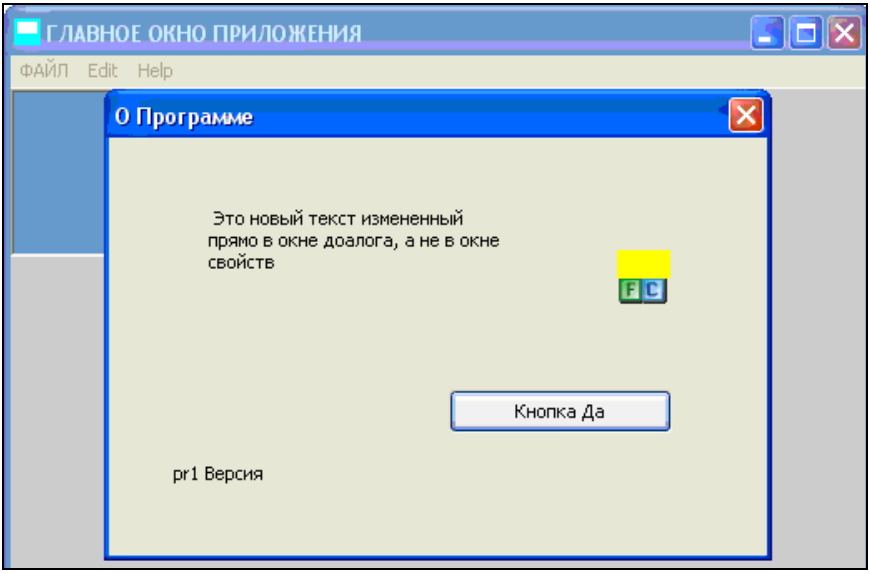


Рис. 1.24. Результат работы программы с внесенными изменениями

Листинг 1.16. Изменения в файле MainFrm.cpp для возможности задания нового заголовка окна приложения через таблицу строк

```
// ...
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
   // ...
   //cs.lpszName = L"Новый заголовок окна";
   // ...
}
```

Результаты работы с внесенными изменениями показаны на рис. 1.24.

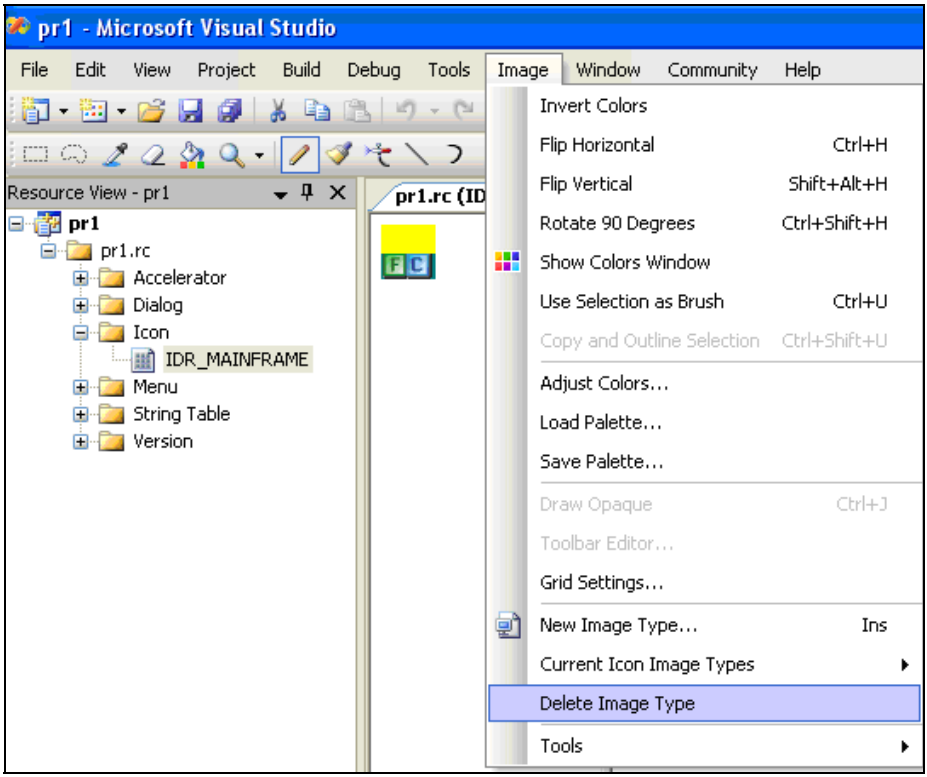


Рис. 1.25. Удаление текущей иконки

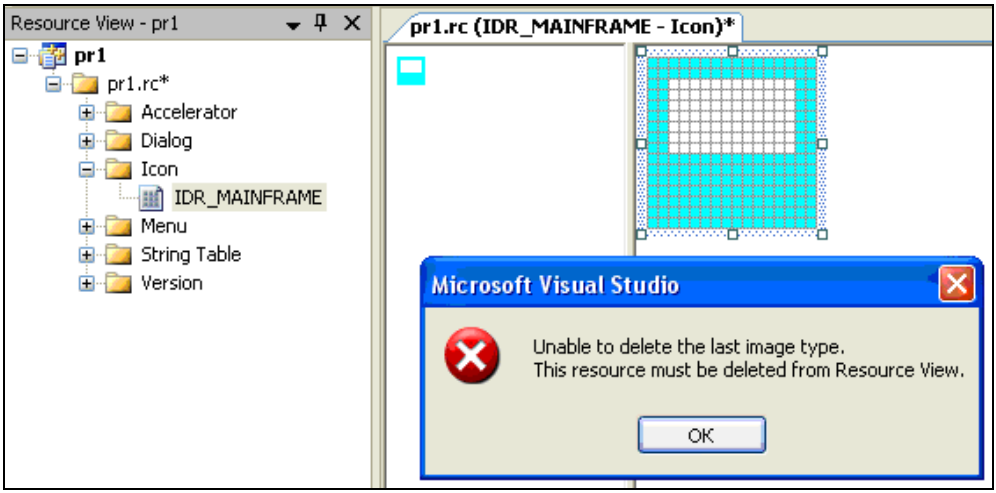


Рис. 1.26. Сообщение при попытке удалить последнюю иконку

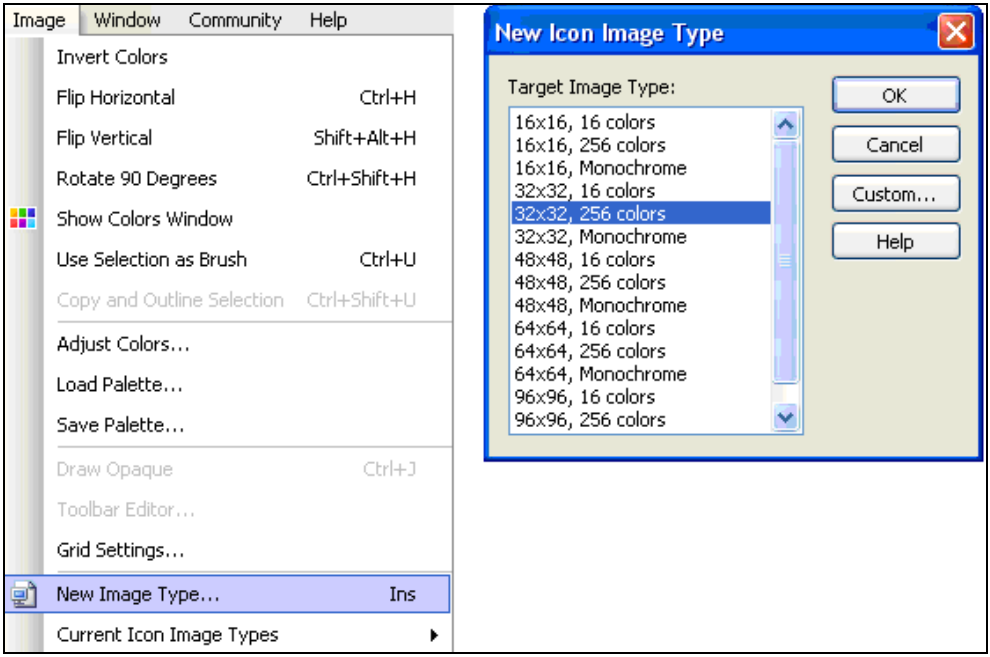


Рис. 1.27. Добавление новой иконки

Если вас не устраивает большое количество иконок, созданных мастером приложения, то можно все удалить и сделать заново только две нужные — для диалогового окна (размер 32 × 32) и для фреймового окна (оставить размер 16 × 16). Если вы хотите, чтобы обе иконки были одинаковые, то иконку для фреймового окна можно не создавать (приложение будет использовать уменьшенную иконку диалогового окна). Удаление иконок показано на рис. 1.25 (удаляется текущая иконка). Надо поочередно удалить все иконки, пока не появится сообщение, показанное на рис. 1.26 (вам не позволят удалить все иконки — одна должна обязательно остаться). Теперь надо добавить свою иконку 32 × 32 (рис. 1.27) и создать изображение (рис. 1.28). Далее можно отредактировать оставшуюся иконку 16 × 16 (выбрав ее в списке) или удалить ее и создать заново. Новая иконка 16 × 16 показана на рис. 1.29. Результат работы такой программы показан на рис. 1.30.

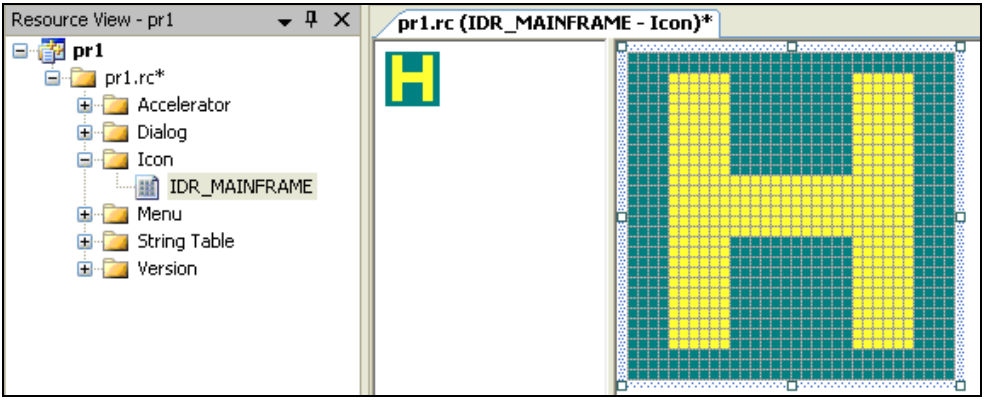


Рис. 1.28. Новая иконка окна **О Программе**

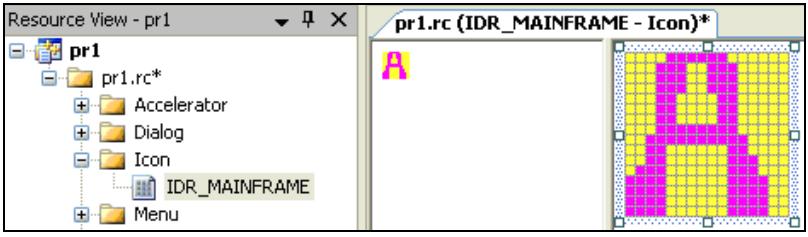


Рис. 1.29. Новая иконка приложения

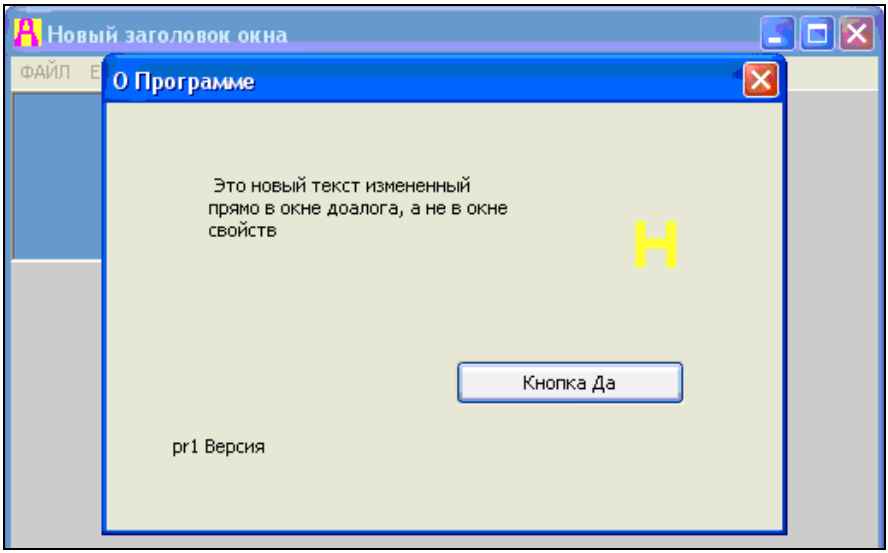


Рис. 1.30. Окончательный результат работы приложения

1.8. Полезные справочные данные

1.8.1. Функции для доступа к данным приложения

Перечислим основные функции, используемые для доступа к различным данным приложения.

- Получение указателя на объект приложения:

  ```
  CwinApp* AFXAPI AfxGetApp();
  ```

  Пример: `AfxGetApp()->LoadStandardCursor(IDC_WAIT);`

- Получения дескриптора приложения:

  ```
  HINSTANCE AFXAPI AfxGetInstanceHandle();
  ```

  Пример:
  `LoadIcon(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDR_MAINFRAME));`

- Получение указателя на строку с именем приложения:

  ```
  LPCTSTR AFXAPI AfxGetAppName();
  ```

  Пример: `AfxMessageBox(AfxGetAppName());`

- Получение указателя на главное окно приложения (`CWnd* CWinThread::m_pMainWnd;`):

```
CWnd* AFXAPI AfxGetMainWnd();
```

Пример 1 (отправление главному окну сообщения о закрытии):

```
AfxGetMainWnd()->PostMessage(WM_CLOSE, 0, 0);
```

Пример 2 (указать главное окно родителем окна сообщения):

```
MessageBox(AfxGetMainWnd()->operator HWND(), L"текст", NULL, NULL);
MessageBox(theApp.m_pMainWnd->operator HWND(), L"текст", NULL, NULL);
```

- Получение дескриптора ресурсов приложения:

```
extern HINSTANCE AfxGetResourceHandle();
```

Пример: `LoadMenu(AfxGetResourceHandle(),MAKEINTRESOURCE(IDR_PANEL));`

- Задание ресурсов приложения:

```
void AFXAPI AfxSetResourceHandle(HINSTANCE hInstResource);
```

Пример:

```
BOOL CMyApp::InitInstance()
{
   HINSTANCE hRes = NULL;
   hRes= LoadLibrary("ResourceD.dll");
   if(hRes)
      AfxSetResourceHandle(hRes);
   return CWinApp::InitInstance();
}
```

1.8.2. Класс приложения *CWinApp*

Класс CWinApp — основной класс, отвечающий за создание и работу приложения. Конструктор:

```
CWinApp::CWinApp(LPCTSTR lpszAppName = NULL);
```

Параметр `lpszAppName` указывает на текстовую строку с именем приложения. Если он равен `NULL`, то используется строка из файла ресурсов с идентификатором `IDR_MAINFRAME` — в ней можно указать имя главного окна приложения.

Члены класса такие:

- `HINSTANCE m_hInstance` — дескриптор приложения. Может быть получен с помощью функции `AfxGetInstanceHandle()`;

- `LPCTSTR m_pszAppName` — указатель на строку с именем приложения. Может быть получен с помощью функции `AfxGetAppName()`;
- `LPCTSTR m_pszExeName` — указатель на строку с именем выполняемого файла без расширения. Обычно совпадает с `m_pszAppName`;
- `BOOL m_bHelpMode` — показатель включения контекстной справки (<Shift>+<F1>): `TRUE` — справка включена, `FALSE` — нет;
- `BOOL m_bUseHtmlHelp` — показатель использования формата справки приложения: `TRUE` — используется HTMLHelp, `FALSE` — WinHelp;
- `LPCTSTR m_pszHelpFilePath` — полное имя пути к файлу справки. Обычно это имя приложения с расширением hlp. При изменении места расположения справки надо изменить значение этой переменной (лучше в `InitInstance()`);
- `LPTSTR m_lpCmdLine` — указатель на командную строку приложения;

  Пример:

```
BOOL CMyApp::InitInstance()
{
   if(m_lpCmdLine[0] == _T('\0'))
      OnFileNew();
   else
      OpenDocumentFile(m_lpCmdLine);
}
```

- `int m_nCmdShow` — определяет режим первоначального отображения главного окна.

  Пример:

```
BOOL CMyApp::InitInstance()
{
   // ...
   CMainFrame* pMainFrame = new CMainFrame;
   if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
      return FALSE;
   m_pMainWnd = pMainFrame;
   pMainFrame->ShowWindow(m_nCmdShow);
   pMainFrame->UpdateWindow();
   // ...
}
```

ГЛАВА 2

Работа с текстом и графикой

Создадим новый проект, на примере которого рассмотрим возможности вывода текста и графических изображений в окно представления.

Создайте проект **pr2** аналогично проекту **pr1** *(см. разд. 1.1)*. При этом надо изменить следующие опции относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ (установить переключатель в положение **Single document**);
  - без поддержки архитектуры документ/представление (снять флажок **Document/View architecture support**);
- на вкладке **User Interface Features**:
  - без строки статуса (снять флажок **Initial status bar**);
  - без панели инструментов (установить переключатель **Toolbars** в положение **None**).

2.1. Описание программы

2.1.1. Работа с текстом

Для работы с текстом надо добавить в класс окна представления новые переменные. Изменения в файле приведены в листинге 2.1.

Листинг 2.1. Изменения в файле ChildView.h для работы с текстом, добавленные вручную

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   COLORREF   oldTxtColor,          // Старый цвет символов
              newTxtColor,          // Новый цвет символов
              oldBkColor,           // Старый цвет фона символов
              newBkColor;           // Новый цвет фона символов
   CFont      newFont,              // Новый шрифт
              *oldFont;             // Указатель на старый шрифт
};
```

Здесь `COLORREF` — тип (целый), позволяющий задать цвет:

```
typedef DWORD COLORREF;
typedef DWORD *LPCOLORREF;
```

Цвет можно задать с помощью макроса `RGB`:

```
COLORREF RGB(
   BYTE byRed,              // Красная составляющая цвета
   BYTE byGreen,            // Зеленая составляющая цвета
   BYTE byBlue);            // Синяя составляющая цвета
```

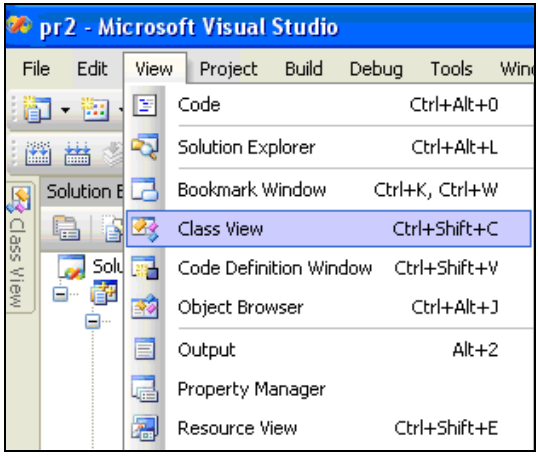
Макрос `RGB` с помощью трех значений (красный, зеленый, синий) от 0 до 255 задает цвета. Например, некоторые стандартные комбинации имеют значения:

- 255, 255, 255 — белый;
- 0, 0, 0 — черный;
- 255, 0, 0 — красный;
- 0, 255, 0 — зеленый;
- 0, 0, 255 — синий;
- 255, 255, 0 — желтый;
- 0, 255, 255 — голубой;

- 255, 0, 255 — сиреневый;
- 192, 192, 192 — светло-серый;
- 128, 128, 128 — темно-серый.

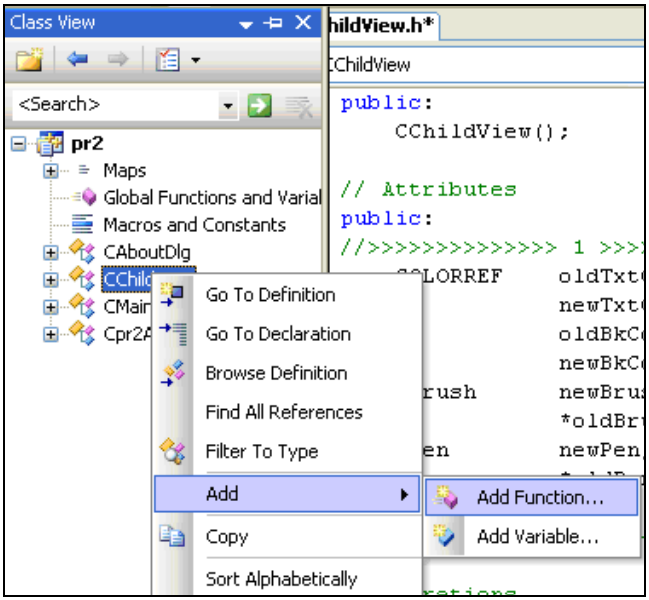
В класс `CChildView` добавим функцию `fText()`, которая будет выполнять вывод текста. Это можно сделать вручную или с помощью мастера:

1. Выполнить команду меню **View | Class View** (Вид | Просмотр классов) (рис. 2.1, *а*).
2. В открывшемся окне **Class View** вызвать для класса `CChildView` контекстное меню (правой кнопкой мыши) и выполнить команду **Add | Add Function** (Добавить | Добавить функцию) (рис. 2.1, *б*).
3. В открывшемся окне **Add Member Function Wizard** (Мастер добавления функций) заполнить следующие поля (рис. 2.1, *в*):
   - **Return type** (Тип возвращаемого значения) — `void`;
   - **Function name** (Имя функции) — `fText`;
   - **Parameter type** (Тип параметра) — `CPaintDC &`;
   - **Parameter name** (Имя параметра) — `dc`;
   - **Comment** (Комментарии) — `Функция для работы с текстом.`

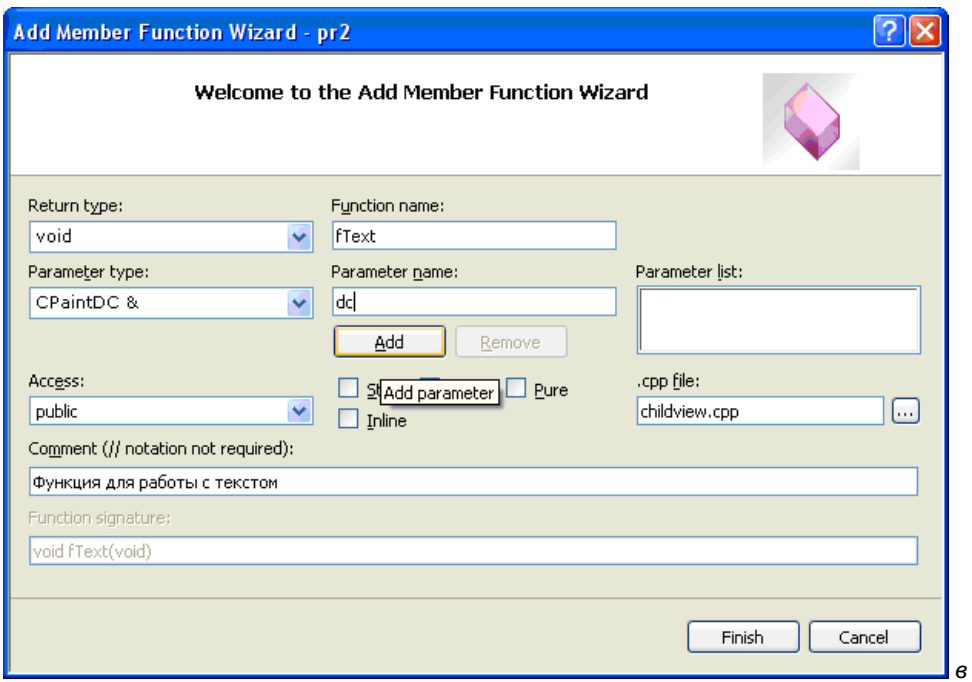


а

Рис. 2.1. Добавление новой функции в класс: открытие окна просмотра классов (*а*)

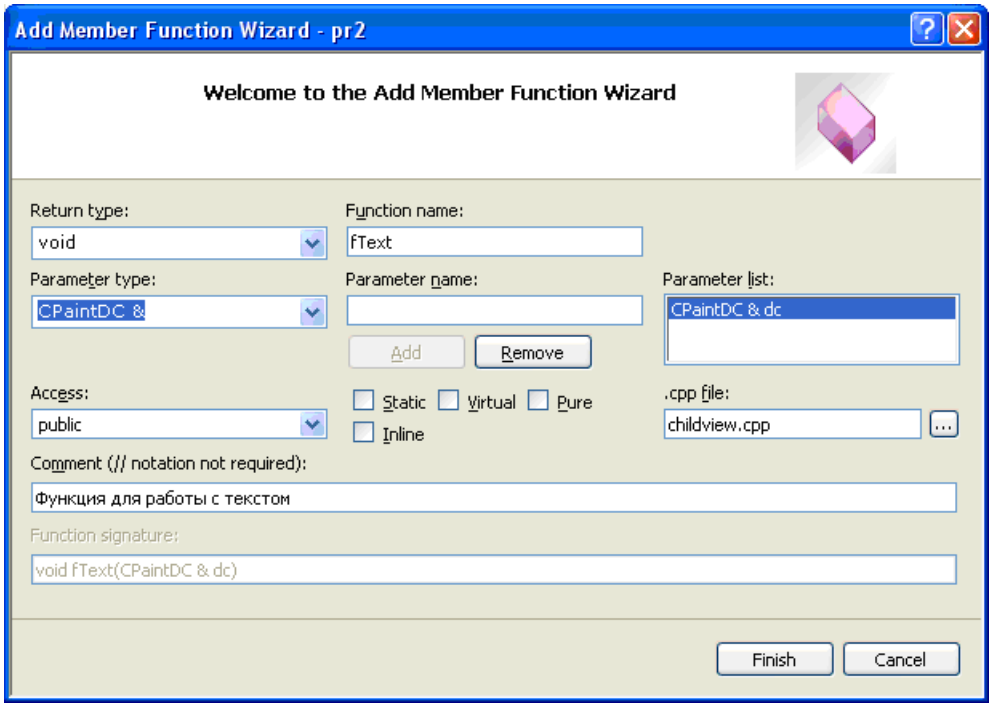


б



в

Рис. 2.1. Вызов мастера добавления функций (*б*); заполнение шаблона функции (*в*)



г

Рис. 2.1. Добавление параметров в функцию (*г*)

4. Нажать кнопку **Add**. В поле **Parameter list** (Список параметров) будет добавлен аргумент `CPaintDC &dc` (рис. 2.1, *г*).

> ***Примечание***
>
> Если у функции должно быть несколько параметров, то надо повторить шаги 3 (заполнив только поля **Parameter type** и **Parameter name**) и 4.

5. Нажать кнопку **Finish**.

Мастер добавит код, приведенный в листингах 2.2 и 2.3.

Листинг 2.2. Изменения в файле ChildView.h для работы с текстом, сделанные мастером

```
// ...
class CChildView : public CWnd
```

```
{
   // ...
public:
   // Функция для работы с текстом
   void fText(CPaintDC &dc);
};
```

Листинг 2.3. Изменения в файле ChildView.cpp для работы с текстом, сделанные мастером

```
// ...
// Функция для работы с текстом
void CChildView::fText(CPaintDC &dc)
{
}
```

Функция `fText()` будет вызываться при обработке сообщения `WM_PAINT` в функции `CChildView::OnPaint()`. Добавим вызов и определение функции `fText()`. Изменения в файлах приведены в листинге 2.4.

Листинг 2.4. Изменения в файле ChildView.cpp для работы с текстом, добавленные вручную

```
// ...
void CChildView::OnPaint()
{
   CPaintDC dc(this);
   fText(dc);
}
// Функция для работы с текстом
void CChildView::fText(CPaintDC & dc)
{
   // Вывод текста (по умолчанию - черный текст на белом фоне)
   // в окно представления
   int x, y;                  // Текущие координаты для вывода текста
   x = 0; y = 0;              // Левый угол окна представления
```

```
CSize sz;                  // Размеры текста в пикселах
int dy;                    // Смещение вниз на заданное количество строк

dc.TextOutW(x, y, L"Text1");          // Вывод текста
sz = dc.GetTextExtent(L"Text1");      // Получение размеров
                                      // выведенного текста
dc.TextOutW(x+sz.cx, y, L"-после Text1");  // Печать текста
                                           // вслед за "Text1"
dy = sz.cy;                           // Вторая строка
dc.TextOutW(x, y+dy, L"под Text1");   // Печать текста под "Text1"
dy += sz.cy;                          // Приращение для координаты Y
                                      // при выводе следующей строки

// "Плохой" вывод текста в несколько строк
dc.TextOutW(x, y+dy, L"Text2\nв две \t\tстроки");
dy += sz.cy;

// "Хороший" вывод текста в несколько строк
int dy2 = dy + sz.cy + sz.cy;
dc.DrawText(L"Text3\nв две \t\tстроки", CRect( x,y+dy, 200, y+dy2 ),
            DT_LEFT | DT_EXPANDTABS);

// Вывод цветного текста
oldTxtColor = dc.GetTextColor();     // Сохраняем старые цвета текста и
oldBkColor = dc.GetBkColor();        // фона
dc.SetTextColor(RGB(0, 255, 0));     // Задаем новые
dc.SetBkColor(RGB(255, 0, 0));
dc.TextOutW(x, y+dy2, L"ЦВЕТНОЙ под Text3");   // Вывод текста
newTxtColor = dc.SetTextColor(oldTxtColor);    // Восстанавливаем
newBkColor = dc.SetBkColor(oldBkColor);        // старые цвета

// Печать текста в центре окна
CRect rect;
GetClientRect(&rect);                // Получение размеров клиентской
                                     // области окна
```

```
int xx = rect.Width();                  // Ширина окна
int yy = rect.Height();                 // Высота окна
sz = dc.GetTextExtent(L"Всегда в центре");
dc.TextOutW(xx/2 - sz.cx/2, yy/2 - sz.cy/2, L"Всегда в центре");

// "Привязка" не к окну, а к экрану
xx = GetSystemMetrics(SM_CXSCREEN);  // Получение ширины и
                                     // высоты экрана
yy = GetSystemMetrics(SM_CYSCREEN);
dc.TextOutW(xx/4, yy/4, L"Стою на месте");

// Выбор другого встроенного шрифта
// Создать новый шрифт из заданных
newFont.CreateStockObject(ANSI_VAR_FONT);
// Подключить новый шрифт и сохранить старый
oldFont = dc.SelectObject(&newFont);
// Новый шрифт подключен, можно удалить объект
newFont.DeleteObject();
dy = dy2 + sz.cy;
dc.TextOutW(1, y+dy, L"Другой шрифт");      // Вывод новым шрифтом
sz = dc.GetTextExtent(L"Другой шрифт");
dy += sz.cy;

// Выбор другого своего шрифта
newFont.CreateFontW(20, 20, 3500, 0,        // Создать новый шрифт
                    FW_DONTCARE, TRUE, TRUE, FALSE, DEFAULT_CHARSET,
                    OUT_CHARACTER_PRECIS, CLIP_DEFAULT_PRECIS,
                    DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE,
                    L"Times New Roman");
dc.SelectObject(&newFont);                  // Подключить новый шрифт
newFont.DeleteObject();                     // Освободить объект
dc.TextOutW(50, y+dy, L"Новый шрифт");

dc.SelectObject(oldFont);                  // Восстановить старый шрифт
}
```

Вывод строки текста в окно выполняется с помощью фукнции:

```
BOOL CDC::TextOut(          // 0 - ошибка
   int x,                   // X-координата начала вывода текста (Xmin = 0)
   int y,                   // Y-координата начала вывода текста (Ymin = 0)
   const CString& str);     // Выводимая строка
```

или

```
virtual BOOL CDC::TextOut(
   int x,                   // X-координата начала вывода текста
   int y,                   // Y-координата начала вывода текста
   LPCTSTR lpszString,      // Указатель на выводимую строку
   int nCount);             // Длина строки (без '\0')
```

Если `nCount` задать больше, чем длина строки, то оставшееся место будет заполнено "мусором" (обычно это символ `'Ъ'`). Если задать `nCount` меньше, чем надо, то выводимая строка будет обрезана до заданного количества символов.

> ***Примечание***
>
> В листинге 2.4 используется функция `TextOutW()`. Выбор варианта названия для многих функций зависит от использования Unicode. Например, при использовании Unicode берется функция `TextOutW()`, без использования Unicode — функция `TextOutA()`. Более подробно об этом говорится в *гл. 7*.

Размер текста (длину `cx` и высоту `cy`) можно получить с помощью функции:

```
CSize CDC::GetTextExtent(          // Структура с размерами текста
   LPCTSTR lpszString,             // Измеряемый текст
   int nCount) const;              // Длина текста
```

или

```
CSize CDC::GetTextExtent(
   const CString &str) const;      // Измеряемый текст
```

Функция `TextOut()` не "понимает" управляющих символов (`'\n'`, `'\t'`), выводит "мусор". Есть еще одна функция вывода текста, более "умная":

```
virtual int CDC::DrawText( // Высота текста(шрифта) в пикселах
   LPCTSTR lpszString,     // Выводимый текст
   int nCount,             // Длина текста в символах
```

```
   LPRECT lpRect,      // Прямоугольная область (Rect), куда
                       // выводится текст. Задает X,Y левого верхнего
                       // угла и X,Y правого нижнего угла (в пикселах)
   UINT nFormat);          // Способ форматирования текста
```

или

```
int CDC::DrawText(
   const CString &str,     // Выводимый текст
   LPRECT lpRect,          // Прямоугольная область (Rect)
   UINT nFormat);          // Способ форматирования текста
```

Значения флагов форматирования текста (`nFormat`) могут быть такие:

- `DT_BOTTOM` — выравнивание по нижней границе `Rect`. Должен использоваться только с `DT_SINGLELINE`;
- `DT_CALCRECT` — определение длины и ширины `Rect` для выводимого текста (полученные данные заносятся в поля `CRect`). При этом сам текст *не выводится*. Например:

```
CRect rect;
dc.DrawText(L"Text3\nв две \t\tстроки", &rect, DT_CALCRECT);
int y = rect.Height();
int x = rect.Width();
```

- `DT_CENTER` — устанавливает горизонтальное выравнивание по центру `Rect`;
- `DT_EXPANDTABS` — при выводе производится замена табулятора на пробелы (если этот флаг не задать, то символы табуляции будут отображаться как `'Ъ'`);
- `DT_EXTERNALLEADING` — определение высоты строки вместе с межстрочным интервалом;
- `DT_HIDEPREFIX` — игнорирование символа `'&'`, который делает стоящую за ним букву подчеркнутой. Например, без этого флага для строки `"Te&xt"` выведется "Te<u>x</u>t", а с этим флагом — `"Text"`;
- `DT_LEFT` — выравнивание по левому краю `Rect`;
- `DT_NOCLIP` — вывод без учета ширины `Rect`;
- `DT_NOPREFIX` — аналогично `DT_HIDEPREFIX`;
- `DT_RIGHT` — выравнивание по правому краю `Rect`;

- `DT_SINGLELINE` — текст выводится в одну строку. Символы перевода строки (`'\n'`) и возврата каретки (`'\r'`) не разбивают строку и отображаются как `'Ъ'`;
- `DT_TABSTOP` — игнорируются все знаки табуляции. Вместо любого количества `'\t'` будет один пробел (при использовании `DT_EXPANDTABS`);
- `DT_TOP` — выравнивание по верхней границе `Rect`. Должен использоваться только с `DT_SINGLELINE`;
- `DT_VCENTER` — вертикальное выравнивание по центру `Rect`. Должен использоваться только с `DT_SINGLELINE`;
- `DT_WORDBREAK` — режим автоматического переноса строки между словами, если следующее слово выходит за границы `Rect`.

Получение текущего цвета текста (возвращаемое значение) выполняется с помощью функции:

```
COLORREF CDC::GetTextColor() const;
```

Получение текущего цвета фона текста (возвращаемое значение):

```
COLORREF CDC::GetBkColor() const;
```

Установить новый цвет текста можно функцией:

```
virtual COLORREF CDC::SetTextColor(      // Старый цвет текста
   COLORREF crColor);                    // Новый цвет текста
```

Установить новый цвет фона:

```
virtual COLORREF CDC::SetBkColor(        // Старый цвет фона
   COLORREF crColor);                    // Новый цвет фона
```

Для того чтобы "привязать" расположение текста к клиентской области окна, используется функция получения размеров клиентской области:

```
void CWnd::GetClientRect(
   LPRECT lpRect) const;       // Указатель на заполняемую структуру
                               // с размерами
```

Кроме членов данных (`left`, `top` — `X`, `Y` координата левого верхнего угла `Rect` и `right`, `bottom` — `X`, `Y` координата правого нижнего угла), у класса `CRect` есть полезные функции:

- получение ширины `Rect` (возвращаемое значение):

  ```
  int CRect::Width() const throw();
  ```

- получение высоты `Rect` (возвращаемое значение):

  ```
  int CRect::Height() const throw();
  ```

- получение координат `Rect` (возвращаемое значение в `CPoint`: `x` и `y`):

```
CPoint CRect::CenterPoint() const throw();
```

Можно сделать "привязку" не к клиентской области окна, а к экрану. Для этого надо получить его текущие размеры. Для этого можно использовать следующую функцию, предназначенную для получения различных данных:

```
int GetSystemMetrics(          // Ответ
   int nIndex);                // Запрашиваемое значение
```

В параметре `nIndex` могут запрашиваться следующие системные данные (в скобках приведено определенное численное значение):

- `SM_CMOUSEBUTTONS` (43) — количество кнопок мыши (0 — если мыши нет);
- `SM_CXBORDER` (5) — ширина рамки окна (в пикселах);
- `SM_CYBORDER` (6) — высота рамки окна (в пикселах);
- `SM_CXCURSOR` (13) — ширина курсора;
- `SM_CYCURSOR` (14) — высота курсора;
- `SM_CXEDGE` (45) — ширина трехмерной рамки;
- `SM_CYEDGE` (46) — высота трехмерной рамки;
- `SM_CXFIXEDFRAME` (7) — ширина рамки окна, которое имеет заголовок, но не может изменить свои размеры;
- `SM_CYFIXEDFRAME` (8) — высота рамки окна, которое имеет заголовок, но не может изменить свои размеры;
- `SM_CXDLGFRAME` (7) — аналогично `SM_CXFIXEDFRAME`;
- `SM_CYDLGFRAME` (8) — аналогично `SM_CYFIXEDFRAME`;
- `SM_CXFRAME` (32) — ширина рамки окна, которое имеет заголовок;
- `SM_CYFRAME` (33) — высота рамки окна, которое имеет заголовок;
- `SM_CXSIZEFRAME` (32) — аналогично `SM_CXFRAME`;
- `SM_CYSIZEFRAME` (33) — аналогично `SM_CYFRAME`;
- `SM_CXFULLSCREEN` (16) — ширина рабочей области максимизированного окна;
- `SM_CYFULLSCREEN` (17) — высота рабочей области максимизированного окна;
- `SM_CXHSCROLL` (21) — ширина горизонтальной линейки прокрутки;

- `SM_CYHSCROLL` (3) — высота горизонтальной линейки прокрутки;
- `SM_CXVSCROLL` (20) — ширина вертикальной линейки прокрутки;
- `SM_CYVSCROLL` (20) — высота вертикальной линейки прокрутки;
- `SM_CXICON` (11) — ширина большой иконки;
- `SM_CYICON` (12) — высота большой иконки;
- `SM_CXSMICON` (49) — ширина малой иконки;
- `SM_CYSMICON` (50) — высота малой иконки;
- `SM_CXMAXIMIZED` (61) — максимально возможная ширина развернутого окна;
- `SM_CYMAXIMIZED` (62) — максимально возможная высота развернутого окна;
- `SM_CXMINIMIZED` (57) — минимальная ширина развернутого окна;
- `SM_CYMINIMIZED` (58) — минимальная высота развернутого окна;
- `SM_CXMAXTRACK` (59) — максимальная ширина, до которой можно изменить размер окна;
- `SM_CYMAXTRACK` (60) — максимальная высота, до которой можно изменить размер окна;
- `SM_CXMINTRACK` (34) — минимальная ширина, до которой можно изменить размер окна;
- `SM_CYMINTRACK` (35) — минимальная высота, до которой можно изменить размер окна;
- `SM_CXMENUSIZE` (54) — ширина меню;
- `SM_CYMENUSIZE` (55) — высота меню;
- `SM_CXMIN` (28) — минимальная допустимая ширина окна;
- `SM_CYMIN` (29) — минимальная допустимая высота окна;
- `SM_CXSCREEN` (0) — ширина экрана;
- `SM_CYSCREEN` (1) — высота экрана;
- `SM_CYCAPTION` (4) — высота заголовка.

Для работы со шрифтами существует класс:

```
class CFont:public CGdiObject
```

Выбрать один из встроенных шрифтов можно с помощью:

```
BOOL CGdiObject::CreateStockObject(      // 0 - ошибка
   int nIndex);                          // Вид встроенного шрифта
```

Виды встроенных шрифтов бывают такие:

- `ANSI_FIXED_FONT` — шрифт с фиксированным размером символов;
- `ANSI_VAR_FONT` — с переменной шириной символов;
- `DEVICE_DEFAULT_FONT` — для данного устройства шрифт, выбираемый по умолчанию;
- `DEFAULT_GUI_FONT` — шрифт, устанавливаемый по умолчанию для графического интерфейса;
- `OEM_FIXED_FONT` — OEM-шрифт (DOS-кодировка 866);
- `SYSTEM_FONT` — системный Windows;
- `SYSTEM_FIXED_FONT` — системный с фиксированным размером.

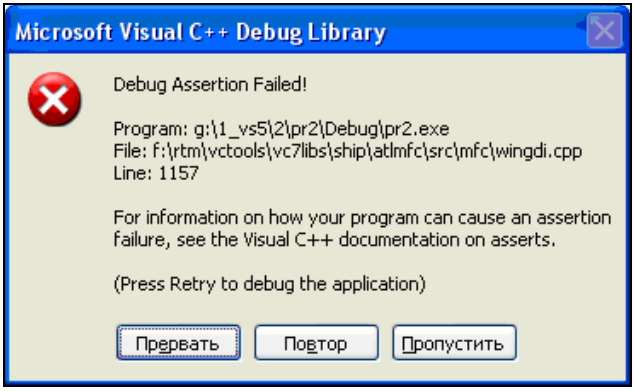


Рис. 2.2. Ошибка при неправильной работе со шрифтами

Затем надо установить выбранный шрифт. Выбор (установка) любого GDI-объекта (шрифт `Font`, перо `Pen`, кисть `Brush` и т. д.) делается с помощью функции `SelectObject()`. Функция в качестве аргумента получает указатель на новый устанавливаемый объект и возвращает указатель на предыдущий (старый) объект:

```
CPen* CDC::SelectObject(CPen* pPen);
CBrush* CDC::SelectObject(CBrush* pBrush);
```

```
virtual CFont* CDC::SelectObject(CFont* pFont);
CBitmap* CDC::SelectObject(CBitmap* pBitmap);
int CDC::SelectObject(CRgn* pRgn);
CGdiObject* CDC::SelectObject(CGdiObject* pObject);
```

После установки нового шрифта его надо удалить. Если этого не сделать, то при дальнейшем вызове функций создания шрифтов будут возникать серьезные ошибки в памяти при работе программы (рис. 2.2). Удаление делается с помощью функции:

```
BOOL CGdiObject::DeleteObject();      // 0 - ошибка
```

Можно не только пользоваться встроенными шрифтами, но и "создавать" свои, на базе установленных:

```
BOOL CFont::CreateFont(     // 0 - ошибка
   int nHeight,             // Высота шрифта
   int nWidth,              // Ширина шрифта
   int nEscapement,     // Угол наклона текста по отношению к оси X
                        // в 0,1 градуса против часовой стрелки
                        // (0 - без наклона, 900 - наклон на 90 градусов)
   int nOrientation,    // Угол наклона букв по отношению к оси X
                        // в 0,1 градуса против часовой стрелки
                        // (0 - без наклона, 900 - наклон на 90 градусов)
   int nWeight,             // Жирность шрифта (от 0 до 1000 или макрос)
   BYTE bItalic,            // TRUE - наклонный шрифт, FALSE - нет
   BYTE bUnderline,         // TRUE - подчеркнутый
   BYTE cStrikeOut,         // TRUE - зачеркнутый
   BYTE nCharSet,           // Набор символов шрифта
   BYTE nOutPrecision,      // Точность выполнения всех заданных значений
                            // шрифта
   BYTE nClipPrecision,     // Точность отсечения, если некоторые символы
                            // выходят за границы области вывода
   BYTE nQuality,           // Желательное качество вывода
   BYTE nPitchAndFamily,    // Тип и семейство шрифтов (тип и семейство
                            // объединяются с помощью '|')
   LPCTSTR lpszFacename);   // Имя шрифта (не более 32 символов,
                            //  включая '\0')
```

Значения жирности шрифта (параметр `nWeight`) могут быть такие:

- `FW_DONTCARE` — 0 (используется значение жирности по умолчанию для определенного шрифта);
- `FW_THIN` — 100;
- `FW_EXTRALIGHT` (или `FW_ULTRALIGHT`) — 200;
- `FW_LIGHT` — 300;
- `FW_NORMAL` (или `FW_REGULAR`) — 400;
- `FW_MEDIUM` — 500;
- `FW_SEMIBOLD` (или `FW_DEMIBOLD`) — 600;
- `FW_BOLD` — 700;
- `FW_EXTRABOLD` (или `FW_ULTRABOLD`) — 800;
- `FW_HEAVY` (или `FW_BLACK`) — 900.

Наборы символов шрифта (`nCharSet`):

- `ANSI_CHARSET`;
- `BALTIC_CHARSET`;
- `CHINESEBIG5_CHARSET`;
- `DEFAULT_CHARSET`;
- `EASTEUROPE_CHARSET`;
- `GB2312_CHARSET`;
- `GREEK_CHARSET`;
- `HANGUL_CHARSET`;
- `MAC_CHARSET`;
- `OEM_CHARSET`;
- `RUSSIAN_CHARSET`;
- `SHIFTJIS_CHARSET`;
- `SYMBOL_CHARSET`;
- `TURKISH_CHARSET`;
- `VIETNAMESE_CHARSET`.

Значения точности шрифта (параметр `nOutPrecision`) могут быть следующими:

- `OUT_DEFAULT_PRECIS` — выбор по умолчанию;
- `OUT_DEVICE_PRECIS` — если в системе установлено несколько шрифтов с одним именем, будет выбран системный шрифт;
- `OUT_OUTLINE_PRECIS` — выбор шрифта только из TrueType;
- `OUT_RASTER_PRECIS` — если в системе установлено несколько шрифтов с одним именем, будет выбран растровый шрифт;
- `OUT_STRING_PRECIS` — этот флаг не используется процедурой масштабирования шрифтов, но эта величина возвращается при нумерации растровых шрифтов;
- `OUT_STROKE_PRECIS` — этот флаг не используется процедурой масштабирования шрифтов, но эта величина возвращается при нумерации шрифтов TrueType;
- `OUT_TT_ONLY_PRECIS` — выбор только шрифтов TrueType. Если таких шрифтов нет, берется `OUT_DEFAULT_PRECIS`;
- `OUT_TT_PRECIS` — если в системе установлено несколько шрифтов с одним именем, будет выбран шрифт TrueType.

Значения точности отсечения (параметр `nClipPrecision`) бывают такими:

- `CLIP_DEFAULT_PRECIS` — выбор по умолчанию;
- `CLIP_EMBEDDED` — этот флаг необходимо указывать при использовании внедренных шрифтов;
- `CLIP_LH_ANGLES` — отсчет угла поворота (против или по часовой стрелке) определяется выбранной системой координат (для данного проекта — против часовой стрелки). Если это значение не задать, то всегда будет поворот против часовой стрелки;
- `CLIP_STROKE_PRECIS` — этот флаг не используется процедурой масштабирования шрифтов, но эта величина возвращается при нумерации шрифтов TrueType.

Значения качества вывода текста (параметр `nQuality`) могут быть такими:

- `DEFAULT_QUALITY` — качество по умолчанию;
- `DRAFT_QUALITY` — "черновое" качество;
- `PROOF_QUALITY` — высокое качество.

Типы шрифтов (параметр `nPitchAndFamily`) следующие:

- `DEFAULT_PITCH` — тип по умолчанию;
- `FIXED_PITCH` — непропорциональный шрифт (все символы имеют одинаковый размер);
- `VARIABLE_PITCH` — пропорциональный шрифт.

Семейства шрифтов (параметр `nPitchAndFamily`) могут быть такими:

- `FF_DECORATIVE` — декоративный шрифт;
- `FF_DONTCARE` — семейство шрифта не имеет значения;
- `FF_MODERN` — непропорциональный шрифт (например, Courier New);
- `FF_ROMAN` — пропорциональный шрифт (например, Times New Roman);
- `FF_SCRIPT` — рукописный шрифт (например, Script MT Bold);
- `FF_SWISS` — пропорциональный шрифт (например, Microsoft Sans Serif).

Если задать вместо имени шрифта (параметр `lpszFacename`) `NULL`, то будет выбран наиболее подходящий шрифт с использованием других параметров и `nCharSet`.

Если задавать все параметры шрифта прямо в функции неудобно (например, если в процессе работы программы надо менять некоторые характеристики шрифта), то можно использовать структуру `LOGFONT` и функцию `CreateFontIndirect()`:

```
typedef struct tagLOGFONT
{
   LONG lfHeight;                    // см. CreateFont()
   LONG lfWidth;
   LONG lfEscapement;
   LONG lfOrientation;
   LONG lfWeight;
   BYTE lfItalic;
   BYTE lfUnderline;
   BYTE lfStrikeOut;
   BYTE lfCharSet;
   BYTE lfOutPrecision;
   BYTE lfClipPrecision;
   BYTE lfQuality;
   BYTE lfPitchAndFamily;
```

```
   TCHAR lfFaceName[LF_FACESIZE];
} LOGFONT, *PLOGFONT;


BOOL CFont::CreateFontIndirect(     // 0 - ошибка
   const LOGFONT* lpLogFont);       // Указатель на заполненную
                                    // структуру LOGFONT
```

Пример использования `LOGFONT` и `CreateFontIndirect()`:

```
   LOGFONT lfont = {20, 20, 3500, 0, FW_DONTCARE, TRUE, TRUE, FALSE,
                    DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
                    CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                    DEFAULT_PITCH | FF_DONTCARE, NULL};
   // ...
   lfont.lfEscapement = 0;
   newFont.CreateFontIndirectW(&lfont);      // Создать новый шрифт
   dc.SelectObject( &newFont );              // Подключить новый шрифт
   newFont.DeleteObject();
   dc.TextOutW(50, y+dy, L"Новый шрифт");
```

Получить характеристики текущего шрифта и занести их в структуру можно с помощью функции:

```
int CFont::GetLogFont(                    // 0 - ошибка
   LOGFONT * pLogFont);                   // Указатель структуру LOGFONT
```

Очень часто надо просто изменить размеры шрифта (не заполняя все параметры нового шрифта). Для этого используется функция:

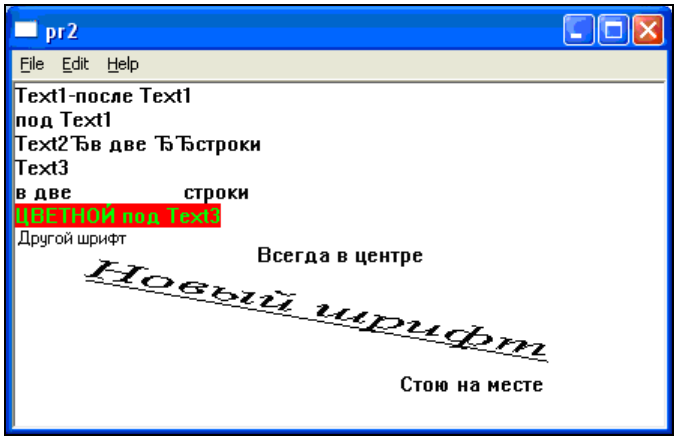
```
BOOL CFont::CreatePointFont(              // 0 - ошибка
   int nPointSize,                        // Размер шрифта
   LPCTSTR lpszFaceName,                  // Имя шрифта
   CDC* pDC = NULL);                      // Контекст устройства
```

Параметр `nPointSize` определяет 0,1 размера шрифта (если надо размер 20, то следует задать значение 200). Если `lpszFaceName = NULL`, то будет ошибка в работе программы (см. рис. 2.2). Параметр `pDC` нужен для правильного преобразования размера шрифта в логические единицы контекста устройства (по умолчанию это пикселы экрана). Пример использования `CreatePointFont()`:

```
   newFont.CreatePointFont(200,L"Times New Roman");
   dc.SelectObject(&newFont);
```

```
newFont.DeleteObject();
dc.TextOutW(50, y+dy, L"Новый шрифт");
```

Результаты работы с текстом показаны на рис. 2.3.



а

б

Рис. 2.3. Работа с текстом и шрифтами (*а*) и положение текста при изменении размеров окна (*б*)

2.1.2. Работа с пером

Для работы с пером надо добавить в класс окна представления новые переменные. Изменения в файле приведены в листинге 2.5.

Листинг 2.5. Изменения в файле ChildView.h для работы с пером, добавленные вручную

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   CPen   newPen,       // Новое перо
          *oldPen;      // Указатель на старое перо
};
```

Аналогично функции `fText()` *(см. разд. 2.1.1)* добавим в класс `CChildView` функцию для работы с пером `fPen()`. Изменения в файлах приведены в листингах 2.6 и 2.7.

Листинг 2.6. Изменения в файле ChildView.h для работы с пером, сделанные мастером

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   // Функция для работы с пером
   void fPen(CPaintDC & dc);
};
```

Листинг 2.7. Изменения в файле ChildView.cpp для работы с пером, сделанные мастером

```
// ...
// Функция для работы с пером
void CChildView::fPen(CPaintDC &dc)
{
}
```

Добавим вызов функции `fPen()`, закомментировав вызов `fText()` (чтобы окно представления было чистым для рисования) и определение `fPen()`. Изменения в файле приведены в листинге 2.8.

Листинг 2.8. Изменения в файле ChildView.cpp для работы с пером, добавленные вручную

```
// ...
void CChildView::OnPaint()
{
   CPaintDC dc(this);
   // fText(dc);
   fPen(dc);
}
// Функция для работы с пером
void CChildView::fPen(CPaintDC &dc)
{
   // Создать новое перо
   newPen.CreatePen(PS_SOLID , 15, RGB( 128, 128, 128));
   oldPen = dc.SelectObject(&newPen);      // Подключить перо
   dc.MoveTo(0, 20);                       // Установить текущую позицию
   dc.LineTo(100, 20);                     // Нарисовать линию

   // Изменить текущее перо
   LOGPEN logpen;                          // Характеристики пера
   newPen.GetLogPen(&logpen);    // Получить характеристики текущего пера
   logpen.lopnWidth.x = 5;                 // Изменить ширину пера
   logpen.lopnColor = RGB(255, 0, 0);      // Изменить цвет пера
   newPen.DeleteObject();                  // Удалить старое перо
   newPen.CreatePenIndirect(&logpen);      // Создать новое перо
   dc.SelectObject(&newPen);               // Подключить новое перо
   dc.LineTo(200, 20);                     // Нарисовать линию
   newPen.DeleteObject();                  // Удалить перо
```

```
   // Создать геометрическое перо
   LOGBRUSH logBrush;                    // Характеристики кисти
   logBrush.lbStyle = BS_SOLID;          // Только этот стиль (для пера)
   logBrush.lbColor = RGB(0,255,0);      // Цвет (для пера)
   newPen.CreatePen(PS_DOT | PS_GEOMETRIC | PS_ENDCAP_SQUARE, 30,
                    &logBrush);
   dc.SelectObject(&newPen);
   dc.LineTo(600, 20);
   newPen.DeleteObject();
   dc.SelectObject(oldPen);              // Восстановить старое перо
}
```

Для работы с пером (рисование линий, приямоугольников, элипсов и т. д.) существует класс:

```
class CPen:public CGdiObject
```

Рассмотрим функции класса `CPen` для работы с пером.

Создание пера выполняется с помощью функции:

```
BOOL CPen::CreatePen(         // 0 - ошибка
   int nPenStyle,              // Стиль пера
   int nWidth,                 // Толщина пера
   COLORREF crColor);          // Цвет пера
```

Стили пера (параметр `nPenStyle`) могут быть такими:

- `PS_SOLID` — сплошная линия;
- `PS_DASH` — длинные штрихи толщиной 1 пиксел (_ _ _ _);
- `PS_DOT` — короткие штрихи толщиной 1 пиксел (. . . .);
- `PS_DASHDOT` — штрихпунктир толщиной 1 пиксел (_ . _ . _);
- `PS_DASHDOTDOT` — двойной штрихпунктир толщиной 1 пиксел (_ . . _ . .);
- `PS_NULL` — прозрачная линия (невидимое перо);
- `PS_INSIDEFRAME` — сплошная линия для рисования по внутренней границе замкнутой области (прямоугольник и т. д.).

Если в процессе работы надо менять некоторые параметры пера, то можно воспользоваться функцией:

```
BOOL CPen::CreatePenIndirect( // 0 - ошибка
   LPLOGPEN lpLogPen);        // Указатель на структуру с данными о пере
```

Структура с данными о пере определена как:

```
typedef struct tagLOGPEN
{
   UINT lopnStyle;            // Стиль пера
   POINT lopnWidth;           // Толщина пера
   COLORREF lopnColor;        // Цвет пера
} LOGPEN;
```

Получить характеристики текущего пера и занести их в структуру можно с помощью следующей функции:

```
int CPen::GetLogPen(          // 0 - ошибка
   LOGPEN* pLogPen);          // Указатель на структуру LOGPEN
```

Можно воспользоваться встроенными перьями с помощью функции:

```
BOOL CGdiObject::CreateStockObject(int nIndex);
```

Виды встроенных перьев (`nIndex`) могут быть такими:

- `BLACK_PEN` — черное перо толщиной 1 пиксел;
- `WHITE_PEN` — белое перо толщиной 1 пиксел;
- `DC_PEN` — перо данного устройства по умолчанию (черное).

Пример использования функции `CreateStockObject()`:

```
   newPen.CreateStockObject(DC_PEN);
   dc.SelectObject(&newPen);
```

Для всех линий штриховки толщина должна быть равна 1 пикселу. Если поставить большее значение, то будет сплошная линия. Чтобы можно было рисовать толстую штриховку надо создать геометрическое перо и использовать второй вариант создания пера:

```
BOOL CPen::CreatePen(               // 0 - ошибка
   int nPenStyle,                   // Стиль пера
   int nWidth,                      // Толщина пера
   const LOGBRUSH* pLogBrush,       // Указатель на структуру
                                    // с данными о кисти
```

```
   int nStyleCount = 0,            // Длина массива стилей
   const DWORD* lpStyle = NULL);   // Указатель на массив стилей
```

Существуют также дополнительные стили пера, которые объединяются с обычными с помощью символа `'|'` и используются только для геометрического пера:

- типы пера:
  - `PS_COSMETIC` — обычное перо (по умолчанию);
  - `PS_GEOMETRIC` — геометрическое перо;
- типы концов линий:
  - `PS_ENDCAP_ROUND` — концы линий закруглены (по умолчанию);
  - `PS_ENDCAP_SQUARE` — концы линий квадратные;
- типы соединения линий:
  - `PS_JOIN_BEVEL` — угол соединения срезан;
  - `PS_JOIN_MITER` — угол соединения обычный (по умолчанию);
  - `PS_JOIN_ROUND` — угол соединения закругленный.

При выборе геометрического пера используется структура для работы с кистью:

```
typedef struct tagLOGBRUSH
{
   UINT lbStyle;             // Стиль кисти
   COLORREF lbColor;         // Цвет кисти
   LONG lbHatch;             // Стиль штриховки
} LOGBRUSH, *PLOGBRUSH;
```

Для применения данной структуры при работе с графическим пером должны быть заданы `lbStyle = BS_SOLID` и `lbColor` — необходимый цвет пера. Параметр `lbHatch` задавать не надо.

В отличие от шрифтов перо можно удалять только после рисования, а не после подключения `SelectObject()` (иначе будет использовано старое перо).

Для рисования созданным пером есть следующие функции:

- установка текущей позиции:

```
   CPoint CDC::MoveTo(          // Старые координаты
      int x,                    // X-координата
      int y);                   // Y-координата
```

или

```
CPoint CDC::MoveTo(
   POINT point);
```

- рисование линии от текущей позиции до заданной:

```
BOOL CDC::LineTo(            // 0 - ошибка
   int x,                    // X-координата конца линии
   int y);                   // Y-координата конца линии
```

или

```
BOOL CDC::LineTo(
   POINT point);
```

Результаты работы программы показаны на рис. 2.4.

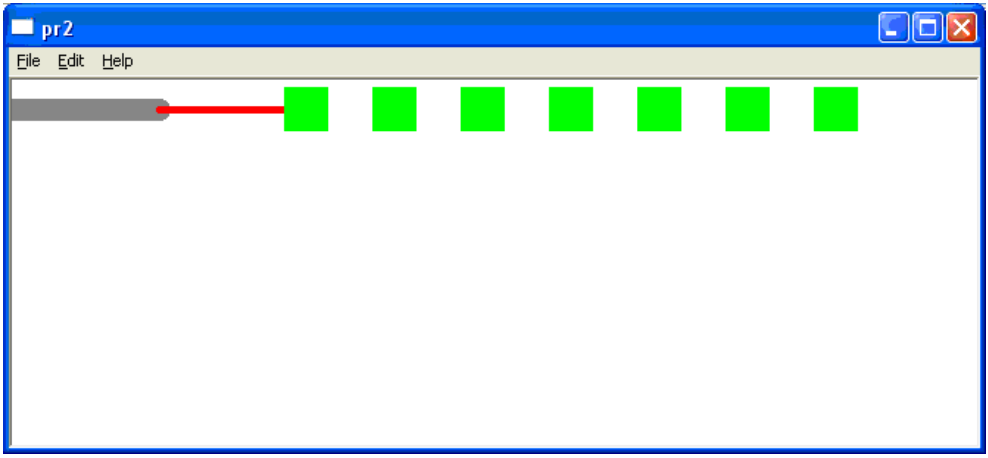


Рис. 2.4. Результаты работы с пером

Для вывода одной цветной точки (пиксела) есть функция:

```
COLORREF CDC::SetPixel(  // Старый цвет пиксела или -1 при ошибке
                         // (например, если точка не попадает
                         // в рабочую область окна)
   int x,                // X-координата точки
   int y,                // Y-координата точки
   COLORREF crColor);    // Цвет вывода точки
```

```
COLORREF CDC::SetPixel(
   POINT point,          // Координаты точки
   COLORREF crColor);    // Цвет вывода точки
```

2.1.3. Работа с кистью

Для работы с кистью надо добавить в класс окна представления новые переменные. Изменения в файле приведены в листинге 2.9.

Листинг 2.9. Изменения в файле ChildView.h для работы с кистью, добавленные вручную

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   CBrush   newBrush,          // Новая кисть
            *oldBrush;         // Указатель на старую кисть
};
```

Аналогично функции `fText()` *(см. разд. 2.1.1)* добавим в класс `CChildView` функцию для работы с кистью `fBrush()`. Изменения в файлах приведены в листингах 2.10 и 2.11.

Листинг 2.10. Изменения в файле ChildView.h для работы с кистью, сделанные мастером

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   // Функция для работы с кистью
   void fBrush (CPaintDC & dc);
};
```

Листинг 2.11. Изменения в файле ChildView.cpp для работы с кистью, сделанные мастером

```
// ...
// Функция для работы с кистью
void CChildView::fBrush (CPaintDC &dc)
{
}
```

Добавим вызов функции `fBrush()`, закомментировав вызов `fPen()` (чтобы окно представления было чистым для рисования) и ее определение. Изменения в файле приведены в листинге 2.12.

Листинг 2.12. Изменения в файле ChildView.cpp для работы с кистью, добавленные вручную

```
// ...
void CChildView::OnPaint()
{
   CPaintDC dc(this);
   // fText(dc);
   // fPen(dc);
   fBrush(dc);
}
// Функция для работы с кистью
void CChildView::fBrush(CPaintDC & dc)
{
   // Создать новую кисть
   newBrush.CreateSolidBrush(RGB(0, 255, 0));
   oldBrush = dc.SelectObject(&newBrush);      // Подключить кисть
   dc.Rectangle(10, 10, 40, 60);               // Нарисовать
прямоугольник
   dc.FillRect(CRect(10, 80, 300, 90), &newBrush);  // Закрасить область
   dc.TextOutW(270, 60, L"FillRect");

   // Изменить текущую кисть
   LOGBRUSH logBrush;                          // Характеристики кисти
```

```
newBrush.GetLogBrush(&logBrush);          // Получить характеристики
                                          // текущей кисти
logBrush.lbStyle = BS_HATCHED;            // Изменить характеристики
logBrush.lbColor = RGB(0, 192, 192);      // (цвет и штриховка)
logBrush.lbHatch = HS_CROSS;
newBrush.DeleteObject();                  // Удалить старую кисть
newBrush.CreateBrushIndirect(&logBrush);  // Создать новую кисть
dc.SelectObject(&newBrush);               // Подключить ее
dc.Rectangle(60, 10, 90, 60);             // Нарисовать прямоугольник
newBrush.DeleteObject();                // Удалить кисть

// Создать штрих-кисть
newBrush.CreateHatchBrush(HS_BDIAGONAL, RGB(0, 0, 255));
dc.SelectObject(&newBrush);
dc.Rectangle(110, 10, 140, 60);
newBrush.DeleteObject();

// Создать кисть для заполнения битовым узором
// Задать массив кодов для битового узора
WORD HatchBits[8] = {10, 110, 210, 310, 10, 110, 210, 310 };
// Создать растр с битовым изображением
CBitmap bm;
bm.CreateBitmap(8,8,1,1, HatchBits);
newBrush.CreatePatternBrush(&bm);         // Создать растровую кисть
dc.SelectObject(&newBrush);
dc.Rectangle(160, 10, 190, 60);
newBrush.DeleteObject();
dc.SelectObject(oldBrush);                // Восстановить старую кисть

// Рисование фигур
CRect rect(10, 110, 90, 190);
dc.Arc(rect,                              // Дуга
       CPoint(rect.right, rect.CenterPoint().y),
       CPoint(rect.left, rect.CenterPoint().y));
dc.TextOutW(40, 160, L"Arc");
```

```
    dc.Chord(CRect(110, 110, 190, 190),       // Дуга с хордой
             CPoint(190, 150), CPoint(110, 150));
    dc.TextOutW(130, 160, L"Chord");

    dc.Ellipse( CRect(210, 110, 290, 150));    // Эллипс
    dc.TextOutW(230, 160, L"Ellipse");

    dc.Pie(CRect(310, 110, 390, 190),          // Сектор
           CPoint(390, 130), CPoint(310, 130));
    dc.TextOutW(340, 160, L"Pie");

    dc.Rectangle(CRect(10, 210, 90, 270));     // Прямоугольник
    dc.TextOutW(20, 280, L"Rectangle");

    dc.RoundRect(CRect(110, 210, 190, 270),  // Закругленный прямоугольник
                 CPoint(20, 20));
    dc.TextOutW(110, 280, L"RoundRect");

    CPoint pts[5];                           // Многоугольник
    pts[0] = CPoint(210, 270);
    pts[1] = CPoint(230, 210);
    pts[2] = CPoint(250, 240);
    pts[3] = CPoint(270, 210);
    pts[4] = CPoint(90, 270);
    dc.Polygon(pts, 5);
    dc.TextOutW(230, 280, L"Polygon");

    pts[0].x += 100;                         // Кривая
    pts[1].x += 100;
    pts[2].x += 100;
    pts[3].x += 100;
    pts[4].x += 100;
    dc.Polyline(pts, 5);
    dc.TextOutW(330, 280, L"Polyline");
}
```

Для работы с кистью (закрашивание замкнутых объектов) существует класс:

```
class CBrush:public CGdiObject
```

Создать новую кисть можно с помощью функций `CreateSolidBrush()`, `CreateBrushIndirect()`, `CreateHatchBrush()` и `CreatePatternBrush()`. Рассмотрим каждую из них.

Создание сплошной кисти выполняется функцией:

```
BOOL CBrush::CreateSolidBrush(         // 0 - при ошибке
   COLORREF crColor);                  // Цвет кисти
```

Создать кисть с помощью структуры можно с помощью функции:

```
BOOL CBrush::CreateBrushIndirect(      // 0 - при ошибке
   const LOGBRUSH* lpLogBrush);        // Указатель на структуру
```

Структура с данными о кисти определена так:

```
typedef struct tagLOGBRUSH
{
   UINT lbStyle;                      // Стиль кисти
   COLORREF lbColor;                  // Цвет кисти
   LONG lbHatch;                      // Стиль штриховки
} LOGBRUSH, *PLOGBRUSH;
```

Стили кисти (поле `lbStyle`) могут быть такими:

- `BS_HATCHED` — используется совместно со стилем штриховки;
- `BS_NULL` — нулевая (прозрачная кисть);
- `BS_SOLID` — сплошния кисть (штриховка игнорируется).

Стили штриховки (поле `lbHatch`) бывают следующими:

- `HS_BDIAGONAL` — диагональная направо 45°;
- `HS_CROSS` — крест-накрест;
- `HS_DIAGCROSS` — диагональная штриховка крест-накрест направо 45°;
- `HS_FDIAGONAL` — диагональная налево 45°;
- `HS_HORIZONTAL` — горизонтальная штриховка;
- `HS_VERTICAL` — вертикальная.

Поля структуры можно заполнить или исправить. Получить данные о текущей кисти можно с помощью функции:

```
int CBrush::GetLogBrush(    // 0 - при ошибке
   LOGBRUSH* pLogBrush);    // Указатель на структуру с данными о кисти
```

Создание штрих-кисти выполняется с помощью функции:

```
BOOL CBrush::CreateHatchBrush(          // 0 - при ошибке
   int nIndex,                          // Стиль штриховки (аналогично
                                        // lbHatch в LOGBRUSH )
   COLORREF crColor);                   // Цвет кисти
```

Кисть на основе битового узора (растрового изображения) создается функцией:

```
BOOL CBrush::CreatePatternBrush(        // 0 - при ошибке
   CBitmap* pBitmap);                   // Указатель на растровое
                                        // изображение
```

Создать растровое изображение можно различными способами. Здесь (см. листинг 2.12) для простоты задается массив с кодами `HatchBits[]` и используется для создания растра:

```
BOOL CBitmap::CreateBitmap(             // 0 - при ошибке
   int nWidth,                          // Ширина растра
   int nHeight,                         // Высота растра
   UINT nPlanes,                        // Количество битовых слоев растра
                                        // (всегда равно единице)
   UINT nBitcount,                    // Количество битов на каждый пиксел
   const void* lpBits);                 // Указатель на битовый узор
                                        // (для заполнения им растра)
```

Можно использовать встроенные кисти:

```
BOOL CGdiObject::CreateStockObject(int nIndex);
```

Встроенные кисти (параметр `nIndex`) могут быть следующими:

- `BLACK_BRUSH` — черная кисть;
- `DKGRAY_BRUSH` — темно-серая;
- `DC_BRUSH` — кисть данного устройства по умолчанию (прозрачная);
- `GRAY_BRUSH` — серая;
- `LTGRAY_BRUSH` — светло-серая;
- `NULL_BRUSH` — нулевая (прозрачная);
- `WHITE_BRUSH` — белая.

Пример использования функции `CreateStockObject()`:

```
newBrush.CreateStockObject(DC_BRUSH);
dc.SelectObject(&newBrush);
```

Для рисования фигур и областей в листинге 2.12 использовались следующие функции:

1. Заполнение прямоугольной области:

```
void CDC::FillRect(
   LPCRECT lpRect,          // Указатель на прямоугольную об-
ласть
   CBrush* pBrush);         // Указатель на кисть для заполнения
```

2. Эллиптическая дуга:

```
BOOL CDC::Arc(              // 0 - при ошибке
   int x1, int y1,          // Левый верхний угол прямоугольника
   int x2, int y2,          // Правый нижний угол
   int x3, int y3,          // Координаты начала дуги
   int x4, int y4);         // Координаты конца дуги
```

или

```
BOOL CDC::Arc(
   LPCRECT lpRect,          // Координаты прямоугольника
   POINT ptStart,           // Координаты начала дуги
   POINT ptEnd);            // Координаты конца дуги
```

Дуга рисуется против часовой стрелки, начинается в точке пересечения эллипса с линией, проходящей через центр прямоугольника и `ptStart`, и заканчивается в точке пересечения эллипса с линией, проходящей через центр прямоугольника и `ptEnd` (рис. 2.5).

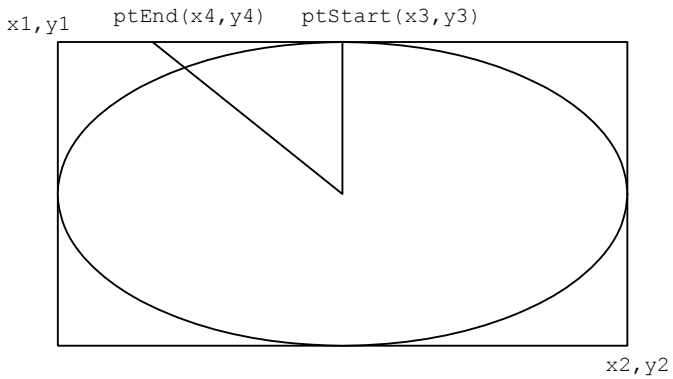


Рис. 2.5. Рисование эллиптической дуги

3. Дуга с хордой

```
BOOL CDC::Chord(
   int x1, int y1,          // Левый верхний угол прямоугольника
   int x2, int y2,          // Левый верхний угол прямоугольника
   int x3, int y3,          // Координаты начала дуги
   int x4, int y4);         // Координаты начала дуги
```

или

```
BOOL CDC::Chord(
   LPCRECT lpRect,          // Координаты прямоугольника
   POINT ptStart,           // Координаты начала дуги
   POINT ptEnd);            // Координаты начала дуги
```

4. Эллипс:

```
BOOL CDC::Ellipse(          // 0 - при ошибке
   int x1, int y1,          // Левый верхний угол прямоугольника
   int x2, int y2);         // Правый нижний угол
```

или

```
BOOL CDC::Ellipse(
   LPCRECT lpRect);         // Правый нижний угол
```

5. Сектор эллипса:

```
BOOL CDC::Pie(              // 0 - при ошибке
   int x1, int y1,          // Левый верхний угол прямоугольника
   int x2, int y2,          // Правый нижний угол
   int x3, int y3,          // Координаты начала дуги
   int x4, int y4);         // Координаты конца дуги
```

или

```
BOOL CDC::Pie(
   LPCRECT lpRect,          // Координаты прямоугольника
   POINT ptStart,           // Координаты начала дуги
   POINT ptEnd);            // Координаты конца дуги
```

6. Прямоугольник (рисуется текущим пером и заполняется текущей кистью):

```
BOOL CDC::Rectangle(        // 0 - при ошибке
   int x1, int y1,          // Левый верхний угол прямоугольника
   int x2, int y2);         // Правый нижний угол
```

или

```
BOOL CDC::Rectangle(
   LPCRECT      lpRect);    //      Координаты      прямоугольника
(x1,y1,x2,y2)
```

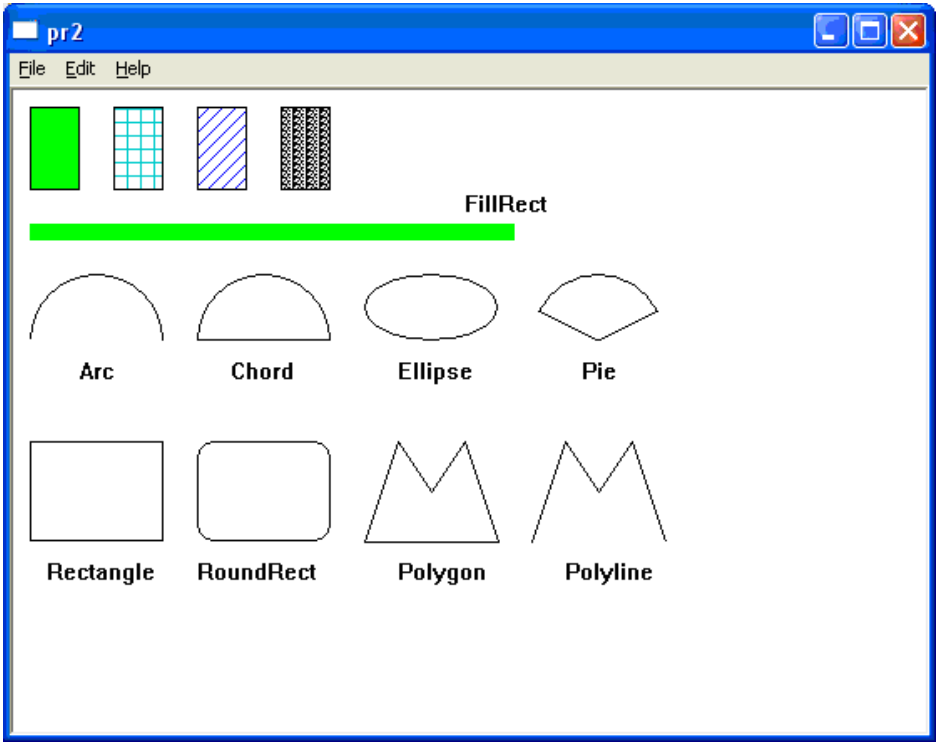


Рис. 2.6. Результаты работы с кистью

7. Прямоугольник с закругленными углами:

```
BOOL CDC::RoundRect(          // 0 - при ошибке
   int x1,                    // Левый верхний угол прямоугольника
   int y1,
   int x2,                    // Правый нижний угол
   int y2,
   int x3,              // Закругление углов. x3 - ширина,
   int y3);             // y3 - высота эллипса, определяющего дуги
```

или

```
BOOL CDC::RoundRect(
   LPCRECT lpRect,          // Координаты прямоугольника
   POINT point);            // Координаты для эллипса закругления
```

8. Многоугольник (последняя точка соединяется с первой):

```
BOOL CDC::Polygon(         // 0 - при ошибке
   LPPOINT lpPoints,    // Массив координат (x,y) точек многоугольни-
ка
   int nCount);         // Количество точек
                        // (2 <= nCount <= размер массива)
```

9. Кривая:

```
BOOL CDC::Polyline(        // 0 - при ошибке
   LPPOINT lpPoints,       // Массив координат (x,y) точек
   int nCount);            // Количество точек
```

Результаты работы с кистью показаны на рис. 2.6.

ГЛАВА 3

Картинки, кнопки и курсоры в окне представления

Создадим проект, на примере которого рассмотрим возможности использования управляющих элементов в окне представления.

Создайте проект **pr3** аналогично проекту **pr1** *(см. разд. 1.1)*. При этом надо изменить следующие опции относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ (установить переключатель в положение **Single document**);
  - без поддержки архитектуры документ/представление (снять флажок **Document/View architecture support**);
- на вкладке **User Interface Features**:
  - без строки статуса (снять флажок **Initial status bar**);
  - без панели инструментов (установить переключатель **Toolbars** в положение **None**).

3.1. Описание программы

3.1.1. Добавление кнопок в класс окна представления

В объявление класса окна представления добавим данные для работы с кнопками. Изменения в файле приведены в листинге 3.1.

Листинг 3.1. Изменения в файле ChildView.h для работы с кнопками

```
// ...
class CChildView : public CWnd
{
// ...
public:
   // Две кнопки
   CButton But1;
   CButton But2;
};
```

Для работы с кнопками существует класс:

```
class CButton:public CWnd
```

В ресурс таблицы строк добавим два новых идентификатора для кнопок: `IDB_BUT1` и `IDB_BUT2` (рис. 3.1).

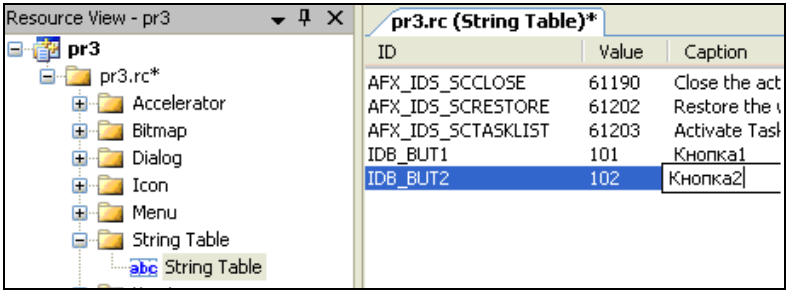


Рис. 3.1. Добавление идентификаторов для кнопок

Примечание

Идентификаторы для кнопок можно было добавить и непосредственно в файл Resource.h (`#define IDB_BUT1 101`). Но лучше это делать через таблицу строк, чтобы не попасть в значения предопределенных идентификаторов.

Кнопки будут создаваться в окне представления, поэтому они будут дочерними по отношению к нему, и обработка их нажатия будет происходить в

определении класса представления. Окно представления создается в `CMainFrame::OnCreate()`. Сразу после создания окна представления создаем в нем две кнопки. Изменения в файле приведены в листинге 3.2.

Листинг 3.2. Изменения в файле MainFrm.cpp для работы с кнопками

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if(CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
   // create a view to occupy the client area of the frame
   if(!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, CRect(0,0,0,0),
                        this, AFX_IDW_PANE_FIRST, NULL))
   {
      TRACE0("Failed to create view window\n");
      return -1;
   }
   m_wndView.But1.Create(L"Кнопка 1", WS_CHILD | WS_VISIBLE |
                         BS_PUSHBUTTON, CRect(640,10,730,30),
                         &m_wndView, IDB_BUT1);
   m_wndView.But2.Create(L"Кнопка 2", WS_CHILD | WS_VISIBLE |
                         BS_PUSHBUTTON, CRect( 640,50,730,70),
                         &m_wndView, IDB_BUT2);
   return 0;
}
```

Функция создания кнопки определена как:

```
virtual BOOL CButton::Create( // 0 - ошибка
   LPCTSTR lpszCaption,       // Текст на кнопке
   DWORD dwStyle,             // Стиль кнопки
   const RECT& rect,          // Расположение и размер кнопки
   CWnd* pParentWnd,          // Указатель на родительское окно (всегда
                              // должен быть определен, т. е. не может
                              // быть равен NULL)
   UINT nID);                 // Идентификатор кнопки
```

Стили кнопок могут быть следующими:

- BS_CHECKBOX — маленький квадратик с текстом (по умолчанию справа), так называемый "флажок". Может иметь два состояния (помечен "галочкой" или нет — пустой квадратик);
- BS_RADIOBUTTON — небольшой кружок с текстом (по умолчанию справа), так называемый "переключатель". Может иметь два состояния (помечен или нет — пустой кружок);
- BS_3STATE — то же, что и BS_CHECKBOX, но имеет три состояния (выбран, не выбран, не определен (серый));
- BS_AUTO3STATE — то же, что и BS_3STATE, но смена состояний производится автоматически;
- BS_AUTOCHECKBOX — то же, что и BS_CHECKBOX, но смена состояний производится автоматически;
- BS_AUTORADIOBUTTON — то же, что и BS_RADIOBUTTON, но кнопка автоматически помечается при выборе пользователем, а с остальных кнопок группы отметка снимается;
- BS_BITMAP — кнопка с битовым изображением;
- BS_BOTTOM — выравнивание текста или изображения по нижнему краю кнопки;
- BS_CENTER — горизонтальное выравнивание текста или изображения по центру кнопки;
- BS_DEFPUSHBUTTON — кнопка, выделенная жирной рамкой (кнопка по умолчанию). При нажатии на клавишу <Enter> срабатывает именно эта кнопка;
- BS_FLAT — плоская кнопка без трехмерной рамки;
- BS_GROUPBOX — прямоугольная рамка, объединяющая другие кнопки. В левом верхнем углу рамки может быть текст;
- BS_ICON — кнопка со значком;
- BS_LEFT — выравнивание текста или изображения по левому краю кнопки;
- BS_LEFTTEXT — комбинируется с BS_CHECKBOX и BS_RADIOBUTTON для расположения текста слева от кнопок;
- BS_MULTILINE — текст на кнопке может располагаться в несколько строк;

- `BS_PUSHBUTTON` — простая кнопка, срабатывающая при нажатии на нее (она посылает сообщение `WM_COMMAND`);
- `BS_PUSHLIKE` — кнопка изображается так же, как обычная кнопка `BS_PUSHBUTTON` (для флажков `BS_CHECKBOX` или переключателей `BS_RADIOBUTTON`);
- `BS_RIGHT` — выравнивание текста или изображения по правому краю кнопки;
- `BS_RIGHTBUTTON` — аналогично `BS_LEFTTEXT`;
- `BS_TEXT` — кнопка с текстом;
- `BS_TOP` — выравнивание текста или изображения по верхнему краю кнопки;
- `BS_VCENTER` — вертикальное выравнивание текста или изображения по центру кнопки.

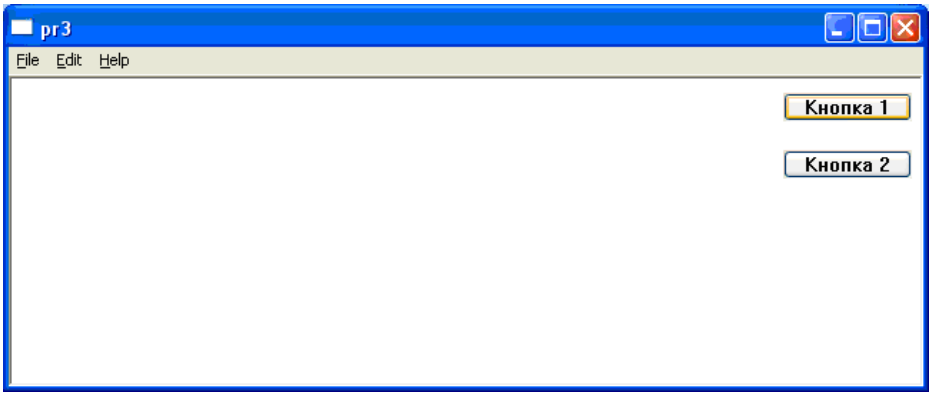


Рис. 3.2. Создание кнопок в окне представления

Дополнительно надо указывать следующие стили окна (т. к. кнопка тоже является окном):

- `WS_CHILD` — надо указывать *всегда* при создании кнопки;
- `WS_VISIBLE` — если кнопка должна быть изначально видимой;
- `WS_DISABLED` — если кнопка должна быть изначально недоступной;
- `WS_GROUP` — для объединения нескольких кнопок в группу. У первой кнопки группы указывается *обязательно*;

- WS_TABSTOP — если кнопка должна входить в группу кнопок, которым по очереди передается фокус ввода при нажатии клавиши <Tab>.

Для кнопок можно использовать обработку следующих сообщений:

- ON_BN_CLICKED — нажали кнопку;
- ON_BN_DISABLE — кнопка заблокирована;
- ON_BN_DOUBLECLICKED — двойной щелчок по кнопке.

Результат выполнения программы показан на рис. 3.2.

Кнопки в окне появились, но если на них нажимать, ничего не происходит. Надо добавить обработку сообщений от кнопок. Изменения в файлах приведены в листингах 3.3 и 3.4.

Листинг 3.3. Изменения в файле ChildView.h для работы с кнопками

```
// ...
class CChildView : public CWnd
{
// ...
protected:
   afx_msg void OnPaint();
   // Обработка нажатия кнопок в окне
   afx_msg void OnButClicket1();        // Кнопка 1
   afx_msg void OnButClicket2();        // Кнопка 2
   DECLARE_MESSAGE_MAP()
};
```

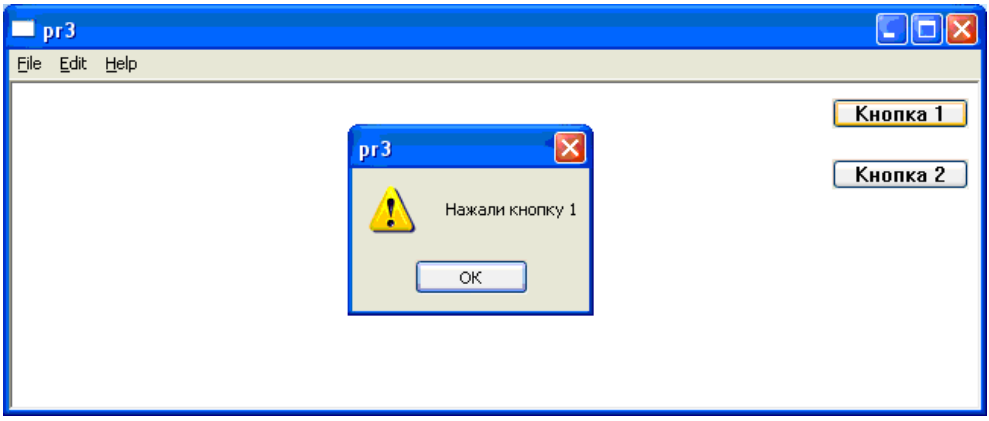
Листинг 3.4. Изменения в файле ChildView.cpp для работы с кнопками

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_BN_CLICKED(IDB_BUT1, OnButClicket1)
   ON_BN_CLICKED(IDB_BUT2, OnButClicket2)
END_MESSAGE_MAP()
// ...
void CChildView::OnPaint()
```

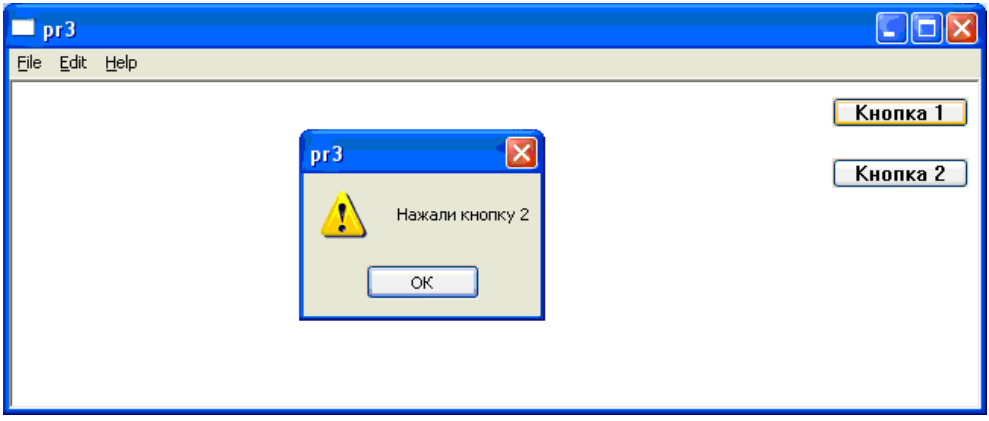
```
{
   CPaintDC dc(this);
}
// Обработка нажатия кнопок в окне
void CChildView::OnButClicket1()
{
   AfxMessageBox(L"Нажали кнопку 1");
}
void CChildView::OnButClicket2()
{
   AfxMessageBox(L"Нажали кнопку 2");
}
```

Иногда обработка сообщений бывает почти одинаковой. Тогда можно воспользоваться одной универсальной функцией. Для нашей программы это может выглядеть так:

```
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_BN_CLICKED(IDB_BUT1, OnButClicket)
   ON_BN_CLICKED(IDB_BUT2, OnButClicket)
END_MESSAGE_MAP()
// ...
void CChildView::OnButClicket()
{
   CString msg("Нажали кнопку ");
   MSG *pMsg = (MSG *)GetCurrentMessage(); // Получить текущее сообщение
   ASSERT(pMsg);                           // Проверить на ошибку
   switch(pMsg->wParam)                    // Проверить, от кого сообщение
   {
      case IDB_BUT1: msg += "1"; break;    // От кнопки 1
      case IDB_BUT2: msg += "2"; break;    // От кнопки 2
   }
   AfxMessageBox(msg);
}
```



а



б

Рис. 3.3. Обработка нажатия кнопки 1 (*а*) и кнопки 2 (*б*)

Когда сообщение выбирается из очереди, оно помещается в специальную структуру `MSG`:

```
typedef struct tagMSG
{
   HWND hwnd;          // Дескриптор окна, которому предназначено сообщение
   UINT message;       // Номер сообщения
   WPARAM wParam;      // wParam, lParam - дополнительная информация
   LPARAM lParam;      // о сообщении (зависит от типа сообщения)
   DWORD time;         // Время в момент постановки сообщения в очередь
```

```
   POINT pt;          // Позиция курсора мыши в момент поступления
                      // сообщения
} MSG;
```

Получить текущее сообщение можно с помощью функции `GetCurrentMessage()`, которая возвращает указатель на структуру `MSG`:

```
static const MSG* PASCAL CWnd::GetCurrentMessage();
```

Для проверки ошибок существует макрос `ASSERT`. При значении выражения `booleanExpression = FALSE` выполнение программы прерывается и печатается сообщение об ошибке.

```
ASSERT(booleanExpression);
```

В процессе работы программы можно изменить стиль кнопки:

```
void CButton::SetButtonStyle(
   UINT nStyle,               // Стиль кнопки
   BOOL bRedraw = TRUE);      // Перерисовать кнопку сразу
```

Результат работы программы показан на рис. 3.3.

Хотелось бы, чтобы эти кнопки работали взаимосвязанно (после нажатия кнопки 1 можно нажимать только кнопку 2, и наоборот). Для этого надо изменить стиль второй кнопки при создании (сделать ее недоступной), а в обработке сообщений от кнопок вставить блокировку и разблокировку соответствующей кнопки. Изменения в файлах приведены в листингах 3.5 и 3.6.

Листинг 3.5. Изменения в файле MainFrm.cpp для синхронной работы с кнопками

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // ...
   m_wndView.But1.Create(L"Кнопка 1", WS_CHILD | WS_VISIBLE |
                         BS_PUSHBUTTON, CRect(640,10,730,30),
                         &m_wndView, IDB_BUT1);
   m_wndView.But2.Create(L"Кнопка 2", WS_CHILD | WS_VISIBLE |
                         BS_PUSHBUTTON | WS_DISABLED,
                         CRect(640,50,730,70), &m_wndView, IDB_BUT2);
   return 0;
}
```

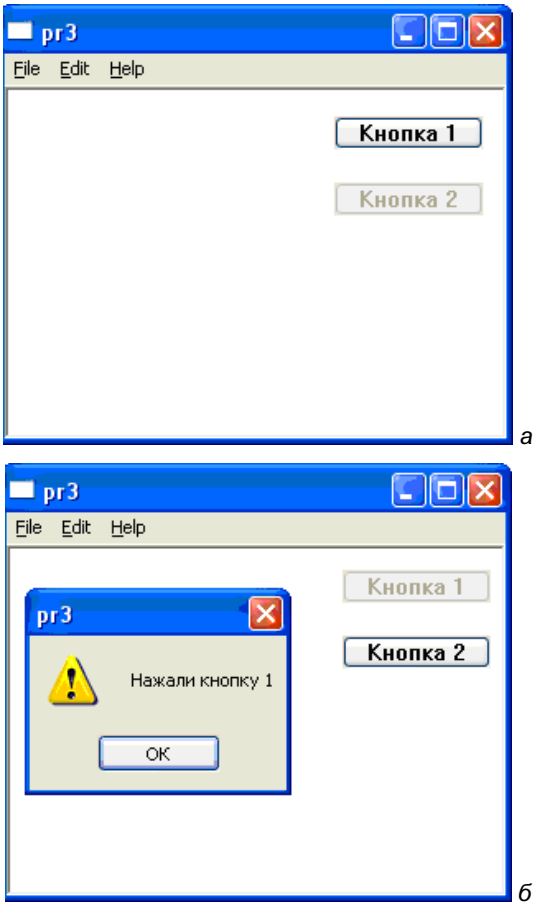


Рис. 3.4. Результат синхронной работы кнопок: при запуске приложения (*а*) и при нажатии кнопки 1 (*б*)

Листинг 3.6. Изменения в файле ChildView.cpp для синхронной работы с кнопками

```
// ...
// Обработка нажатия кнопок в окне
void CChildView::OnButClicket1()
{
   But1.EnableWindow(FALSE);        // Заблокировать кнопку 1
   But2.EnableWindow(TRUE);         // Разблокировать кнопку 2
```

```
   AfxMessageBox(L"Нажали кнопку 1");
}
void CChildView::OnButClicket2()
{
   But2.EnableWindow(FALSE);          // Заблокировать кнопку 2
   But1.EnableWindow(TRUE);           // Разблокировать кнопку 1
   AfxMessageBox(L"Нажали кнопку 2");
}
```

Для изменения доступа к окну существует функция:

```
BOOL CWnd::EnableWindow(
   BOOL bEnable = TRUE);            // Разрешен, false - запрещен
```

Можно не только делать кнопку неактивной, но и совсем скрывать ее с помощью функции `CWnd::ShowWindow()`, например:

```
   But1.ShowWindow(SW_HIDE);          // Скрыть кнопку 1
   But1.ShowWindow(SW_SHOWNORMAL);    // Показать кнопку 1
```

Результат синхронной работы кнопок показан на рис. 3.4.

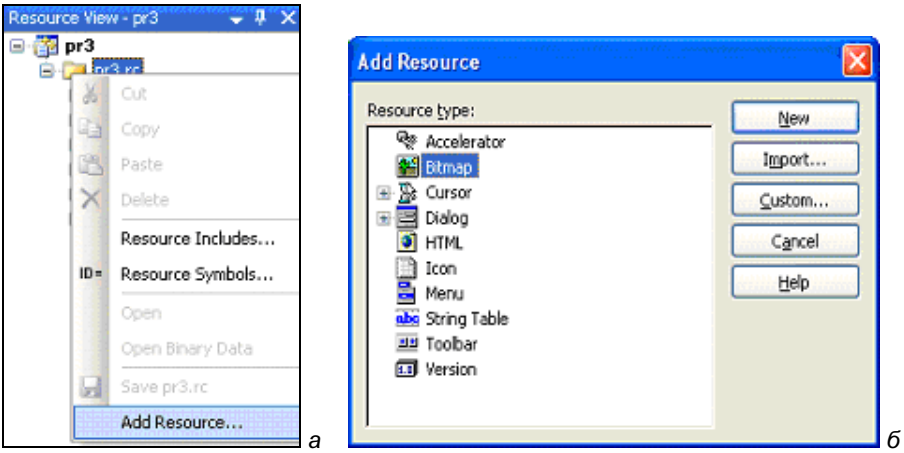
3.1.2. Добавление битового рисунка в класс окна представления

Добавим в ресурс приложения битовый рисунок. Для этого надо вызвать контекстное меню (щелкнув правой кнопкой мыши по файлу ресурсов pr3.rc в окне **Resource View**) и выбрать команду **Add Resource** (Добавить ресурс). В появившемся окне **Add Resource** выбрать тип ресурса **Bitmap** (Битовый рисунок) и нажать кнопку **New** (Создать) (рис. 3.5).

В окне свойств рисунка выберем нужное количество цветов (**Colors**: 16). Можно выбрать размер рисунка: **Height** (Высота) = 48, **Width** (Ширина) = 48 (рис. 3.6).

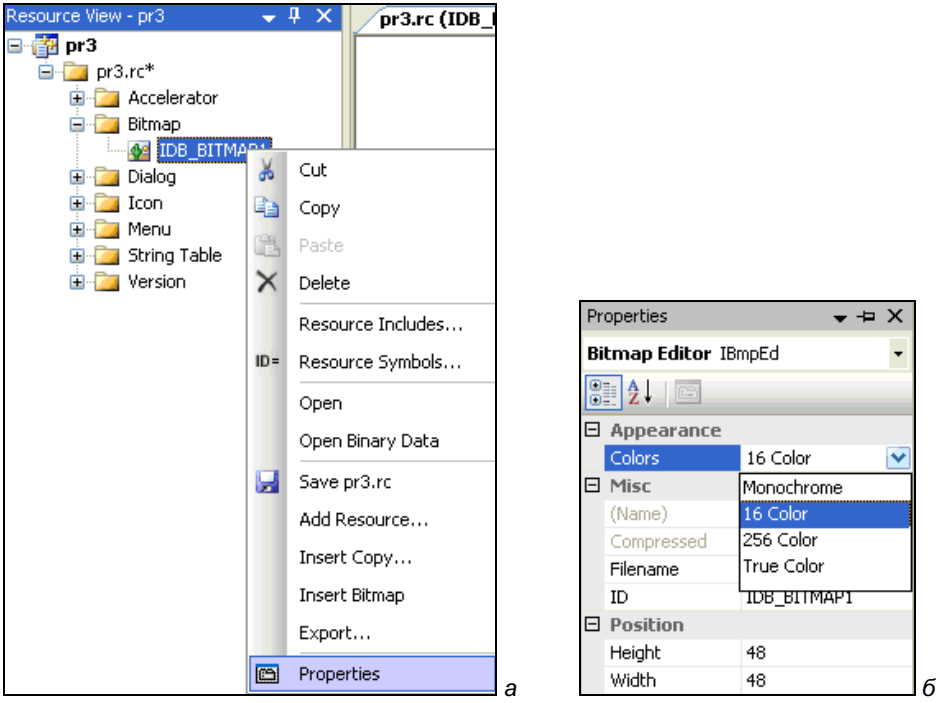
Нарисуем произвольное изображение (рис. 3.7).

Сделаем так, чтобы при нажатии кнопки 1 (в окне представления) рисунок появлялся, а при нажатии кнопки 2 — исчезал. Изменения в файлах приведены в листингах 3.7—3.9.



а *б*

Рис. 3.5. Добавление нового ресурса в приложение: контекстное меню файла ресурсов (*а*) и окно выбора типа добавляемого ресурса **Add Resource** (*б*)



а *б*

Рис. 3.6. Открытие свойств рисунка с помощью контекстного меню (*а*) и изменение свойств битового рисунка (*б*)

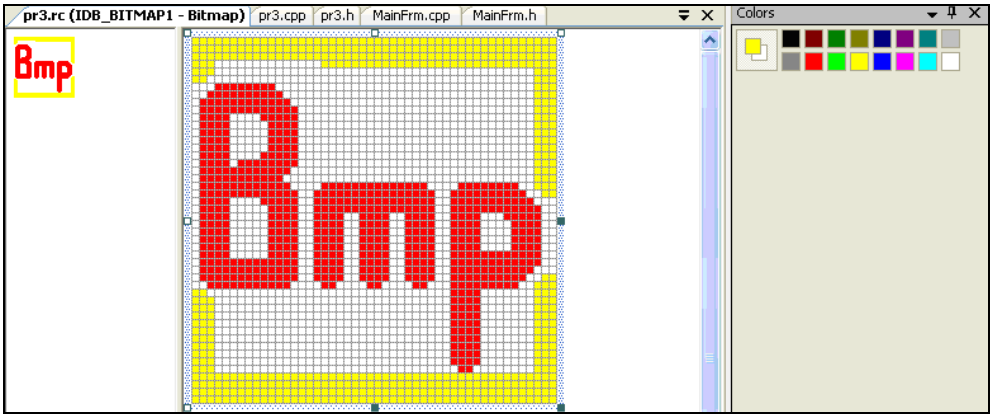


Рис. 3.7. Добавляемый рисунок

Листинг 3.7. Изменения в файле ChildView.h для работы с битовым рисунком

```
// ...
class CChildView : public CWnd
{
   // ...
// Attributes
public:
   // Две кнопки
   CButton But1;
   CButton But2;
   CBitmap bmp;              // Битовый рисунок
   CDC dc_bmp;               // Контекст битового рисунка
   bool fbut;                // Флаг: 1 - показать рисунок, 0 - нет
   void setbmp(void);        // Подключение битового рисунка
   // ...
};
```

Листинг 3.8. Изменения в файле MainFrm.cpp для работы с битовым рисунком

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
   // ...
   m_wndView.But2.Create(L"Кнопка 2",WS_CHILD | WS_VISIBLE |
                         BS_PUSHBUTTON | WS_DISABLED,
                         CRect(540,50,630,70), &m_wndView, IDB_BUT2 ;
   m_wndView.setbmp();                  // Подключение битового рисунка
   return 0;
}
```

Листинг 3.9. ChildView.cpp для работы с битовым рисунком

```
// ...
CChildView::CChildView()
{
   fbut = false;
}
CChildView::~CChildView()
{
   dc_bmp.DeleteDC();
}
// Подключение битового рисунка
void CChildView::setbmp(void)
{
   CDC *dc;                          // Контекст устройства окна
   dc = GetDC();                     // Получаем контекста устройства окна
   dc_bmp.CreateCompatibleDC(dc);    // Строим растр, совместимый с окном
   bmp.LoadBitmap(IDB_BITMAP1);      // Загружаем битовый рисунок
   dc_bmp.SelectObject(&bmp);        // Подключаем рисунок
   ReleaseDC(dc);                    // Освобождаем контекст устройства
}

void CChildView::OnPaint()
{
   CPaintDC dc(this);                // device context for painting
   if(!fbut)
      return;
```

```
   // Копирование рисунка из dc_bmp в dc (из контекста рисунка в контекст
   // окна)
   dc.BitBlt(20, 20, 48, 48, &dc_bmp, 0,0, SRCCOPY);
}

// Обработка нажатия кнопок в окне
void CChildView::OnButClicket1()
{
   But1.EnableWindow(FALSE);        // Заблокировать кнопку 1
   But2.EnableWindow(TRUE);         // Разблокировать кнопку 2
   // AfxMessageBox( L"Нажали кнопку 1" );
   fbut = true;                     // Можно рисовать
   Invalidate();                    // Перерисовка окна представления
}
void CChildView::OnButClicket2()
{
   But2.EnableWindow(FALSE);        // Заблокировать кнопку 2
   But1.EnableWindow(TRUE);         // Разблокировать кнопку 1
   //AfxMessageBox( L"Нажали кнопку 2" );
   fbut = false;                    // Нельзя рисовать
   Invalidate();                    // Перерисовать окно
}
```

Для работы с битовыми рисунками существует класс:

```
class CBitmap : public CGdiObject
```

Связь между программой и рисунком устанавливается через контекст устройства (в данном случае контекст рисунка `dc_bmp`). Для работы с контекстами есть классы:

```
class CDC:public CObject
class CPaintDC:public CDC
```

Битовый рисунок должен появляться при нажатии кнопки 1 и исчезать при нажатии кнопки 2. Для этого используются значения флага `fbut`.

Получение контекста устройства выполняется функцией:

```
CDC* CWnd::GetDC();
```

Функция возвращает указатель на контекст устройства или 0 при ошибке. Указатель может быть временным, не должен сохраняться для дальнейшего использования и *должен освобождаться* после использования (с помощью функции `ReleaseDC()`).

Функция `CreateCompatibleDC()` создает в памяти растровое изображение (контекст устройства в памяти), совместимое с заданным контекстом устройства (в данном случаее окном представления). Если в качестве аргумента задать `NULL`, то создастся растр, совместимый с экраном дисплея. Функция определена как:

```
BOOL CDC::CreateCompatibleDC(   // 0 - ошибка
   CDC* pDC);                   // Контекст устройства, с которым должно
                                // быть совместимо растровое изображение
```

Загрузка битового рисунка выполняется с помощью фукнции:

```
BOOL CBitmap::LoadBitmap(       // 0 - ошибка
   LPCTSTR lpszResourceName);   // Битовый рисунок, заданный текстовой
                                // строкой
```

или

```
BOOL CBitmap::LoadBitmap(
   UINT nIDResource);        // Битовый рисунок, заданный идентификатором
```

Можно загрузить стандартные рисунки (их идентификаторы определены в файле Microcoft Visual Studio 8\Vc\ce\altmfc\include\ResDefCE.h, который надо подключить):

```
BOOL CBitmap::LoadOEMBitmap(    // 0 - ошибка
   UINT nIDBitmap);             // Идентификатор ресурса
```

Стандартные битовые ресурсы могут быть следующих видов:

-  — `OBM_BTNCORNERS`;
-  — `OBM_BTSIZE`;
-  — `OBM_SIZE`;
-  — `OBM_CHECK`;
-  — `OBM_CHECKBOXES`;
-  — `OBM_COMBO`;

-  — OBM_MNARROW;
-  — OBM_DNARROW;
-  — OBM_DNARROWD (OBM_DNARROW нажатая);
-  — OBM_DNARROWI (OBM_DNARROW заблокированная);
-  — OBM_LFARROW;
- OBM_LFARROWD (нажатая OBM_LFARROW);
- OBM_LFARROWI (заблокированная OBM_LFARROW);
-  — OBM_RGARROW;
- OBM_RGARROWD (нажатая OBM_RGARROW);
- OBM_RGARROWI (заблокированная OBM_RGARROW);
-  — OBM_UPARROW;
- OBM_UPARROWD (нажатая OBM_UPARROW);
- OBM_UPARROWI (заблокированная OBM_UPARROW);
-  — OBM_CLOSE;
-  — OBM_REDUCE;
-  — OBM_REDUCED (нажатая OBM_REDUCE);
-  — OBM_RESTORE;
-  — OBM_RESTORED (OBM_RESTORE нажатая);
-  — OBM_ZOOM;
-  — OBM_ZOOMD (OBM_ZOOM нажатая).

Прежде чем растровое изображение будет выведено на экран, его надо выбрать в текущем контексте устройства с помощью функции SelectObject():

```
CBitmap* CDC::SelectObject(  // Указатель на старое (текущее) изображение
   CBitmap* pBitmap);        // Указатель на новое изображение
```

После этого надо освободить контекст устройства (количество контекстов устройства в Windows ограничено, и если их не освобождать, то потом может не хватить):

```
int CWnd::ReleaseDC(          // 0 - ошибка
   CDC* pDC);                 // Указатель на контекст устройства,
                              // который надо освободить
```

Растровое изображение является ресурсом, который должен быть освобожден перед завершением программы (создание изображения выполняется функцией `CreateCompatibleDC()`, удаление — `DeleteDC()`). Освобождение ресурса выполняется при разрушении окна представления (в деструкторе):

```
BOOL CDC::DeleteDC();         // 0 - ошибка
```

Чтобы вывести изображение на экран, надо использовать функцию копирования битового изображения из одного контекста в другой. 20, 20 — координаты начала вывода рисунка, 48, 48 — ширина и высота рисунка (задавались при создании рисунка, см. рис. 3.6):

```
   dc.BitBlt(20, 20, 48, 48, &dc_bmp, 0, 0, SRCCOPY);
```

Функция копирования одного контекста в другой объявлена как:

```
BOOL CDC::BitBlt(   // 0 - ошибка
   int x,           // Координаты начала прямоугольной области,
   int y,           // куда копируем
   int nWidth,      // Ширина области
   int nHeight,     // Высота области
   CDC* pSrcDC,     // Контекст, из которого копируем
   int xSrc,       // Координаты левого верхнего угла контекста-источника
   int ySrc,       // (контекста битового рисунка), обычно равны 0
   DWORD dwRop);    // Растровая операция, используемая при копировании
```

Для операций, не требующих битового массива в качестве источника, параметр `pSrcDC` должен быть равен `NULL`.

Растровые операции (значение `dwRop`) могут быть такими:

- `BLACKNESS` — заданная область закрашивается сплошным черным цветом;
- `DSTINVERT` — инвертирование текущего заполнения;
- `MERGECOPY` — комбинирование текущего заполнения с текущей кистью с помощью оператора `AND`;

- MERGEPAINT — комбинирование инвертированного битового массива с текущим заполнением с помощью оператора OR;
- NOTSRCCOPY — копирование инвертированного битового массива;
- NOTSRCERASE — инвертирование результата комбинирования текущего заполнения с битовым массивом с помощью оператора OR;
- PATCOPY — заполнение области при помощи текущей кисти;
- PATINVERT — комбинирование текущего заполнения и кисти с помощью оператора XOR;
- PATPAINT — копирование инвертированного битового массива с текущей кистью с помощью оператора OR и комбинирование результата этой операции с текущим заполнением с помощью оператора OR;
- SRCAND — комбинирование текущего заполнения и битового массива с помощью оператора AND;
- SRCCOPY — заполнение заданной области битовым массивом;
- SRCERASE — комбинирование битового массива и инвертированного текущего заполнения с помощью оператора AND;
- SRCINVERT — комбинирование битового массива и текущего заполнения с помощью оператора XOR;
- SRCPAINT — комбинирование битового массива и текущего заполнения с помощью оператора OR;
- WHITENESS — заданная область закрашивается сплошным белым цветом.

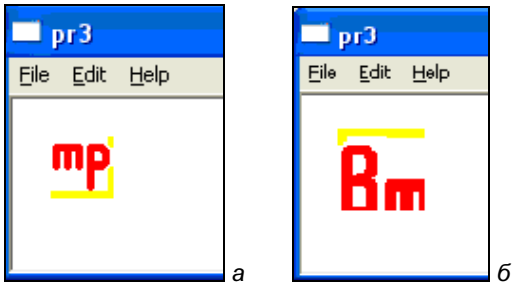
Если в качестве координат левого верхнего угла (xSrc, ySrc) взять не 0 (например, dc.BitBlt(20, 20, 48, 48, &dc_bmp, **15, 15,** SRCCOPY)), то выводимый рисунок будет отсечен от левого верхнего угла до этих координат (рис. 3.8, *а*). Если задать размеры области nWidth и nHeight меньше размеров Width и Height рисунка, то он вообще не отобразится. Если надо отсечь рисунок от правого нижнего угла, существует функция копирования одного контекста в другой с изменением размеров:

```
BOOL CDC::StretchBlt(        // см. CDC::BitBlt()
   int x,
   int y,
   int nWidth,
   int nHeight,
   CDC* pSrcDC,
```

```
    int xSrc,
    int ySrc,
    int nSrcWidth,              // Размеры – ширина выводимой части рисунка
    int nSrcHeight,             // Высота
    DWORD dwRop);
```

Пример использования функции `StretchBlt()` показан на рис. 3.8, *б*:

```
dc.StretchBlt(20, 20, 48, 48, &dc_bmp, 0, 0, 30, 30, SRCCOPY);
```



а *б*

Рис. 3.8. Усечение рисунка от левого верхнего (*а*) и от правого нижнего (*б*) угла изображения

Во время работы программы при нажатии кнопки 1 вызывается обработчик `CChildView::OnButClicket1()`. В нем блокируется кнопка 1, разблокируется кнопка 2. Далее устанавливается флаг для рисования, и с помощью `Invalidate()` окну представления посылается сообщение о перерисовке (`CChildView::OnPaint()`). Функция перерисовки окна определена как:

```
void CWnd::Invalidate(
    BOOL bErase = TRUE);
```

Флаг `bErase` указывает, надо ли стирать предыдущее наполнение окна (`TRUE` — надо, `FALSE` — нет).

Если надо перерисовать не все окно, а его часть, то надо использовать функцию перерисовки заданной области окна:

```
void CWnd::InvalidateRect(
    LPCRECT lpRect,             // Указатель на CRect с координатами
                                // области перерисовки
    BOOL bErase = TRUE);
```

Результаты работы программы показаны на рис. 3.9.

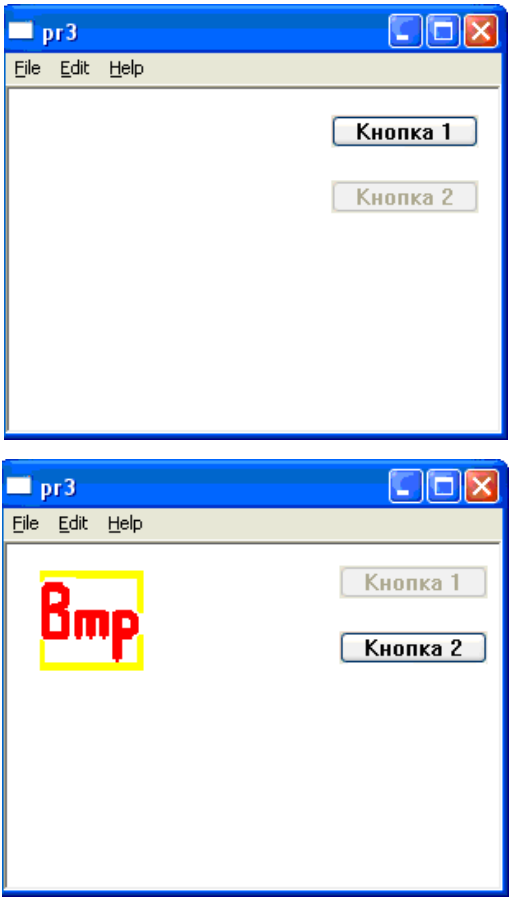


Рис. 3.9. Результат работы программы с битовым рисунком: при запуске приложения (*а*) и при нажатии кнопки 1 (*б*)

3.1.3. Добавление готовых ресурсов в приложение

Добавление рисунка

Добавить готовое изображение можно следующим образом: скопировать нужный файл (например, b.bmp) в каталог проекта (лучше в каталог pr3\res). Добавить новый ресурс (см. рис. 3.5, *а*), но в окне **Add Resource** нажать не

кнопку **New**, а кнопку **Import** (Импортировать) (см. рис. 3.5, *б*). В появившемся окне **Import** выбрать файл b.bmp (рис. 3.10). В файле ресурсов приложения появится новый ресурс `IDB_BITMAP2` — рисунок размером 500 × 300 (рис. 3.11).

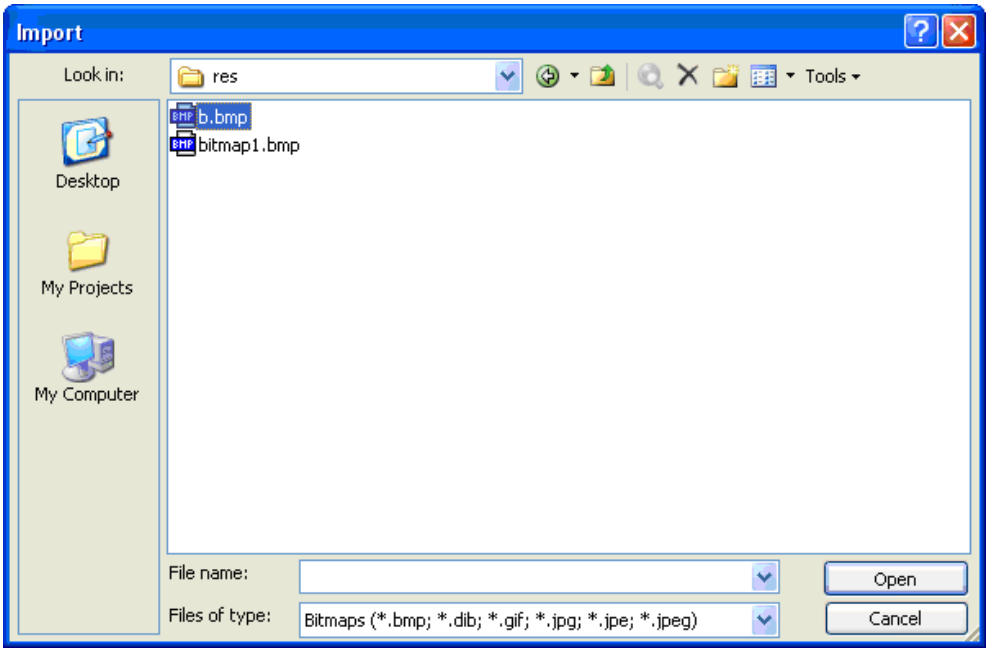


Рис. 3.10. Выбор файла с рисунком

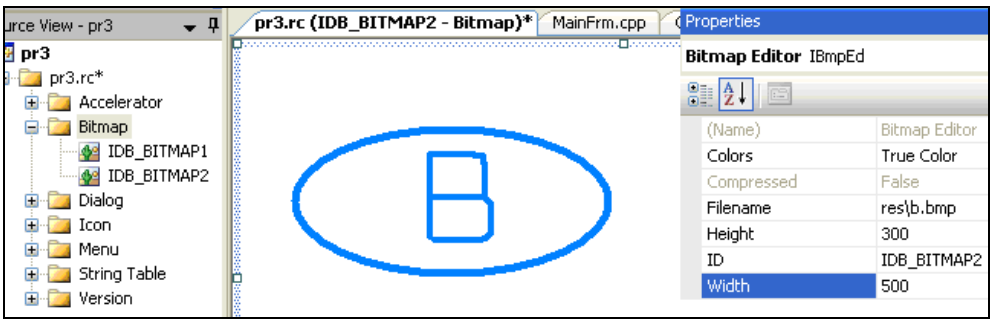


Рис. 3.11. Подключение готового рисунка к ресурсам приложения

В программе надо будет внести изменения, приведенные в листинге 3.10.

Листинг 3.10. Изменения в файле ChildView.cpp для работы с заранее подготовленным битовым рисунком

```
// ...
void CChildView::setbmp(void)
{
   CDC *dc;                          // Контекст устройства окна
   dc = GetDC();                     // Получаем контекст устройства окна
   dc_bmp.CreateCompatibleDC(dc);    // Строим растр, совместимый с окном
   bmp.LoadBitmap(IDB_BITMAP2);      // Загружаем битовый рисунок
   dc_bmp.SelectObject(&bmp);        // Подключаем рисунок
   ReleaseDC(dc);                    // Освобождаем контекст устройства
}
void CChildView::OnPaint()
{
   CPaintDC dc(this);                // device context for painting
   if(!fbut)
      return;
   // Копирование рисунка из dc_bmp в dc
   dc.BitBlt(20, 20, 500, 300, &dc_bmp, 0, 0, SRCCOPY);
}
```

Добавление рисунка-курсора

Вначале надо скопировать нужный файл (например, wait_rm.cur) в каталог проекта (лучше в каталог pr3\res). Добавить свой курсор можно двумя способами:

1. Загрузить рисунок курсора из файла, указав в программе имя этого файла. Изменения в файле приведены в листинге 3.11.

Листинг 3.11. Изменения в файле ChildView.cpp для загрузки рисунка курсора из файла

```
// ...
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
```

```
   if (!CWnd::PreCreateWindow(cs))
      return FALSE;
   cs.dwExStyle |= WS_EX_CLIENTEDGE;
   cs.style &= ~WS_BORDER;
   cs.lpszClass =  AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
                        LoadCursorFromFile(L"res\\wait_rm.cur"),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return TRUE;
}
```

Загрузка курсора из файла выполняется с помощью функции:

```
HCURSOR LoadCursorFromFile(  // Дескриптор курсора
   LPCTSTR lpFileName)       // Строка с именем файла картинки-курсора
```

2. Добавить новый ресурс (см. рис. 3.5, *а*). Выбрать тип ресурса — **Cursor** (Курсор) и нажать кнопку **Import** (см. рис. 3.5, *б*). В появившемся окне **Import** выбрать файл wait_rm.cur. В ресурсы добавится новый курсор (рис. 3.12). Теперь можно его загрузить. Изменения в файле приведены в листинге 3.12.

Листинг 3.12. Изменения в файле ChildView.cpp для загрузки рисунка курсора из файла с добавлением нового ресурса

```
// ...
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
   if (!CWnd::PreCreateWindow(cs))
      return FALSE;
   cs.dwExStyle |= WS_EX_CLIENTEDGE;
   cs.style &= ~WS_BORDER;
   cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
                        ::LoadCursor(AfxGetInstanceHandle(),
                                     MAKEINTRESOURCE(IDC_CURSOR1)),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return TRUE;
}
```

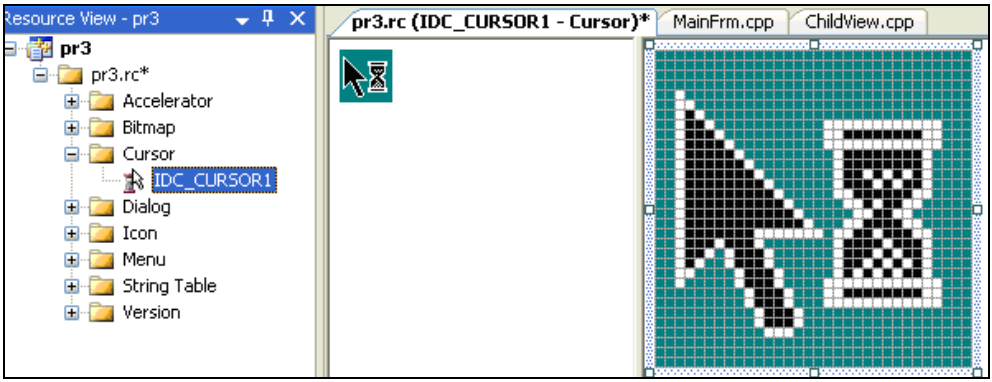


Рис. 3.12. Добавление графического курсора

3.1.4. Изменение формы курсора во время работы

Можно сделать так, чтобы при запуске программы был курсор в виде "руки", при нажатии кнопки 1 он менялся на "песочные часы", а при нажатии кнопки 2 — обратно на "руку". Изменения в файлах приведены в листингах 3.13 и 3.14.

Листинг 3.13. Изменения в файле ChildView.h для динамического изменения формы курсора

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   HCURSOR hCurs1, hCurs2;        // Дескрипторы курсоров
   // ...
};
```

Листинг 3.14. Изменения в файле ChildView.cpp для динамического изменения формы курсора

```
// ...
CChildView::CChildView()
```

```
{
   fbut = false;
   hCurs1 = LoadCursor(NULL, IDC_HAND);  // Загрузить курсор "рука"
   hCurs2 = LoadCursor(NULL, IDC_WAIT);  // Загрузить курсор "часы"
   SetCursor(hCurs1);                    // Сделать текущим курсор "рука"
}

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
   // ...
   cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
                        0 /*::LoadCursor(NULL, IDC_ARROW)*/,
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return TRUE;
}
void CChildView::OnButClicket1()
{
   // ...
   fbut = true;          // Можно рисовать
   SetCursor(hCurs2);    // Сделать текущим курсор "часы"
   Invalidate();         // Перерисовка окна представления
}
void CChildView::OnButClicket2()
{
   // ...
   fbut = false;          // Нельзя рисовать
   SetCursor(hCurs1);     // Сделать текущим курсор "рука"
   Invalidate();          // Перерисовать окно
}
```

Обязательно надо "отменить" загрузку курсора при регистрации класса окна (`LoadCursor(NULL, IDC_ARROW)`), иначе курсоры нельзя будет менять. Результат работы программы показан на рис. 3.13.

Установка нового (уже загруженного) курсора выполняется с помощью функции:

```
HCURSOR SetCursor(          // Дескриптор старого курсора
   HCURSOR hCursor);        // Дескриптор нового курсора
```

Если надо скрыть курсор, используется функция:

```
int ShowCursor(
   BOOL bShow);             // true – показать, false – скрыть курсор
```

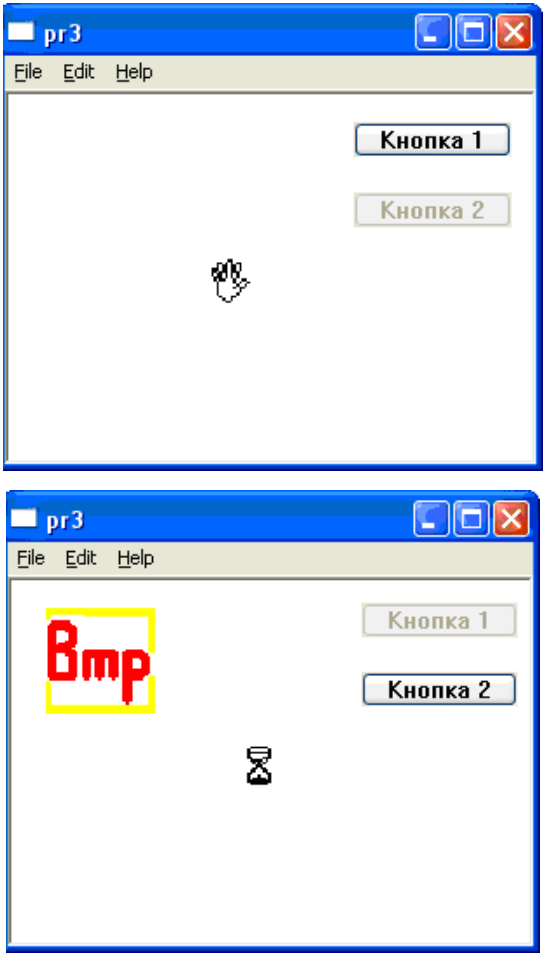


Рис. 3.13. Динамическое изменение формы курсора: при открытии приложения (*а*) и нажатии кнопки 1 (*б*)

3.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 3.15. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDB_BUT1                101
#define IDB_BUT2                102
#define IDR_MAINFRAME           128
#define IDR_pr3TYPE             129
#define IDB_BITMAP1             130
// ...
```

Листинг 3.16. Файл ChildView.h (объявление класса представления)

```
// ...
class CChildView : public CWnd
{
// ...
public:
   // Две кнопки
   CButton But1;
   CButton But2;
   CBitmap bmp;                    // Битовый рисунок
   CDC dc_bmp;                     // Контекст битового рисунка
   bool fbut;                      // Флаг: 1 - показать рисунок, 0 -
нет
   void setbmp(void);              // Подключение битового рисунка
   HCURSOR hCurs1, hCurs2;         // Дескрипторы курсоров
// ...
protected:
   afx_msg void OnPaint();
   // Обработка нажатия кнопок в окне
   afx_msg void OnButClicket1();           // Кнопка 1
```

```
   afx_msg void OnButClicket2();            // Кнопка 2
   DECLARE_MESSAGE_MAP()
};
```

Листинг 3.17. Файл MainFrm.cpp (определение класса фрейма)

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
   // create a view to occupy the client area of the frame
   if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW,
                    CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL))
   {
      TRACE0("Failed to create view window\n");
      return -1;
   }
   // Создание кнопок 1 и 2
   m_wndView.But1.Create(L"Кнопка 1", WS_CHILD | WS_VISIBLE |
                    BS_PUSHBUTTON, CRect(540, 10, 630, 30), &m_wndView,
                    IDB_BUT1);
   m_wndView.But2.Create(L"Кнопка 2", WS_CHILD | WS_VISIBLE |
                    BS_PUSHBUTTON | WS_DISABLED,
                    CRect(540, 50, 630, 70), &m_wndView, IDB_BUT2);
   m_wndView.setbmp();                      // Подключение битового рисунка
   return 0;
}
// ...
```

Листинг 3.18. Файл ChildView.cpp (определение класса представления)

```
// ...
CChildView::CChildView()
{
   fbut = false;
```

```
   hCurs1 = LoadCursor(NULL, IDC_HAND);  // Загрузить курсор "рука"
   hCurs2 = LoadCursor(NULL, IDC_WAIT);  // Загрузить курсор "часы"
   SetCursor(hCurs1);                    // Сделать текущим курсор "рука"
}
CChildView::~CChildView()
{
   dc_bmp.DeleteDC();
}
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_BN_CLICKED(IDB_BUT1, OnButClicket1)
   ON_BN_CLICKED(IDB_BUT2, OnButClicket2)
END_MESSAGE_MAP()
// CChildView message handlers
BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
   if (!CWnd::PreCreateWindow(cs))
      return FALSE;
   cs.dwExStyle |= WS_EX_CLIENTEDGE;
   cs.style &= ~WS_BORDER;
   cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
                  0, reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return TRUE;
}
// Подключение битового рисунка
void CChildView::setbmp( void )
{
   CDC *dc;                          // Контекст устройства окна
   dc = GetDC();                     // Получаем контекст устройства окна
   dc_bmp.CreateCompatibleDC(dc);  // Строим растр, совместимый с окном
   bmp.LoadBitmap(IDB_BITMAP1);      // Загружаем битовый рисунок
   dc_bmp.SelectObject(&bmp);        // Подключаем рисунок
   ReleaseDC(dc);                    // Освобождаем контекст устройства
}
void CChildView::OnPaint()
```

```
{
   CPaintDC dc(this);            // device context for painting
   if(!fbut)
      return;
   // Копирование рисунка из dc_bmp в dc
   dc.BitBlt(20, 20, 48, 48, &dc_bmp, 0, 0, SRCCOPY);
}
// Обработка нажатия кнопок в окне
void CChildView::OnButClicket1()
{
   But1.EnableWindow( FALSE );          // Заблокировать кнопку 1
   But2.EnableWindow( TRUE );           // Разблокировать кнопку 2
   // AfxMessageBox(L"Нажали кнопку 1");
   fbut = true;                         // Можно рисовать
   SetCursor( hCurs2 );                 // Сделать текущим курсор "часы"
   Invalidate();                       // Перерисовка окна представления
}
void CChildView::OnButClicket2()
{
   But2.EnableWindow(FALSE);           // Заблокировать кнопку 2
   But1.EnableWindow(TRUE);            // Разблокировать кнопку 1
   //AfxMessageBox(L"Нажали кнопку 2");
   fbut = false;                       // Нельзя рисовать
   SetCursor(hCurs1);                  // Сделать текущим курсор "рука"
   Invalidate();                       // Перерисовать окно
}
```

ГЛАВА 4

Работа с меню

Создадим новый проект, на примере которого рассмотрим возможности создания и работы с различными видами меню (выпадающими, контекстными и т. п.).

Создайте проект **pr4** аналогично проекту **pr1** *(см. разд. 1.1)*.

Надо изменить следующие опции относительно изначально предложенных мастером:

- ❒ на вкладке **Application Type**:
  - SDI-документ (установить переключатель в положение **Single document**);
  - без поддержки архитектуры документ/представление (снять флажок **Document/View architecture support**);
- ❒ на вкладке **User Interface Features**:
  - без строки статуса (снять флажок **Initial status bar**);
  - без панели инструментов (установить переключатель **Toolbars** в положение **None**).

4.1. Описание программы

4.1.1. Добавление новых пунктов в меню

В меню приложения, созданного мастером, подготовлены три пункта: **File** (Файл), **Edit** (Правка) и **Help** (Справка). Эти пункты можно редактировать или удалить (об этом говорилось в *гл. 1*). Добавим свое выпадающее меню.

Откройте в файле ресурсов папку **Menu** (Меню). Добавьте новый пункт меню **Menu1** (щелкните левой кнопкой мыши по серому полю **Type Here** (Введите здесь), введите в нем название нового пункта меню и нажмите клавишу <Enter>, рис. 4.1). Аналогичным образом введите подпункты **M11** и **M12** (рис. 4.2). Новый пункт или подпункт меню всегда можно добавить в конец, но пункты меню можно перетаскивать ("подцепив" пункт левой кнопкой мыши). Переместим наше меню перед пунктом **Help** (рис. 4.3).

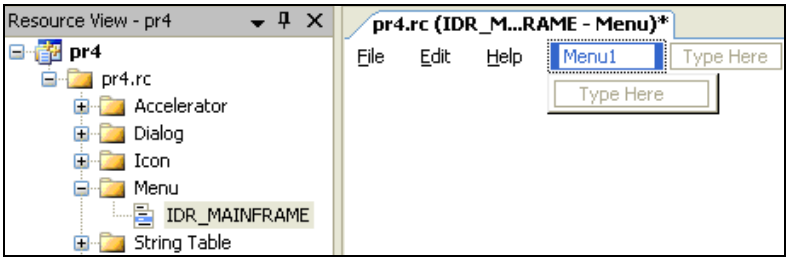


Рис. 4.1. Добавление пункта меню

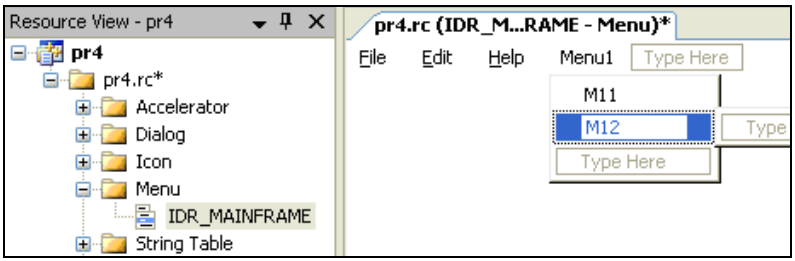


Рис. 4.2. Добавление подпунктов меню

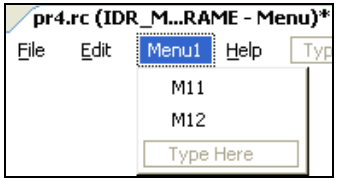


Рис. 4.3. Новое расположение пункта меню

Примечание

Добавлять меню можно и на русском языке. Только в этом случае мастер дает цифровые (непонятные) названия идентификаторов. Для получения

понятных названий надо изменить их в свойствах меню или сначала добавить английское название меню, а потом (когда мастер сделает нормальный идентификатор) изменить текст меню на русский.

Изменения в файле приведены в листинге 4.1.

Листинг 4.1. Изменения в файле Resourse.h для работы с новыми пунктами меню

```
// ...
#define ID_MENU1_M11                32771
#define ID_MENU1_M12                32772
// ...
```

Результаты работы программы показаны на рис. 4.4. Подпункты **M11** и **M12** по умолчанию недоступны (светло-серые).

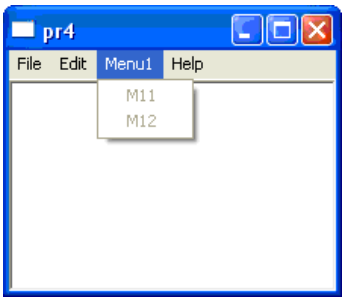


Рис. 4.4. Работа программы с новыми пунктами меню

Чтобы меню заработало, следует добавить обработку сообщений. Для этого надо вызвать для нужного пункта контекстное меню (щелкнув левой кнопкой мыши по выбранному пункту меню) и выполнить команду **Add Event Handle** (Добавить обработку события) (рис. 4.5).

В появившемся окне **Event Handle Wizard** (Мастер добавления событий) надо выбрать в поле **Message type** (Тип сообщения) — **COMMAND** (Команда). В поле **Class list** (Список классов) класс, который будет отвечать за обработку данного сообщения — **Cpr4App** (рис. 4.6), и имя функции, в которой будет обрабатываться сообщение (автоматически предлагается `OnMenu1M11()`).

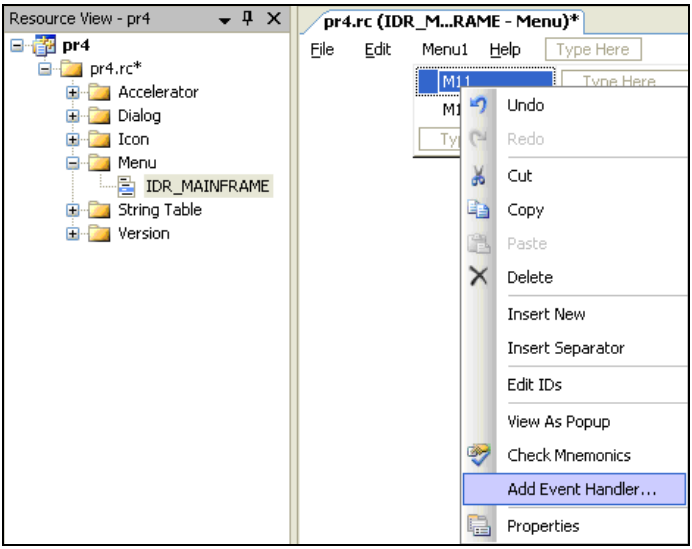


Рис. 4.5. Добавление обработки события пункта меню

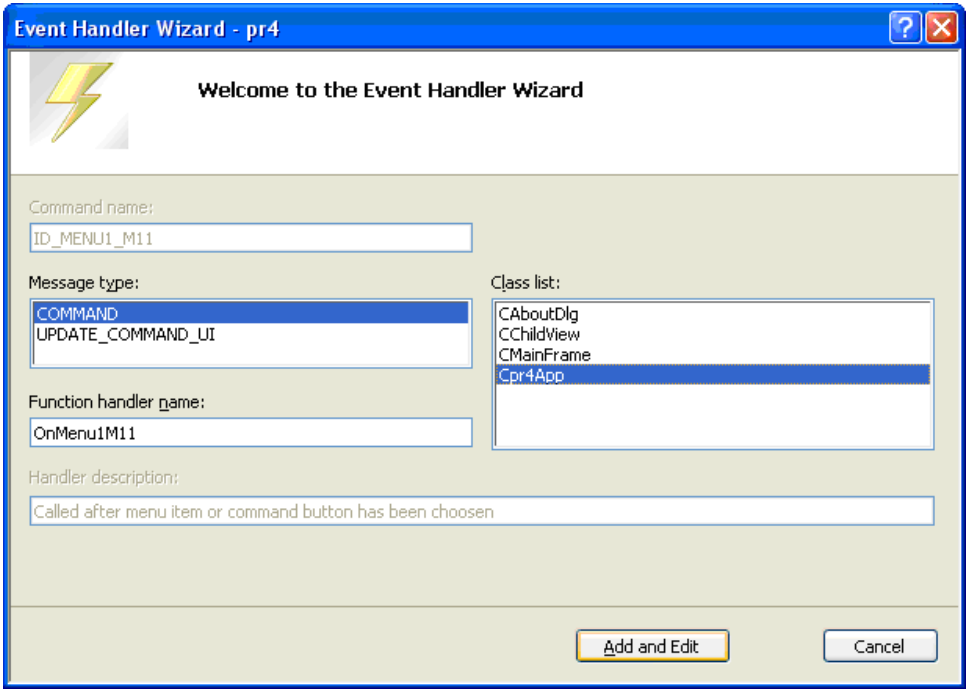


Рис. 4.6. Мастер добавления событий меню

Последовательность обработки командных сообщений следующая: сначала обработка командного сообщения ищется в `CChildView`, если там нет — то в `CMainFrame`, и потом в `Cpr4App`. Так что можно выбрать любой из этих классов. Предлагаемый по умолчанию класс `CAboutDlg` (отвечающий за диалоговое окно справки) выбирать *нельзя* (объект данного класса создается только при выборе меню **Help | Abuot pr4**, выводит окно справки и уничтожается; следовательно, он не может отвечать за обработку событий меню, т. к. его в это время просто не существует). После этого надо нажать кнопку **Add and Edit** (Добавить и отредактировать). Изменения в файлах приведены в листингах 4.2 и 4.3.

Листинг 4.2. Изменения в файле pr4.h для обработки меню

```
// ...
class Cpr4App : public CWinApp
{
   // ...
public:
   afx_msg void OnAppAbout();
   DECLARE_MESSAGE_MAP()
public:
   afx_msg void OnMenu1M11();
};
```

Листинг 4.3. Изменения в файле pr4.cpp для обработки меню

```
// ...
BEGIN_MESSAGE_MAP(Cpr4App, CWinApp)
   ON_COMMAND(ID_APP_ABOUT, &Cpr4App::OnAppAbout)
   ON_COMMAND(ID_MENU1_M11, &Cpr4App::OnMenu1M11)
END_MESSAGE_MAP()
// ...
void Cpr4App::OnMenu1M11()
{
   // TODO: Add your command handler code here
}
```

Теперь при запуске программы меню **M11** будет доступным, но при его выборе ничего происходить не будет. Добавим в обработку меню выдачу окна сообщения. Изменения в файле приведены в листинге 4.4.

Листинг 4.4. Изменения в файле pr4.cpp для обработки сообщения нового пункта меню

```
// ...
void Cpr4App::OnMenu1M11()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"M11");
}
```

Аналогичным образом добавим обработку пункта **M12** (с сообщением `AfxMessageBox(L"M12")`). Результаты работы программы показаны на рис. 4.7.

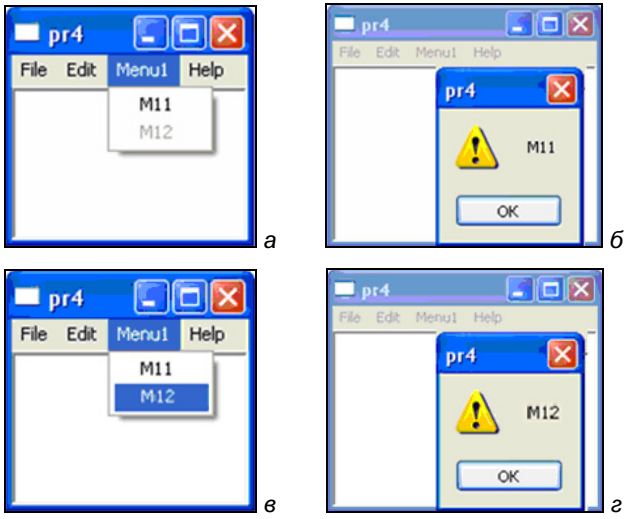


Рис. 4.7. Работа новых пунктов меню: выбор нового меню **M11** (*а*) и его обработка (*б*), выбор нового меню **M12** (*в*) и его обработка (*г*)

Теперь добавим "одиночное" верхнее меню **Menu2** (рис. 4.8, *а*). По умолчанию для верхних меню нет доступа к обработке сообщения (рис. 4.8, *б*). Чтобы добавить эту возможность, надо в свойствах (**Properties**) пункта ме-

ню в поле **Popup** (Выпадающее подменю) заменить `True` на `False` (рис. 4.8, *в*). Теперь можно добавить обработку события, аналогично пунктам меню **M11** и **M12**.

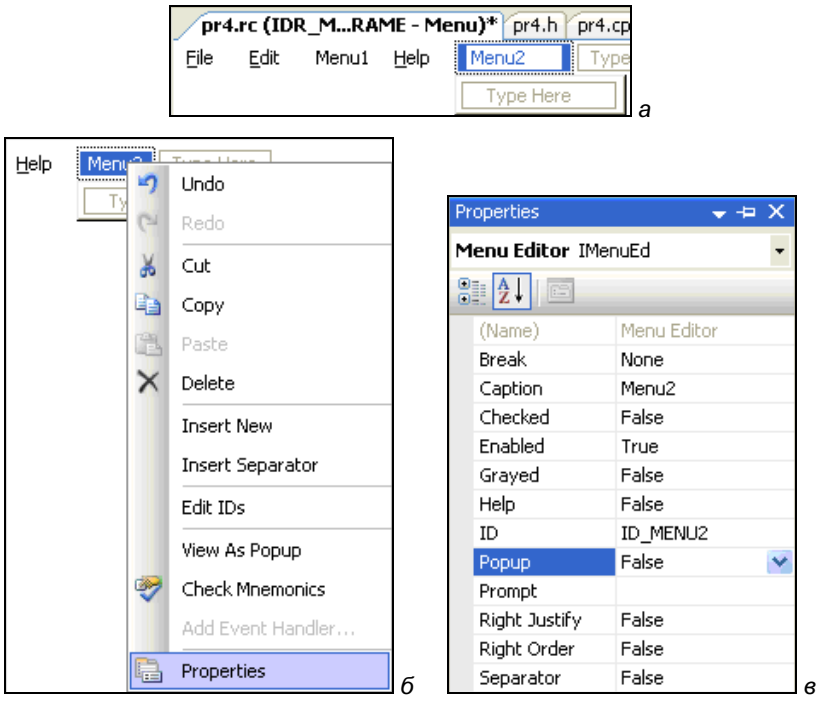


Рис. 4.8. Добавление "одиночного" пункта меню (*а*), контекстное меню нового пункта (*б*) и изменение свойств пункта меню (*в*)

Изменения в программе (для обработки всех добавленных пунктов меню) приведены в листингах 4.5—4.7.

Листинг 4.5. Изменения в файле Resourse.h для работы с меню верхнего и нижнего уровня

```
// ...
#define ID_MENU1_M11          32771
#define ID_MENU1_M12          32772
#define ID_MENU2              32773
// ...
```

Листинг 4.6. Изменения в файле pr4.h для работы с меню верхнего и нижнего уровня

```
// ...
class Cpr4App : public CWinApp
{
// ...
public:
   afx_msg void OnAppAbout();
   DECLARE_MESSAGE_MAP()
public:
   afx_msg void OnMenu1M11();
public:
   afx_msg void OnMenu1M12();
public:
   afx_msg void OnMenu2();
};
```

Листинг 4.7. Изменения в файле pr4.cpp для работы с меню верхнего и нижнего уровня

```
// ...
BEGIN_MESSAGE_MAP(Cpr4App, CWinApp)
   ON_COMMAND(ID_APP_ABOUT, &Cpr4App::OnAppAbout)
   ON_COMMAND(ID_MENU1_M11, &Cpr4App::OnMenu1M11)
   ON_COMMAND(ID_MENU1_M12, &Cpr4App::OnMenu1M12)
   ON_COMMAND(ID_MENU2, &Cpr4App::OnMenu2)
END_MESSAGE_MAP()
// ...
void Cpr4App::OnMenu1M11()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"M11");
}
void Cpr4App::OnMenu1M12()
{
```

```
    // TODO: Add your command handler code here
    AfxMessageBox(L"M12");
}
void Cpr4App::OnMenu2()
{
    // TODO: Add your command handler code here
    AfxMessageBox(L"Menu2");
}
```

4.1.2. Изменение работы пунктов меню

Сделаем так, чтобы при запуске программы меню **M12** было недоступно. При выборе **M11** меню **M12** становилось доступным (а **M11** — нет), при выборе **M12** меню **M11** становилось доступным (а **M12** — нет). Для этого добавим для флага. Изменения в файлах приведены в листингах 4.8 и 4.9.

Листинг 4.8. Изменения в файле pr4.h для синхронизации работы новых пунктов меню

```
// ...
class Cpr4App : public CWinApp
{
// ...
public:
    // Флаги доступности меню M11 и M12: true - меню доступно, false - нет
    bool    f_m11,
            f_m12;
// ...
};
```

Листинг 4.9. Изменения в файле pr4.cpp для синхронизации работы новых пунктов меню

```
// ...
// Cpr4App construction
Cpr4App::Cpr4App()
```

```
{
   // TODO: add construction code here,
   // Place all significant initialization in InitInstance
   f_m11 = true;        // Меню M11 изначально доступно
   f_m12 = false;       // Меню M12 - нет
}
```

Теперь надо добавить для **M11** обработку сообщения в класс `Cpr4App` (см. рис. 4.5 и 4.6). Только тип сообщения надо выбрать не **COMMAND**, а **UPDATE_COMMAND_UI** (рис. 4.9).

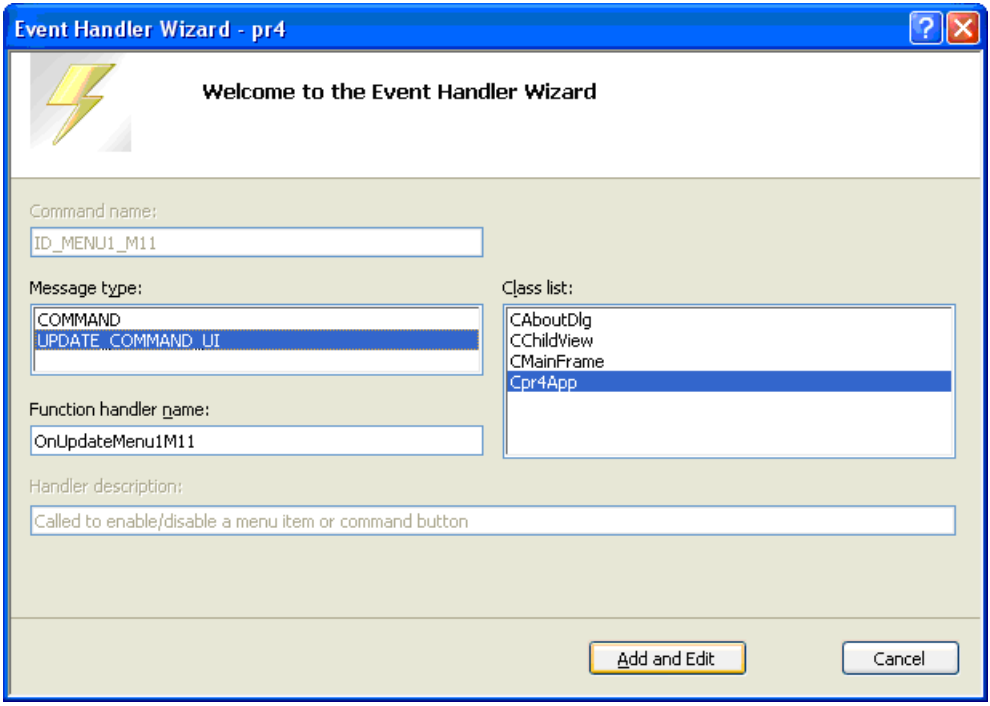


Рис. 4.9. Добавление сообщения для динамического изменения свойств меню

Изменения в файлах приведены в листингах 4.10 и 4.11.

Листинг 4.10. Изменения в файле pr4.h для динамического изменения состояния меню

```
// ...
class Cpr4App : public CWinApp
{
// ...
public:
   afx_msg void OnUpdateMenu1M11(CCmdUI *pCmdUI);
};
```

Листинг 4.11. Изменения в файле pr4.cpp для динамического изменения состояния меню

```
// ...
BEGIN_MESSAGE_MAP(Cpr4App, CWinApp)
   // ...
   ON_UPDATE_COMMAND_UI(ID_MENU1_M11, &Cpr4App::OnUpdateMenu1M11)
END_MESSAGE_MAP()
// ...
void Cpr4App::OnUpdateMenu1M11(CCmdUI *pCmdUI)
{
   // TODO: Add your command update UI handler code here
   pCmdUI->Enable(f_m11);
}
```

Сообщение `ON_UPDATE_COMMAND_UI` отвечает за обновление элемента (в данном случае меню **M11** — `ID_MENU1_M11`). Для организации обновления используется указатель на объект класса `CCmdUI`:

```
class CCmdUI;          // Класс интерфейса с командами пользователя
                       // (command user interface)
```

У данного класса есть следующие функции:

- блокировка элемента:

```
virtual void CCmdUI::Enable(
   BOOL bOn = TRUE);     // TRUE - элемент доступен, FALSE - нет
```

- установка или снятие флажка:

```
virtual void CCmdUI::SetCheck(
   int nCheck = 1);  // 1 - установлен, 0 - нет, 2 - неопределенное
                     // состояние (2 - только для кнопок панели
                     // инструментов
```

- блокировка группы переключателей:

```
virtual void CCmdUI::SetRadio(
   BOOL bOn = TRUE);   // TRUE - разблокирует, FALSE - блокирует
```

- устанавливает текст для данного элемента:

```
virtual void CCmdUI::SetText(
   LPCTSTR lpszText);  // Устанавливаемый текст
```

Аналогичным образом надо добавить обработку для **M12**. Только в обработке `OnUpdateMenu1M12()` написать:

```
pCmdUI->Enable(f_m12);
```

Теперь меню **M11** всегда доступно, а **M12** — нет. Чтобы пункты меню работали синхронно, надо добавить изменение значений флагов в обработку сообщений меню. Изменения в файле приведены в листинге 4.12.

Листинг 4.12. Изменения в файле pr4.cpp для синхронизации работы меню

```
// ...
void Cpr4App::OnMenu1M11()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"M11");
   f_m11 = false;
   f_m12 = true;
}
void Cpr4App::OnMenu1M12()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"M12");
   f_m12 = false;
   f_m11 = true;
}
```

Результат работы программы показан на рис. 4.10.

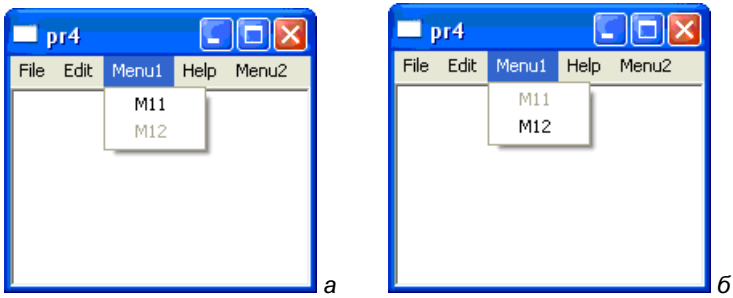


Рис. 4.10. Динамическое изменение меню: при запуске приложения (*а*), после выбора пункта меню **M11** (*б*)

4.1.3. Добавление и удаление пунктов меню

Добавим новый пункт меню **FMenu1** и два его подпункта **FMenu1 | Add** и **FMenu1 | Del** (рис. 4.11). Добавим обработку этих пунктов (только в класс `CMainFrame` — чтобы не "путаться" с предыдущими пунктами) (рис. 4.12).

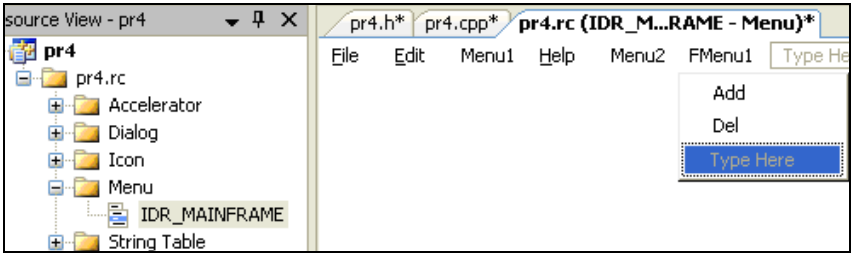


Рис. 4.11. Добавление новых пунктов меню

Изменения в файлах приведены в листингах 4.13—4.15.

Листинг 4.13. Изменения в файле Resourse.h для новых пунктов меню, позволяющих динамически добавлять и удалять пункты меню

```
// ...
#define ID_FMENU1_ADD                   32778
```

```
#define ID_FMENU1_DEL                   32779
// ...
```

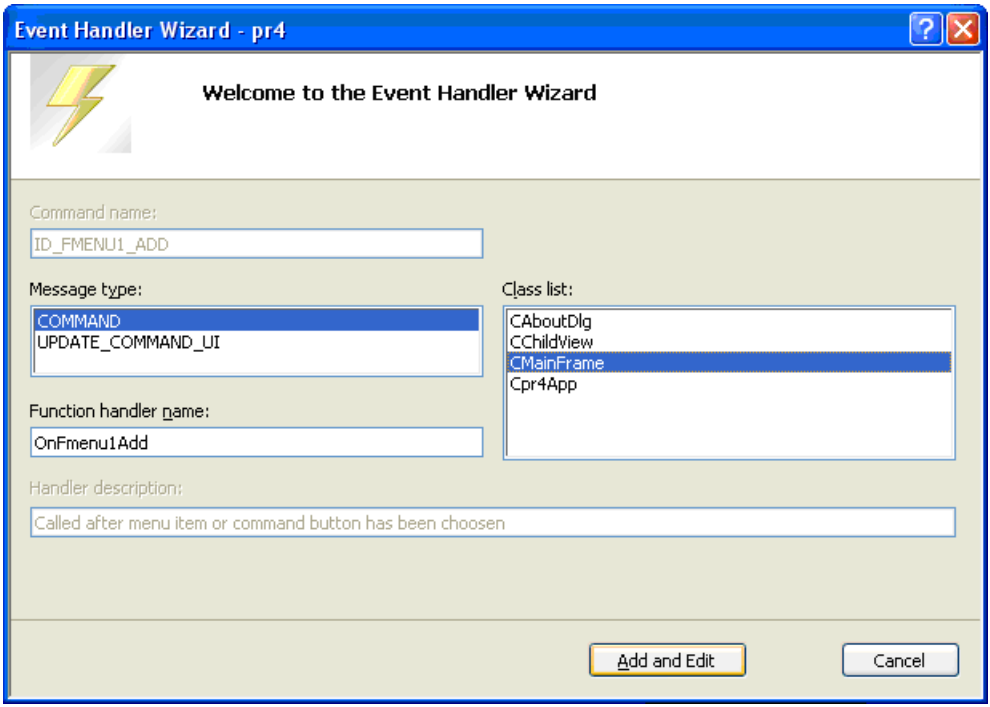


Рис. 4.12. Добавление обработки сообщений новых пунктов меню

Листинг 4.14. Изменения в файле MainFrm.h для новых пунктов меню, позволяющих динамически добавлять и удалять пункты меню

```
// ...
class CMainFrame : public CFrameWnd
{
   // ...
public:
   afx_msg void OnFmenu1Add();
public:
   afx_msg void OnFmenu1Del();
};
```

Листинг 4.15. Изменения в файле MainFrm.cpp для новых пунктов меню, позволяющих динамически добавлять и удалять пункты меню

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_CREATE()
   ON_WM_SETFOCUS()
   ON_COMMAND(ID_FMENU1_ADD, &CMainFrame::OnFmenu1Add)
   ON_COMMAND(ID_FMENU1_DEL, &CMainFrame::OnFmenu1Del)
END_MESSAGE_MAP()
// ...
void CMainFrame::OnFmenu1Add()
{
   // TODO: Add your command handler code here
}
void CMainFrame::OnFmenu1Del()
{
   // TODO: Add your command handler code here
}
```

Теперь сделаем так, чтобы при выборе меню в его конец (после меню **Del**) добавлялась разделительная черта (сепаратор) и новый пункт меню **New**. Для нового пункта надо завести идентификатор (вручную). Изменения в файлах приведены в листингах 4.16—4.18.

Листинг 4.16. Изменения в файле Resource.h для работы с новым пунктом меню

```
// ...
#define ID_FMENU1_ADD              32778
#define ID_FMENU1_DEL              32779
#define ID_FMENU1_NEW              2000
// ...
```

Листинг 4.17. Изменения в файле MainFrm.h для динамического создания и удаления нового пункта меню

```
// ...
class CMainFrame : public CFrameWnd
```

```
{
   // ...
public:
   CMenu *pmenu,  // Указатель на "верхнее" меню
         *psubm;  // Указатель на выпадающее меню
   int f_new;     // Флаг добавления нового пункта: 1 - добавлен, 0 -нет
   // ...
};
```

Листинг 4.18. Изменения в файле MainFrm.cpp для динамического создания и удаления нового пункта меню

```
// ...
CMainFrame::CMainFrame()
{
   f_new = 0;                // Вначале нового пункта нет
}
void CMainFrame::OnFmenu1Add()
{
   // TODO: Add your command handler code here
   static int n;
   if(!n)                    // Только для первого раза
   {
      // Получить указатель на главное окно
      CWnd* pMain = AfxGetMainWnd();
      // Получить указатель на "верхнее" меню
      pmenu = pMain->GetMenu();
      // Получить указатель на выпадающее из "Fmenu1"
      psubm = pmenu->GetSubMenu(5);
      n = 1;
   }
   if(!f_new)                // Если новый пункт не добавлен
   {
      // Добавить сепаратор (черту-разделитель)
      psubm->AppendMenuW(MF_SEPARATOR);
```

```
      // Добавить в конец существующего выпадающего меню новый пункт
      psubm->AppendMenuW(MF_STRING, ID_FMENU1_NEW, L"New" );
      f_new = 1;
   }
}

void CMainFrame::OnFmenu1Del()
{
   // TODO: Add your command handler code here
   if(f_new)                        // Если новый пункт существует
   {
      // Удалить новый пункт
      psubm->RemoveMenu(ID_FMENU1_NEW, MF_BYCOMMAND);
      // Удалить сепаратор
      psubm->RemoveMenu(2, MF_BYPOSITION);
      f_new = 0;
   }
}
```

Получение указателя на меню окна выполняется с помощью функции:

```
CMenu* CWnd::GetMenu() const;
```

Получение указателя на подменю (выпадающего меню):

```
CMenu* CMenu::GetSubMenu(
   int nPos) const;  // Индекс пункта меню, из которого открывается
                     // выпадающее
```

Индексация пунктов меню начинается с 0. Для данной программы `nPos = 5` (0 — **File**, 1 — **Edit**, 2 — **Menu1**, 3 — **Help**, 4 — **Menu2**, 5 — **Fmenu1**).

Добавление нового пункта в конец выполняется функцией:

```
#define AppendMenuW AppendMenu
BOOL CMenu::AppendMenu(           // 0 - ошибка
   UINT nFlags,                   // Состояние нового пункта
   UINT_PTR nIDNewItem = 0,       // Идентификатор команды нового пункта
   LPCTSTR lpszNewItem = NULL);   // Название (текст) нового пункта
```

или

```
BOOL CMenu::AppendMenu(
   UINT nFlags,
   UINT_PTR nIDNewItem,
   const CBitmap* pBmp);          // Рисунок вместо названия нового пункта
```

Флаги состояния меню (`nFlags`) могут быть такими:

- `MF_CHECKED` — пункт меню помечено "галочкой";
- `MF_UNCHECKED` — пункт меню не помечено "галочкой";
- `MF_DISABLED` — пункт меню заблокирован;
- `MF_ENABLED` — пункт меню доступен;
- `MF_GRAYED` — пункт меню заблокирован и серого цвета;
- `MF_MENUBREAK` — пункт меню верхнего уровня выводится с новой строки, а пункт подменю — в новом столбце без разделительной вертикальной линии;
- `MF_MENUBARBREAK` — аналогично `MF_MENUBREAK`, но новый столбец отделяется разделительной вертикальной линией;
- `MF_OWNERDRAW` — меню само отвечает за изображение своего элемента;
- `MF_POPUP` — пункт меню имеет свое подменю, дескриптор которого должен задаваться в `nIDNewItem`;
- `MF_SEPARATOR` — вместо пункта меню выводится разделительная черта (сепаратор);
- `MF_STRING` — пункт меню представляет собой текстовую строку.

При этом есть недопустимые комбинации флагов:

- `MF_DISABLED`, `MF_ENABLED` и `MF_GRAYED`;
- `MF_STRING`, `MF_OWNERDRAW`, `MF_SEPARATOR` и битовый массив (bitmap);
- `MF_MENUBARBREAK` и `MF_MENUBREAK`;
- `MF_CHECKED` и `MF_UNCHECKED`.

Удаление пункта меню выполняется с помощью функции:

```
BOOL CMenu::RemoveMenu(          // 0 - ошибка
   UINT nPosition,               // Задает элемент меню
   UINT nFlags);                 // Флаг определения элемента
```

Значения флагов определения элемента меню (`nFlags`) следующие:

- `MF_BYCOMMAND` — удаление по идентификатору (в `nPosition` надо указать идентификатор пункта меню);
- `MF_BYPOSITION` — удаление по индексу (в `nPosition` надо указать индекс пункта меню).

В данной программе у сепаратора индекс 2 (0 — **Add**, 1 — **Del**, 2 — сепаратор, 3 — **New**). В результате работы такой программы меню **New** появляется, но оно не активно. Надо добавить его обработку *самостоятельно* (с помощью мастера этого сделать нельзя, т. к. изначально этого меню не существует). Изменения в файлах приведены в листингах 4.19 и 4.20.

Листинг 4.19. Изменения в файле MainFrm.h для обработки сообщения нового пункта меню

```
// ...
class CMainFrame : public CFrameWnd
{
   // ...
public:
   afx_msg void OnFmenu1New();
};
```

Листинг 4.20. Изменения в файле MainFrm.cpp для обработки сообщения нового пункта меню

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   // ...
   ON_COMMAND(ID_FMENU1_DEL, &CMainFrame::OnFmenu1Del)
   ON_COMMAND(ID_FMENU1_NEW, &CMainFrame::OnFmenu1New)
END_MESSAGE_MAP()
   // ...
void CMainFrame::OnFmenu1New()
{
   AfxMessageBox(L"New");
}
```

Результат работы программы показан на рис. 4.13.

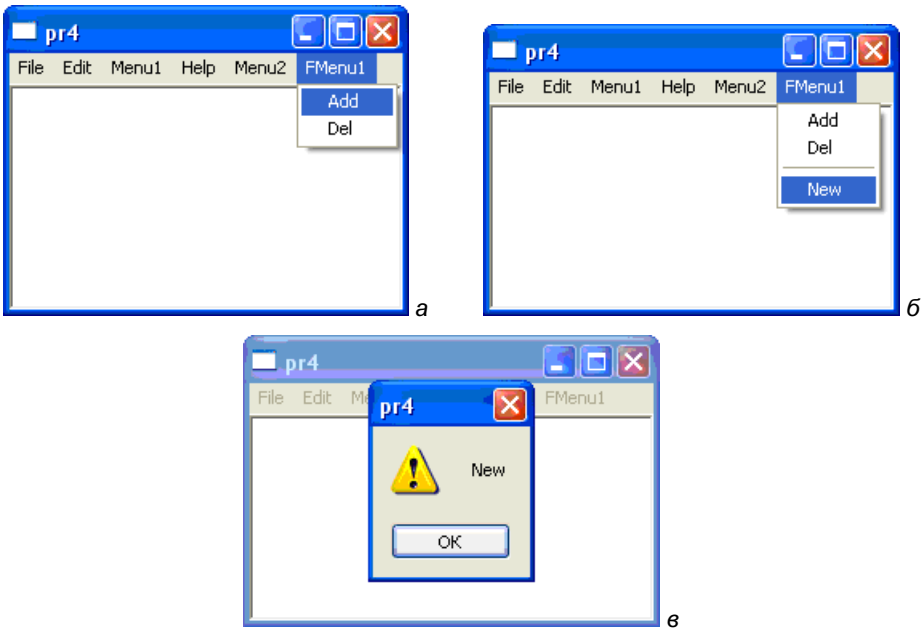


Рис. 4.13. Динамическое добавление выпадающего меню: при загрузке приложения (*а*); добавлен новый пункт меню (*б*); работа динамического меню (*в*)

4.1.4. Добавление контекстного меню

Добавим в программу вызов контекстного меню (которое появляется при нажатии в области окна представления правой кнопки мыши). Для этого надо вызвать для ресурса меню контекстное меню и выполнить команду **Insert Menu** (Вставить меню). В появившемся новом ресурсе меню `IDR_MENU1` добавим пункт **con1** и два подпункта **con11** и **con12** (рис. 4.14).

Изменения в файле приведены в листинге 4.21.

Листинг 4.21. Изменения в файле Resourse.h для работы с контекстным меню

```
// ...
#define IDR_MENU1                       130
```

```
#define ID_CON1_CON11                32780
#define ID_CON1_CON12                32781
// ...
```

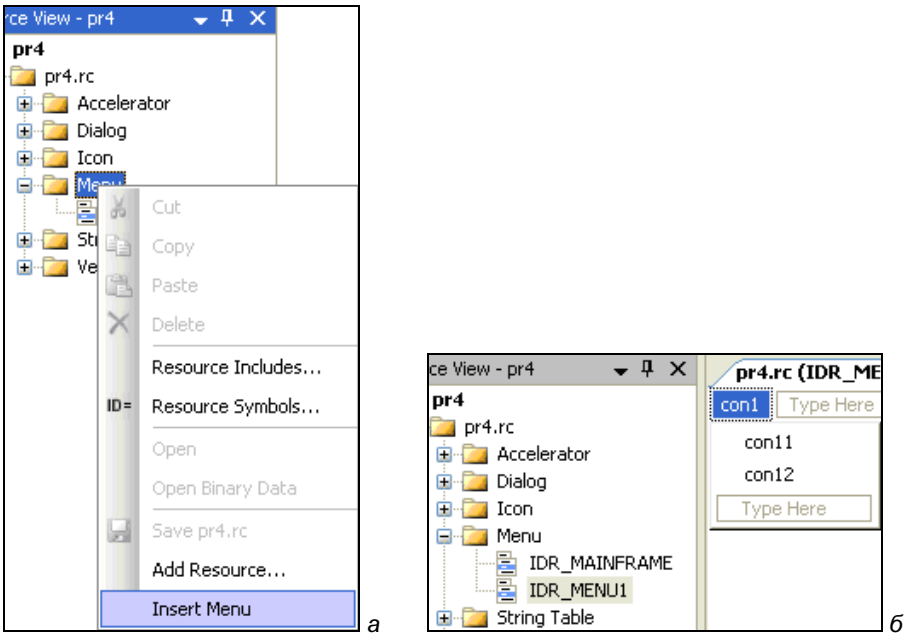


Рис. 4.14. Добавление нового ресурса меню (*а*) и его пунктов (*б*)

Теперь внесем изменения в программу. За работу контекстного меню будет отвечать класс окна представления (это логично и нет путаницы с предыдущими обработками меню). Для вызова контекстного меню надо в класс окна представления (`CChildView`) добавить обработку нажатия правой кнопки мыши:

- в окне **Class View** вызвать контекстное меню и открыть окно свойств **Properties** для класса `CChildView` (рис. 4.15, *а* и *б*);
- в окне свойств выбрать пятую кнопку **Message** (Сообщения) в верхней строке (рис. 4.15, *в*);
- в окне появившегося списка сообщений найти сообщение `WM_RBUTTONDOWN`. Раскрыть его (нажав кнопку со стрелкой вниз справа от сообщения) и выбрать **<Add>OnRButtonDown** (рис. 4.15, *г*).

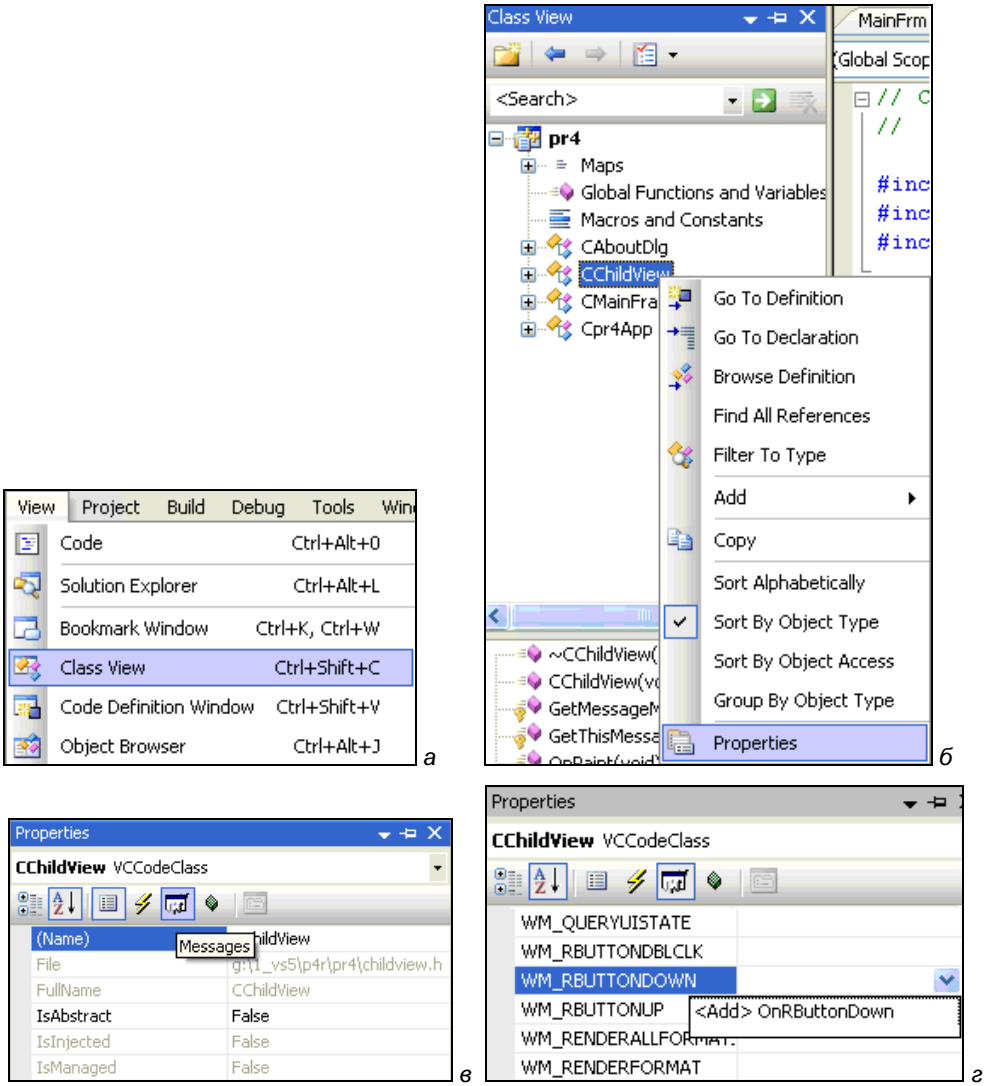


Рис. 4.15. Добавление обработки сообщения в класс: открытие окна просмотра классов (*а*), вызов окна свойств для класса (*б*), выбор списка сообщений для класса (*в*), добавление обработки сообщения о нажатии правой кнопки мыши (*г*)

Примечание

Иногда при добавлении обработчика происходит сбой и выдается сообщение об ошибке. Тогда надо просто повторить эти действия еще раз.

Изменения в файлах приведены в листингах 4.22 и 4.23.

Листинг 4.22. Изменения в файле ChildView.h для работы с контекстным меню

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   CChildView();
   CMenu *p_popup;              // Указатель на новый ресурс меню
   // ...
protected:
   afx_msg void OnPaint();
   DECLARE_MESSAGE_MAP()
public:
   // Обработка сообщения при нажатии правой кнопки мыши
   afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
};
```

Листинг 4.23. Изменения в файле ChildView.cpp для работы с контекстным меню

```
// ...
CChildView::CChildView()
{
   p_popup = new CMenu;
   p_popup->LoadMenuW(IDR_MENU1);   // Загружаем новый ресурс меню
}
CChildView::~CChildView()
{
   if(p_popup)
   {
      p_popup->DestroyMenu();       // Удаляем новый ресурс меню
      delete p_popup;
   }
```

```
}
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_RBUTTONDOWN()
END_MESSAGE_MAP()
// ...
void CChildView::OnRButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   // Преобразовать координаты клиентской области в координаты экрана
   ClientToScreen(&point);
   // Получить нужное подменю из нового ресурса меню
   CMenu *psub = p_popup->GetSubMenu(0);
   if(psub)                          // Если такое подменю существует
   {
      // Отобразить его в том месте, где щелкнули правой кнопкой мыши
      psub->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON, point.x,
                           point.y, this);
   }

   CWnd::OnRButtonDown(nFlags, point);
}
```

Рассмотрим далее функции, используемые при работе с контекстным меню.

Загрузка ресурса меню выполняется с помощью:

```
BOOL CMenu::LoadMenu(               // 0 - ошибка
   LPCTSTR lpszResourceName);       // Строка с именем загружаемого меню
```

или

```
BOOL CMenu::LoadMenu(
   UINT nIDResource);               // Идентификатор загружаемого меню
```

Удаление меню выполняется функцией:

```
BOOL CMenu::DestroyMenu();          // 0 - ошибка
```

Обработка сообщения от правой кнопки мыши реализуется следующим образом:

```
afx_msg void CWnd::OnRButtonDown(
   UINT nFlags,              // Состояние кнопок в момент сообщения мыши
   CPoint point);            // Координаты курсора в момент сообщения мыши
```

Значения флагов кнопок (`nFlags`) могут быть такие:

- `MK_CONTROL` — в момент нажатия кнопки мыши была нажата клавиша <Ctrl>;
- `MK_LBUTTON` — была нажата левая кнопка мыши;
- `MK_MBUTTON` — была нажата средняя кнопка мыши;
- `MK_RBUTTON` — была нажата правая кнопка мыши;
- `MK_SHIFT` — в момент нажатия кнопки мыши была нажата клавиша <Shift>.

Значения флагов могут комбинироваться. Пример использования флага:

```
if(nFlags && MK_SHIFT)
{
   // Была нажата кнопка мыши вместе с клавишей <Shift>
}
```

Преобразование координат клиентское области в координаты экрана выполняется с помощью функции:

```
void CWnd::ClientToScreen(
   LPPOINT lpPoint) const;
void CWnd::ClientToScreen(
   LPRECT lpRect) const;
```

> ***Примечание***
>
> Для использования нескольких контекстных меню не надо добавлять новые ресурсы. Надо в ресурсе `IDR_MENU1` задать нужное количество меню (с подменю), как показано на рис. 4.16, и, с помощью функции `GetSubMenu()`, выбрать нужное (`GetSubMenu(0)` — подменю у **con1**, `GetSubMenu(1)` — подменю у **mmm**).

Для поддержки работы с контекстным меню используется функция:

```
BOOL CMenu::TrackPopupMenu (    // 0 - ошибка
   UINT nFlags,          // Расположение на экране и задание кнопки мыши
   int x,                // Начальные координаты появления меню
```

```
int y,
CWnd* pWnd,           // Указатель на окно, которое будет обрабатывать
                      // сообщения от контекстного меню
LPCRECT lpRect = 0);  // Область действия меню
```

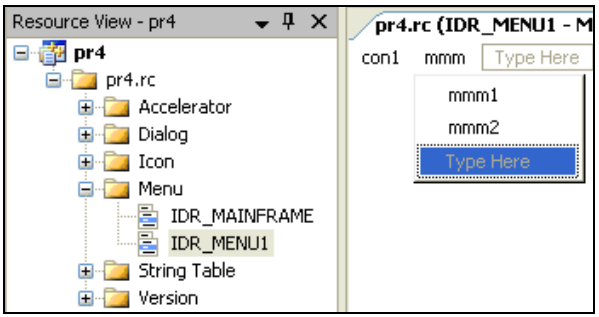


Рис. 4.16. Поддержка нескольких контекстных меню

Эта функция выводит на экран контекстное меню и создает собственный цикл обработки сообщений. Работа функции завершается только после окончания работы с контекстным меню (выбором пункта меню или отказом).

Значения расположения меню на экране (`nFlags`) могут быть такие:

- `TPM_CENTERALIGN` — центрирование относительно координаты, заданной в параметре `x`;
- `TPM_LEFTALIGN` — выравнивание по левой границе относительно координаты, заданной в `x`;
- `TPM_RIGHTALIGN` — выравнивание по правой границе относительно координаты, заданной в `x`.

Значения кнопок мыши для меню (`nFlags`) могут быть такие:

- `TPM_LEFTBUTTON` — использование левой кнопки мыши для появления меню;
- `TPM_RIGHTBUTTON` — использование правой кнопки мыши для появления меню.

Область действия меню (`lpRect`) — это координаты прямоугольной области, в которой пользователь может выполнять выбор из меню. Если щелчок мышью (для выбора пункта) будет сделан вне этой области, то контекстное ме-

ню исчезнет. Если `lpRect = 0`, то размеры и расположение области действия меню будут совпадать с размерами самого контекстного меню.

Меню появляется, но при выборе его пунктов ничего не происходит. Надо добавить обработку пунктов меню в класс окна представления. Это делается с помощью мастера, аналогично обработке обычного меню. Изменения в файлах приведены в листингах 4.24 и 4.25.

Листинг 4.24. Изменения в файле ChildView.h для обработки сообщений контекстного меню

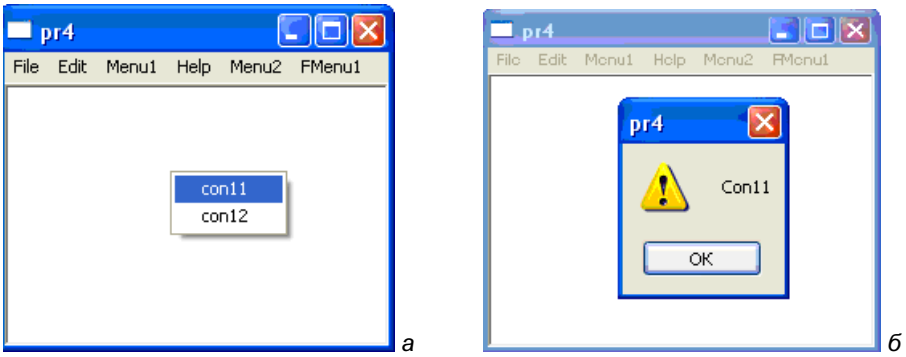
```
// ...
class CChildView : public CWnd
{
   // ...
public:
   afx_msg void OnCon1Con11();
public:
   afx_msg void OnCon1Con12();
};
```

Листинг 4.25. Изменения в файле ChildView.cpp для обработки сообщений контекстного меню

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   // ...
   ON_COMMAND(ID_CON1_CON11, &CChildView::OnCon1Con11)
   ON_COMMAND(ID_CON1_CON12, &CChildView::OnCon1Con12)
END_MESSAGE_MAP()
// ...
void CChildView::OnCon1Con11()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"Con11");
}
void CChildView::OnCon1Con12()
```

```
{
    // TODO: Add your command handler code here
    AfxMessageBox(L"Con12");
}
```

Результат работы программы показан на рис. 4.17.



а *б*

Рис. 4.17. Работа с контекстным меню: вызов контекстного меню (*а*) и обработка выбора его пункта (*б*)

4.1.5. Некоторые полезные функции для работы с меню

Вставка нового пункта меню в заданную позицию выполняется с помощью функции:

```
BOOL CMenu::InsertMenu(      // 0 - ошибка
   UINT nPosition,           // Позиция для вставки или идентификатор
   UINT nFlags,              // Флаг для выбора позиции или идентификатора
   UINT_PTR nIDNewItem = 0, // Идентификатор команды нового пункта
   LPCTSTR lpszNewItem = NULL);   // Название (текст) нового пункта
```

или

```
BOOL CMenu::InsertMenu(
   UINT nPosition,
   UINT nFlags,
```

```
   UINT_PTR nIDNewItem,
   const CBitmap* pBmp);     // Рисунок вместо названия нового пункта
```

Значения флагов для вставки нового пункта меню (`nFlags`) такие же, как и у `AppendMenu()`. Кроме этого, эти значения могут комбинироваться (с помощью |) со следующими:

- `MF_BYCOMMAND` — в параметре `nPosition` задается идентификатор элемента меню, перед которым будет вставлен новый;
- `MF_BYPOSITION` — в параметре `nPosition` задается индекс элемента меню (начиная с нуля), перед которым будет вставлен новый. Если `nPosition = -1`, то новый элемент будет добавлен в конец меню.

Замена существующего пункта меню (параметры аналогичны `AppendMenu()` и `InsertMenu()`, только новый элемент не вставляется в меню, а заменяет тот, на который указывает параметр `nPosition`) выполняется функцией:

```
BOOL CMenu::ModifyMenu(            // 0 - ошибка
   UINT nPosition,
   UINT nFlags,
   UINT_PTR nIDNewItem = 0,
   LPCTSTR lpszNewItem = NULL);
```

или

```
BOOL CMenu::ModifyMenu(
   UINT nPosition,
   UINT nFlags,
   UINT_PTR nIDNewItem,
   const CBitmap* pBmp);
```

При удалении контекстного меню с помощью функции `RemoveMenu()` в дальнейшем можно снова воспользоваться удаленным элементом по его идентификатору. Для полного удаления контекстного меню и освобождения связанных с ним ресурсов используется функция:

```
BOOL CMenu::DeleteMenu(             // 0 - ошибка
   UINT nPosition,
   UINT nFlags);
```

Параметры функции аналогичны `InsertMenu()`.

Доступ к пункту меню изменяется функцией:

```
UINT CMenu::EnableMenuItem(        // Предыдущее состояние элемента или
                                   // -1, если элемента нет
```

```
   UINT nIDEnableItem,           // Аналогично nPosition в InsertMenu()
   UINT nEnable);                // Флаг нового состояния элемента меню
```

Значения флагов состояния элемента меню (`nEnable`) могут быть такими:

- `MF_DISABLED` — элемент доступен;
- `MF_ENABLED` — элемент недоступен;
- `MF_GRAYED` — элемент заблокирован и серого цвета.

Значение флага `nEnable` *должно* комбинироваться с `MF_BYCOMMAND` или `MF_BYPOSITION` для определения того, что задано в `nIDEnableItem`.

Пометить элемент меню ("галочкой") можно с помощью функции:

```
UINT CMenu::CheckMenuItem(   // Предыдущее состояние элемента
                             // или -1, если элемента нет
   UINT nIDCheckItem,        // Аналогично nPosition в InsertMenu()
   UINT nCheck);             // Флаг пометки элемента меню
```

Значения флагов пометки меню (`nCheck`) могут быть такие:

- `MF_CHECKED` — помечен;
- `MF_UNCHECKED` — нет.

Значение флага `nCheck` *должно* комбинироваться с `MF_BYCOMMAND` или `MF_BYPOSITION` для определения того, что задано в `nIDCheckItem`.

Получить количество элементов меню можно с помощью функции:

```
UINT CMenu::GetMenuItemCount() const;
```

Получить идентификатор подменю по его индексу можно с помощью функции:

```
UINT CMenu::GetMenuItemID(
   int nPos) const;           // Индекс элемента (начиная с 0)
```

Если параметр `nPos` указывает на сепаратор, то функция возвращает 0.

4.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 4.26. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDR_MENU1               130
#define ID_MENU1_M11            32771
#define ID_MENU1_M12            32772
#define ID_MENU2                32773
#define ID_FMENU1_NEW           2000
#define ID_FMENU1_ADD           32778
#define ID_FMENU1_DEL           32779
#define ID_CON1_CON11           32780
#define ID_CON1_CON12           32781
// ...
```

Листинг 4.27. Файл Pr4.h (объявление класса приложения)

```
// ...
class Cpr4App : public CWinApp
{
   // ...
public:
   Cpr4App();
   // Флаги доступности меню M11 и M12: true - меню доступно, false - нет
   bool  f_m11,
         f_m12;
   // ...
public:
   afx_msg void OnMenu1M11();
public:
   afx_msg void OnMenu1M12();
public:
   afx_msg void OnMenu2();
public:
   afx_msg void OnUpdateMenu1M11(CCmdUI *pCmdUI);
public:
   afx_msg void OnUpdateMenu1M12(CCmdUI *pCmdUI);
};
```

Листинг 4.28. Файл Pr4.cpp (определение класса приложения)

```
// ...
BEGIN_MESSAGE_MAP(Cpr4App, CWinApp)
   ON_COMMAND(ID_APP_ABOUT, &Cpr4App::OnAppAbout)
   ON_COMMAND(ID_MENU1_M11, &Cpr4App::OnMenu1M11)
   ON_COMMAND(ID_MENU1_M12, &Cpr4App::OnMenu1M12)
   ON_COMMAND(ID_MENU2, &Cpr4App::OnMenu2)
   ON_UPDATE_COMMAND_UI(ID_MENU1_M11, &Cpr4App::OnUpdateMenu1M11)
   ON_UPDATE_COMMAND_UI(ID_MENU1_M12, &Cpr4App::OnUpdateMenu1M12)
END_MESSAGE_MAP()

// Cpr4App construction
Cpr4App::Cpr4App()
{
   // TODO: add construction code here,
   // Place all significant initialization in InitInstance
   f_m11 = true;            // Меню M11 изначально доступно
   f_m12 = false;           // Меню M12 - нет
}
// ...
// Cpr4App message handlers
void Cpr4App::OnMenu1M11()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"M11");
   f_m11 = false;
   f_m12 = true;
}
void Cpr4App::OnMenu1M12()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"M12");
   f_m12 = false;
   f_m11 = true;
}
```

```
void Cpr4App::OnMenu2()
{
   // TODO: Add your command handler code here
   AfxMessageBox(L"Menu2");
}
void Cpr4App::OnUpdateMenu1M11(CCmdUI *pCmdUI)
{
   // TODO: Add your command update UI handler code here
   pCmdUI->Enable(f_m11);
}
void Cpr4App::OnUpdateMenu1M12(CCmdUI *pCmdUI)
{
   // TODO: Add your command update UI handler code here
   pCmdUI->Enable(f_m12);
}
```

Листинг 4.29. Файл MainFrm.h (объявление класса окна фрейма)

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   CMainFrame();
   CMenu    *pmenu,          // "Верхнее" меню
            *psubm;          // Выпадающее меню
   int f_new;                // Флаг добавления нового пункта:
                             // 1 - добавлен, 0 - нет
// ...
public:
   afx_msg void OnFmenu1Add();
public:
   afx_msg void OnFmenu1Del();
public:
   afx_msg void OnFmenu1New();
};
```

Листинг 4.30. Файл MainFrm.cpp (определение класса окна фрейма)

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_CREATE()
   ON_WM_SETFOCUS()
   ON_COMMAND(ID_FMENU1_ADD, &CMainFrame::OnFmenu1Add)
   ON_COMMAND(ID_FMENU1_DEL, &CMainFrame::OnFmenu1Del)
   ON_COMMAND(ID_FMENU1_NEW, &CMainFrame::OnFmenu1New)
END_MESSAGE_MAP()
// CMainFrame construction/destruction
CMainFrame::CMainFrame()
{
   // TODO: add member initialization code here
   f_new = 0;              // Вначале нового пункта нет
}
// ...
void CMainFrame::OnFmenu1Add()
{
   // TODO: Add your command handler code here
   static int n;
   if(!n)                  // Только для первого раза
   {
      // Получить указатель на главное окно
      CWnd* pMain = AfxGetMainWnd();
      // Получить указатель на "верхнее" меню
      pmenu = pMain->GetMenu();
      // Получить указатель на выпадающее из "Fmenu1"
      psubm = pmenu->GetSubMenu(5);
      n = 1;
   }
   if(!f_new)              // Если новый пункт не добавлен
   {
      // Добавить сепаратор (черту-разделитель)
      psubm->AppendMenuW(MF_SEPARATOR);
```

```
      // Добавить в конец существующего выпадающего меню новый пункт
      psubm->AppendMenuW(MF_STRING, ID_FMENU1_NEW, L"New");
      f_new = 1;
   }
}
void CMainFrame::OnFmenu1Del()
{
   // TODO: Add your command handler code here
   if(f_new)                    // Если новый пункт существует
   {
      // Удалить новый пункт
      psubm->RemoveMenu(ID_FMENU1_NEW, MF_BYCOMMAND);
      // Удалить сепаратор
      psubm->RemoveMenu(2, MF_BYPOSITION);
      f_new = 0;
   }
}
void CMainFrame::OnFmenu1New()
{
   AfxMessageBox(L"New");
}
```

Листинг 4.31. Файл ChildView.h (объявление класса окна представления)

```
// ...
class CChildView : public CWnd
{
// ...
public:
   CChildView();
   CMenu *p_popup;             // Указатель на новый ресурс меню
// ...
public:
   // Обработка сообщения при нажатии правой кнопки мыши
   afx_msg void OnRButtonDown(UINT nFlags, CPoint point );
```

```
public:
   afx_msg void OnCon1Con11();
public:
   afx_msg void OnCon1Con12();
};
```

Листинг 4.32. Файл ChildView.cpp (определение класса окна представления)

```
// ...
CChildView::CChildView()
{
   p_popup = new CMenu;
   p_popup->LoadMenuW(IDR_MENU1);        // Загружаем новый ресурс меню
}
CChildView::~CChildView()
{
   if(p_popup)
   {
      p_popup->DestroyMenu();            // Удаляем новый ресурс меню
      delete p_popup;
   }
}
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_RBUTTONDOWN()
   ON_COMMAND(ID_CON1_CON11, &CChildView::OnCon1Con11)
   ON_COMMAND(ID_CON1_CON12, &CChildView::OnCon1Con12)
END_MESSAGE_MAP()
// ...
void CChildView::OnRButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   // Преобразовать координаты клиентской области в координаты экрана
   ClientToScreen(&point);
```

```
    // Получить нужное подменю из нового ресурса меню
    CMenu *psub = p_popup->GetSubMenu(0);
    if(psub)                          // Если такое подменю существует
    {   // Отобразить его в том месте, где щелкнули правой кнопкой мыши
       psub->TrackPopupMenu(TPM_LEFTALIGN | TPM_RIGHTBUTTON , point.x,
                            point.y, this);
    }
    CWnd::OnRButtonDown(nFlags, point);
}
void CChildView::OnCon1Con11()
{
    // TODO: Add your command handler code here
    AfxMessageBox(L"Con11");
}
void CChildView::OnCon1Con12()
{
    // TODO: Add your command handler code here
    AfxMessageBox(L"Con12");
}
```

ГЛАВА 5

Виртуальное окно, клавиатура, дочернее окно

Создадим новый проект, на примере которого рассмотрим возможности работы с виртуальным окном и использование его для масштабирования изображения, работу с вертикальной линейкой прокрутки, правила обработки сообщений клавиатуры и способы создания дочерних окон.

Создайте проект **pr5** аналогично проекту **pr1** *(см. разд. 1.1)*. Надо изменить следующие опции, относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ;
  - без поддержки архитектуры документ/представление;
- на вкладке **User Interface Features**:
  - без строки статуса;
  - без панели инструментов.

5.1. Описание программы

5.1.1. Проблема перерисовки — виртуальное окно

В предыдущих проектах *(см. разд. 2.1)* вся выводимая в окно представления информация создавалась и отображалась в обработке сообщения `CChildView::OnPaint()`. Это нерентабельно, т. к. эти действия придется делать каждый раз при перерисовке окна (например, при изменении его размеров). Намного удобнее всю выводимую информацию создавать в виртуаль-

ном окне, а при перерисовке просто копировать его содержимое в реальное окно. Кстати, виртуальных окон может быть несколько. Виртуальное окно строится на основе растрового изображения (битового рисунка). Работа с битовыми рисунками была рассмотрена в *разд. 3.1.2* (см. функции `CChildView::setbmp()`, `CChildView::OnPaint()`, `CMainFrame::OnCreate()`).

Действия при создании виртуального окна практически идентичные. Добавим новые данные. Изменения в файлах приведены в листингах 5.1 и 5.2.

Листинг 5.1. Изменения в файле ChildView.h для работы с виртуальным окном

```
// ...
class CChildView : public CWnd
{
// ...
public:
   CChildView();
   CBitmap virt_wnd;                // Битовый рисунок - виртуальное окно
   CDC dc_virt;                     // Контекст виртуального окна
   int maxX, maxY;                  // Ширина и высота экрана
   void create_virt(void);          // Создание виртуального окна
// ...
};
```

Листинг 5.2. Изменения в файле ChildView.cpp для работы с виртуальным окном

```
// ...
CChildView::~CChildView()
{
   dc_virt.DeleteDC();
}
void CChildView::create_virt( void )
{
   // Получение размеров экрана
   maxX = GetSystemMetrics(SM_CXSCREEN);
   maxY = GetSystemMetrics(SM_CYSCREEN);
```

```
   //Строим растр (виртуальное окно), совместимый с реальным окном
   CDC *dc;                          // Контекст устройства окна
   dc = GetDC();                     // Получаем контекста устройства окна
   dc_virt.CreateCompatibleDC(dc);  // Строим растр, совместимый с окном
   // Создаем в памяти растровое изображение (виртуальное окно),
   // совместимое с окном представления
   virt_wnd.CreateCompatibleBitmap(dc, maxX, maxY);
   dc_virt.SelectObject(&virt_wnd);  // Подключаем виртуальное окно
   // Закрашиваем виртуальное окно текущей кистью (чтобы оно было
   // идентично реальному окну)
   dc_virt.PatBlt(0, 0, maxX, maxY, PATCOPY);
   ReleaseDC(dc);                     // Освобождаем контекст устройства
}
```

Функция `GetSystemMetrics()` описана в *разд. 2.1.1*. Функция `CChildView::create_virt()` практически полностью идентична функции `CChildView::setbmp()` из *разд. 3.1.2* (в этом разделе можно найти описание всех этих функций), за двумя исключениями:

1. Вместо `LoadBitmap()` (загрузка готового рисунка) используется функция создания в памяти битового рисунка:

```
BOOL CBitmap::CreateCompatibleBitmap(     // 0 - ошибка
   CDC* pDC,         // Контекст устройства, с которым должен
быть
                     // совместим битовый рисунок
   int nWidth,       // Ширина рисунка
   int nHeight);     // Высота рисунка
```

2. Для того чтобы виртуальное окно было полностью идентично реальному окну, надо закрасить его текущей кистью, т. к. реальное окно тоже закрашивается текущей кистью по умолчанию.

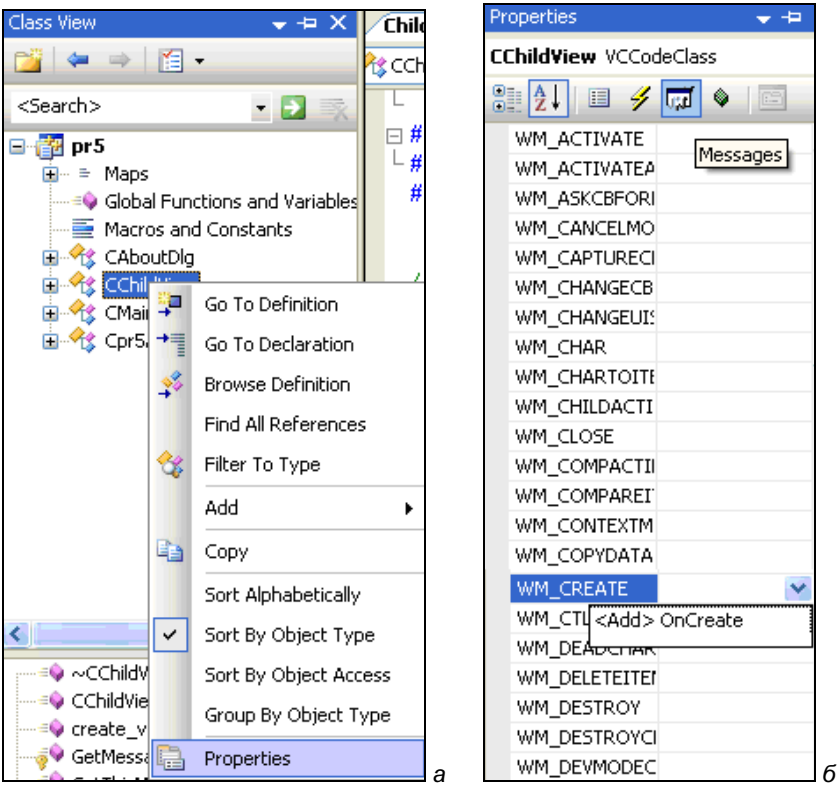
Заполнение заданной прямоугольной области текущей кистью выполняется с помощью функции:

```
BOOL CDC::PatBlt(         // 0 - ошибка
   int x,                 // Координаты левого верхнего угла
   int y,                 // прямоугольной области
   int nWidth,            // Ширина области
```

```
    int nHeight,           // Высота области
    DWORD dwRop);          // Способ применения кисти
```

При этом способы применения кисти (параметр `dwRop`) могут быть такими:

- `PATCOPY` — область заполняется текущей кистью;
- `PATINVERT` — выполняется логическая операция `OR` для цвета области с текущей кистью;
- `DSTINVERT` — выполняется инвертирование цвета области (игнорирование кисти);
- `BLACKNESS` — область заполняется черным цветом (игнорирование кисти);
- `WHITENESS` — область заполняется белым цветом (игнорирование кисти).



а *б*

Рис. 5.1. Открытие окна свойств окна представления (*а*) и добавление обработки сообщения о создании окна (*б*)

Функцию `PatBlt()` с `dwRop = PATCOPY` можно использовать и для "очистки" виртуального окна.

В *разд. 3.1.2* мы вызывали функцию создания битового рисунка `CChildView::setbmp()` сразу после создания окна представления (и кнопок в нем, см. функцию `CMainFrame::OnCreate()`). Теперь сделаем так, чтобы все функции, связанные с окном представления, вызывались и обрабатывались в нем. Для этого добавим обработку сообщения о создании окна (окна представления) `WM_CREATE`. Чтобы это сделать, надо в окне **Class View** вызвать контекстное меню для класса `CChildView` и выполнить команду **Properties**. В окне **Properties** выбрать вкладку **Messages**, в списке сообщений найти **WM_CREATE** и выполнить команду контекстного меню **<Add> OnCreate** (рис. 5.1).

Изменения в файлах приведены в листингах 5.3 и 5.4.

Листинг 5.3. Изменения в файле ChildView.h для обработки сообщения о создании окна

```
// ...
class CChildView : public CWnd
{
// ...
public:
   afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
};
```

Листинг 5.4. Изменения в файле ChildView.cpp для обработки сообщения о создании окна

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_CREATE()
END_MESSAGE_MAP()
// ...
int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CWnd::OnCreate(lpCreateStruct) == -1)
```

```
        return -1;
    // TODO:  Add your specialized creation code here
    return 0;
}
```

Добавим вызов функции создания виртуального окна, вывод в виртуальное окно текста и его копирование в окно представления. Изменения в файле приведены в листинге 5.5.

Листинг 5.5. Изменения в файле ChildView.cpp для создания виртуального окна и вывода в него текста

```
// ...
int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    // TODO:  Add your specialized creation code here
    create_virt();
    dc_virt.TextOutW(10, 10, L"Вывод текста в виртуальное окно");
    return 0;
}
void CChildView::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    // TODO: Add your message handler code here

    // Копирование содержимого виртуального окна (из dc_virt)
    // в реальное окно представления (в dc)
    dc.BitBlt(0, 0, maxX, maxY, &dc_virt, 0, 0, SRCCOPY);
    // Do not call CWnd::OnPaint() for painting messages
}
```

Описание функции `BitBlt()` можно посмотреть в *разд. 3.1.2*. Результат работы программы показан на рис. 5.2.

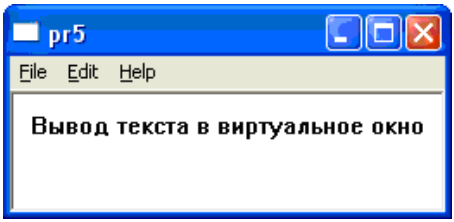


Рис. 5.2. Использование виртуального окна

5.1.2. Масштабирование изображения

Единицами измерения координат при выводе текста и графики являются логические единицы. При выводе информации на конкретное устройство логические единицы преобразуются в физические (пикселы). Процесс преобразования единиц измерения из логических в физические определяется текущим режимом отображения (mapmode). По умолчанию одна логическая единица равна одному пикселу, но соотношение логических и физических единиц можно изменить. Кроме этого, можно задать размеры окна (window) в текущих логических единицах и задать физические размеры области вывода (viewport).

Для работы с масштабированием добавим новые пункты меню **Size | big** и **Size | small** (рис. 5.3) и их обработку в класс окна представления `CChildView` *(см. разд. 4.1.1)*. Для наглядности выведем в окно представления, кроме текста, еще и прямоугольник.

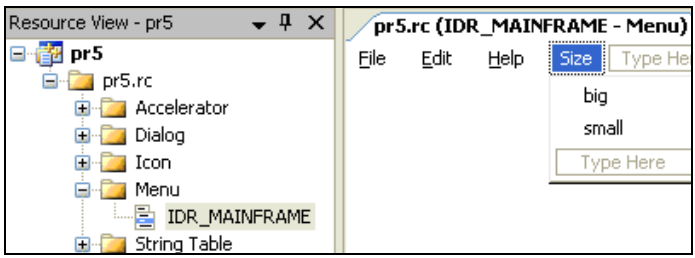


Рис. 5.3. Добавление пунктов меню для масштабирования изображения

Изменения в файлах приведены в листингах 5.6 и 5.7.

Листинг 5.6. Изменения в файле ChildView.h для возможности масштабирования изображения

```
// ...
class CChildView : public CWnd
{
// ...
public:
   int x, y;                        // Размеры области вывода
// ...
public:
   afx_msg void OnSizeBig();
public:
   afx_msg void OnSizeSmall();
};
```

Листинг 5.7. Изменения в файле ChildView.cpp для возможности масштабирования изображения

```
// ...
CChildView::CChildView()
{
   x = 10;
   y = 10;
}

BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_CREATE()
   ON_COMMAND(ID_SIZE_BIG, &CChildView::OnSizeBig)
   ON_COMMAND(ID_SIZE_SMALL, &CChildView::OnSizeSmall)
END_MESSAGE_MAP()

void CChildView::OnPaint()
{
   CPaintDC dc(this);           // device context for painting
   // TODO: Add your message handler code here
```

```
   // Устанавливаем режим отображения
   dc.SetMapMode(MM_ANISOTROPIC);
   // Устанавливаем размеры логической области вывода
   dc.SetWindowExt(500, 500);
   // Устанавливаем размеры физической области вывода
   dc.SetViewportExt(x+500, y+500);

   // Копирование содержимого виртуального окна (из dc_virt)
   // в реальное окно представления (в dc)
   dc.BitBlt(0, 0, maxX, maxY, &dc_virt, 0, 0, SRCCOPY);

// Do not call CWnd::OnPaint() for painting messages
}

int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // ...
   dc_virt.TextOutW(10, 10, L"Вывод текста в виртуальное окно");
   dc_virt.Rectangle(50, 50, 200, 200);
   return 0;
}

// Увеличение изображения
void CChildView::OnSizeBig()
{
   // TODO: Add your command handler code here
   x += 500;
   y += 500;
   Invalidate();              // Перерисовать окно представления
}
// Уменьшение изображения
void CChildView::OnSizeSmall()
{
   // TODO: Add your command handler code here
   x -= 500;
```

```
    y -= 500;
    Invalidate();                // Перерисовать окно представления
}
```

Установка текущего режима отображения выполняется с помощью функции:

```
virtual int CDC::SetMapMode(     // Прежний режим отображения
    int nMapMode);               // Новый режим отображения
```

Значения режимов отображения (`nMapMode`) могут быть такими:

- `MM_ANISOTROPIC` — режим логических единиц, определенный программистом (с помощью функций `SetWindowExt()` и `SetViewportExt()`) с различным масштабированием по X и Y;
- `MM_HIENGLISH` — одна логическая единица равна 0,001 дюйма;
- `MM_HIMETRIC` — одна логическая единица равна 0,01 миллиметра;
- `MM_ISOTROPIC` — аналогично `MM_ANISOTROPIC`, но соотношение X и Y всегда 1:1;
- `MM_LOENGLISH` — одна логическая единица равна 0,01 дюйма;
- `MM_LOMETRIC` — одна логическая единица равна 0,1 миллиметра;
- `MM_TEXT` — одна логическая единица равна 1 пикселу (это режим по умолчанию);
- `MM_TWIPS` — одна логическая единица равна около 1/1440 дюйма.

Настройка размеров логической области вывода (размеры окна в логических единицах) выполняется функцией:

```
virtual CSize CDC::SetWindowExt(     // Старые размеры окна
    int cx,                          // Новые размеры окна
    int cy);
```

или

```
CSize CDC::SetWindowExt(
    SIZE size);
```

Настройка размеров физической области вывода выполняется функцией:

```
virtual CSize CDC::SetViewportExt(   // Старые размеры
    int cx,                          // Новые размеры
    int cy);
```

или

```
CSize CDC::SetViewportExt(
   SIZE size);
```

Функцию `Invalidate()` можно посмотреть в проекте *разд. 3.1.2*. Результат работы программы показан на рис. 5.4.

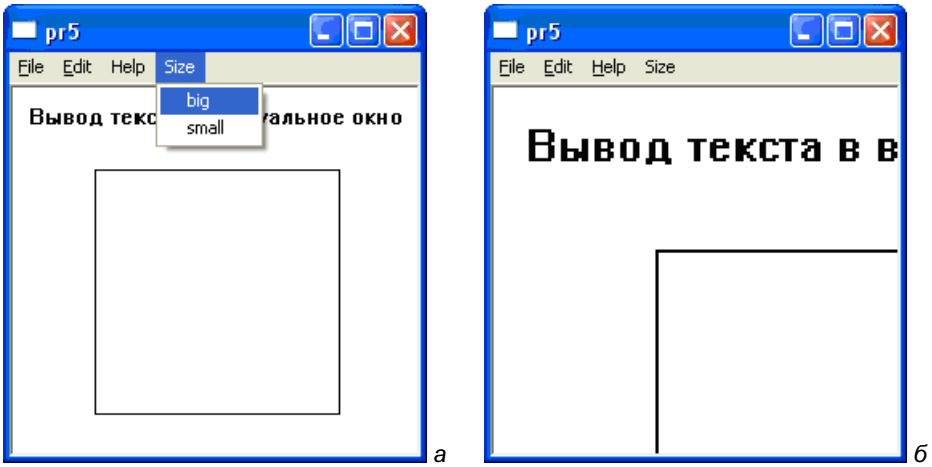


Рис. 5.4. Масштабирование изображения: исходный (*а*) и увеличенный (*б*) размер изображения

5.1.3. Работа с линейкой прокрутки

Теперь, когда изображение в окне можно увеличивать, возникла необходимость добавить линейки прокрутки. Добавим вертикальную линейку (работу с горизонтальной линейкой можно добавить аналогично). Для создании окна представления надо добавить стиль `WS_VSCROLL` (окно имеет вертикальную линейку прокрутки). Стили можно посмотреть в *разд. 1.6.1*, в описании функции CFrameWnd::LoadFrame(). Изменения в файле приведены в листинге 5.8.

Листинг 5.8. Изменения в файле MainFrm.cpp для добавления вертикальной линейки прокрутки

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
   // ...
   // create a view to occupy the client area of the frame
   if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW | WS_VSCROLL ,
                        CRect(0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL))
   // ...
}
```

Добавим обработку сообщения линейки прокрутки в класс представления. Для этого надо в окне **Class View** вызвать контекстное меню для класса `CChildView` и выполнить команду **Properties**. В окне **Properties** выбрать вкладку **Messages**, в списке сообщений найти `WM_VSCROLL` и выполнить команду контекстного меню **<Add> OnVScroll** (рис. 5.5). Добавим переменные для работы с линейкой прокрутки (текущее значение ползунка на линейке прокрутки будет отображаться в прямоугольнике в окне представления).

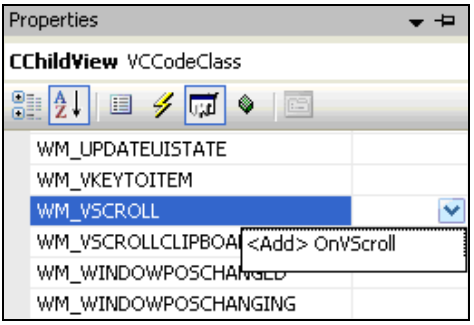


Рис. 5.5. Добавление обработки сообщения вертикальной линейки прокрутки

Изменения в файлах приведены в листингах 5.9 и 5.10.

Листинг 5.9. Изменения в файле ChildView.h для обработки сообщения вертикальной линейки прокрутки

```
// ...
class CChildView : public CWnd
{
```

```
// ...
public:
   int z;                // Текущая позиция ползунка на линейке прокрутки
   CString s;            // Значение z для вывода в окно
// ...
public:
   afx_msg void OnVScroll(UINT nSBCode, UINT nPos,
                          CScrollBar* pScrollBar);
};
```

Листинг 5.10. Изменения в файле ChildView.cpp для обработки сообщения вертикальной линейки прокрутки

```
// ...
CChildView::CChildView()
{
   // ...
   z = 0;
}

BEGIN_MESSAGE_MAP(CChildView, CWnd)
   // ...
   ON_WM_VSCROLL()
END_MESSAGE_MAP()

void CChildView::OnPaint()
{
   // ...
   dc.BitBlt(0,0, maxX, maxY, &dc_virt, 0, 0+z,SRCCOPY);
}

int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // ...
   // Вывод значения начального положения (0) ползунка
   // на линейке прокрутки
```

```
   dc_virt.TextOutW(100,100, L"0");
   return 0;
}

// ...
// Обработка сообщения линейки прокрутки
void CChildView::OnVScroll(UINT nSBCode, UINT nPos,
                           CScrollBar* pScrollBar)
{
   // TODO: Add your message handler code here and/or call default

   // Установка значений линейки (от 0 до 100)
   SetScrollRange(SB_VERT, 0, 100);
   z = nPos;                    // Сохранение текущей позиции ползунка
   switch(nSBCode)              // Что было нажато на линейке
   {
      // Зацепили и потащили ползунок
      case SB_THUMBPOSITION:
                                // Выставить ползунок в текущую позицию
                                SetScrollPos(SB_VERT, nPos);
                                break;
      // Закончили работу с линейкой
      case SB_ENDSCROLL:
                                // Получить текущую позицию ползунка
                                z = GetScrollPos(SB_VERT);
                                break;
      // Нажали на стрелку вверх (наверху линейки)
      case SB_LINEUP:
                                nPos = GetScrollPos(SB_VERT);
                                nPos--;
                                if(nPos < 0)
                                   nPos = 0;
                                SetScrollPos(SB_VERT, nPos);
                                z = nPos;
                                break;
```

```
      // Нажали на стрелку вниз (внизу линейки)
      case SB_LINEDOWN:
                              nPos = GetScrollPos(SB_VERT);
                              nPos++;
                              if(nPos > 100)
                                 nPos = 100;
                              SetScrollPos(SB_VERT, nPos);
                              z = nPos;
                              break;
   }
   // Записать текущую позицию в строку для вывода в окно
   s.Format(L"%d", z);
   // Затереть старую строку
   dc_virt.TextOutW(100,100, L"    ");
   // Записать новую
   dc_virt.TextOutW(100,100, s);
   // Перерисовать окно
   Invalidate();

   CWnd::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

Обработка сообщения вертикальной полосы прокрутки выполняется с помощью следующей функции (для горизонтальной — `OnHScroll()`):

```
afx_msg void CWnd::OnVScroll(
   UINT nSBCode,                  // Код события
   UINT nPos,                     // Текущее положение ползунка
   CScrollBar* pScrollBar);       // Указатель на линейку,
                                  // от которой пришло сообщение
```

Код событий линейки прокрутки (`nSBCode`) может быть такой:

- `SB_BOTTOM` — прокрутка до самой нижней границы линейки (для `OnHScroll()`: `SB_RIGHT` — до самой правой границы);
- `SB_ENDSCROLL` — завершение прокрутки (завершение работы с линейкой);
- `SB_LINEDOWN` — прокрутка на одну позицию вниз (для `OnHScroll()`: `SB_LINERIGHT` — вправо);

- SB_LINEUP — прокрутка на одну позицию вверх (для OnHScroll(): SB_LINELEFT — влево);
- SB_PAGEDOWN — прокрутка на одну страницу вниз (для OnHScroll(): SB_PAGERIGHT — вправо);
- SB_PAGEUP — прокрутка на одну страницу вверх (для OnHScroll(): SB_PAGELEFT — влево);
- SB_THUMBPOSITION — прокрутка до позиции, определяемой параметром nPos;
- SB_THUMBTRACK — перенос ползунка в позицию, указанную в параметре nPos;
- SB_TOP — прокрутка до самой верхней границы (для OnHScroll(): SB_LEFT — до левой границы).

Параметр pScrollBar = NULL, если сообщение пришло от оконной полосы прокрутки, в противном случае это указатель на другой объект прокрутки.

Установить границы линейки прокрутки можно с помощью функции:

```
void CWnd::SetScrollRange(
   int nBar,                // Идентификатор линейки
   int nMinPos,             // Минимальная позиция
   int nMaxPos,             // Максимальная позиция
   BOOL bRedraw = TRUE);    // true - перерисовать линейку, false - нет
```

Идентификатор линейки (nBar) может принимать 2 значения:

- SB_HORZ — горизонтальная линейка;
- SB_VERT — вертикальная линейка.

Получить границы линейки прокрутки можно, используя функцию:

```
void CWnd::GetScrollRange(
   int nBar,              // Идентификатор линейки (см. SetScrollRange())
   LPINT lpMinPos,        // Указатель на минимальное значение
   LPINT lpMaxPos) const; // Указатель на максимальное значение
```

Установить текущее значение ползунка можно функцией:

```
int CWnd::SetScrollPos(
   int nBar,              // Идентификатор линейки (см. SetScrollRange())
   int nPos,              // Устанавливаемое значение
   BOOL bRedraw = TRUE);  // true - перерисовать линейку, false - нет
```

Получить текущее значение ползунка можно с помощью функции:

```
int CWnd::GetScrollPos(    // Текущее значение ползунка
   int nBar) const;        // Идентификатор линейки (см. SetScrollRange())
```

Результаты работы программы показаны на рис. 5.6.

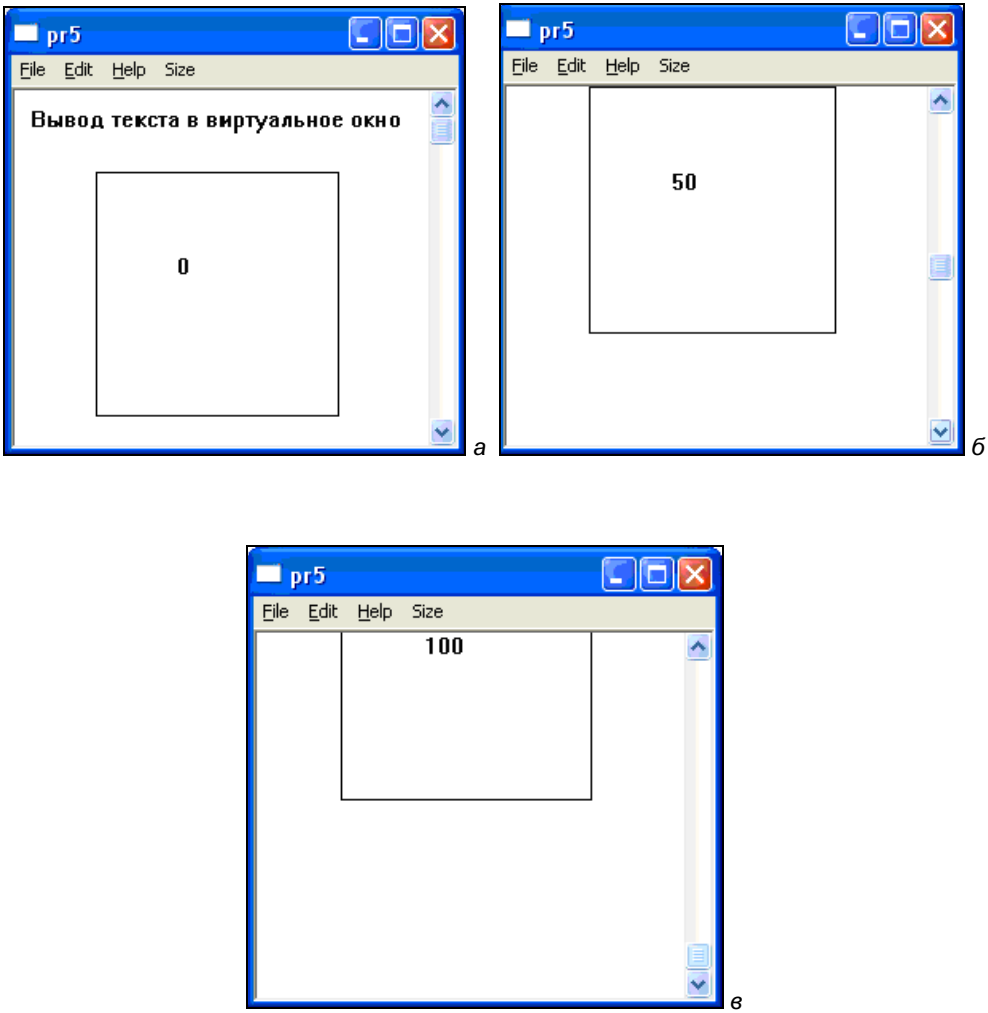


Рис. 5.6. Работа с вертикальной линейкой прокрутки: начальное положение вертикальной линейки прокрутки (*а*), изменение состояния линейки прокрутки (*б*, *в*)

5.1.4. Обработка нажатия клавиш

Добавим в окно представления обработку нажатия клавиш. Для этого надо в окне **Class View** вызвать контекстное меню для класса `CChildView` и выполнить команду **Properties**. В окне **Properties** выбрать вкладку **Messages**, в списке сообщений найти `WM_CHAR` и выполнить команду контекстного меню **<Add> OnChar** (рис. 5.7). Сделаем так, чтобы при нажатии символьной клавиши в окне представления отображался символ (при нажатии на новую клавишу значение старой буква затирается и на ее месте появляется значение новой).

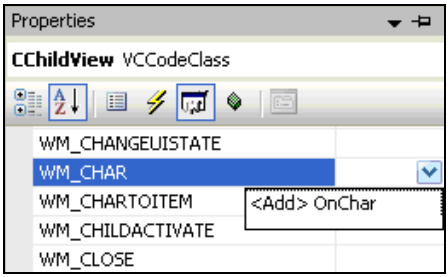


Рис. 5.7. Добавление обработки сообщения нажатия клавиши

Изменения в файлах приведены в листингах 5.11 и 5.12.

Листинг 5.11. Изменения в файле ChildView.h для обработки сообщения клавиатуры

```
// ...
class CChildView : public CWnd
{
// ...
public:
   afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
};
```

Листинг 5.12. Изменения в файле ChildView.cpp для обработки сообщения клавиатуры

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
```

```
// ...
   ON_WM_CHAR()
END_MESSAGE_MAP()

// Обработка нажатия клавиши
void CChildView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
   // TODO: Add your message handler code here and/or call default

   CString ch;                          // Для вывода в окно
   ch.Format(L"%c", nChar);
   // "Затираем" старый символ
   dc_virt.TextOutW(100,150, L"   ");
   // Выводим новый
   dc_virt.TextOutW(100,150, ch);
   // Перерисовать окно
   Invalidate();

   CWnd::OnChar(nChar, nRepCnt, nFlags);
}
```

Обработка нажатия символьных клавиш выполняется с помощью функции:

```
afx_msg void CWnd::OnChar(
   UINT nChar,              // Код символа
   UINT nRepCnt,            // Количество нажатий клавиши
   UINT nFlags);            // Предыдущее состояние клавиши
```

Данная обработка (`OnChar()`) годится только для простых символьных клавиш. Для обработки нажатия специальных клавиш (<Ctrl>, <F1> и т. п.) надо использовать функции, приведенные далее.

Нажатие специальной клавиши обрабатывается функцией:

```
afx_msg void CWnd::OnKeyDown(
   UINT nChar,              // Виртуальный код клавиши
   UINT nRepCnt,            // Количество нажатий клавиши
   UINT nFlags);            // Предыдущее состояние клавиши
```

Отпускание специальной клавиши обрабатывается функцией:

```
afx_msg void CWnd::OnKeyUp(
   UINT nChar,              // Виртуальный код клавиши
   UINT nRepCnt,            // Количество нажатий клавиши
   UINT nFlags);            // Предыдущее состояние клавиши
```

Коды виртуальных клавиш (`nChar`) могут принимать следующие значения:

- `VK_CANCEL` — комбинация клавиш <Ctrl>+<Break>;
- `VK_BACK` — клавиша <BackSpace> (удаление символа перед курсором);
- `VK_TAB` — клавиша <Tab>;
- `VK_RETURN` — клавиша <Enter>;
- `VK_SHIFT` — клавиша <Shift>;
- `VK_CONTROL` — клавиша <Ctrl>;
- `VK_PAUSE` — клавиша <Pause/Break>;
- `VK_CAPITAL` — клавиша <CapsLock>;
- `VK_ESCAPE` — клавиша <Esc>;
- `VK_SPACE` — клавиша <Пробел>;
- `VK_PRIOR` — клавиша <PageUp>;
- `VK_NEXT` — клавиша <PageDown>;
- `VK_END` — клавиша <End>;
- `VK_HOME` — клавиша <Home>;
- клавиши-стрелки:
  - `VK_LEFT` — клавиша со стрелкой влево;
  - `VK_UP` — клавиша со стрелкой вверх;
  - `VK_RIGHT` — клавиша со стрелкой вправо;
  - `VK_DOWN` — клавиша со стрелкой вниз;
- `VK_INSERT` — клавиша <Insert>;
- `VK_DELETE` — клавиша <Delete>;
- `VK_LWIN` — левая клавиша <Windows> (кнопка с изображением логотипа системы флажка, обычно находится между правыми клавишами <Ctrl> и <Alt>);

- `VK_RWIN` — правая клавиша <Windows>;
- `VK_APPS` — клавиша с изображением контекстного меню (обычно находится между правыми клавишами <Windows> и <Ctrl>);
- `VK_SLEEP` — клавиша <Sleep>;
- цифры на дополнительной клавиатуре (при нажатом режиме <NumLock>):
  - `VK_NUMPAD0` — клавиша с цифрой 0;
  - `VK_NUMPAD1` — клавиша с цифрой 1;
  - `VK_NUMPAD2` — клавиша с цифрой 2;
  - `VK_NUMPAD3` — клавиша с цифрой 3;
  - `VK_NUMPAD4` — клавиша с цифрой 4;
  - `VK_NUMPAD5` — клавиша с цифрой 5;
  - `VK_NUMPAD6` — клавиша с цифрой 6;
  - `VK_NUMPAD7` — клавиша с цифрой 7;
  - `VK_NUMPAD8` — клавиша с цифрой 8;
  - `VK_NUMPAD9` — клавиша с цифрой 9;
- функциональные клавиши:
  - `VK_F1` — клавиша <F1>;
  - `VK_F2` — клавиша <F2>;
  - `VK_F3` — клавиша <F3>;
  - `VK_F4` — клавиша <F4>;
  - `VK_F5` — клавиша <F5>;
  - `VK_F6` — клавиша <F6>;
  - `VK_F7` — клавиша <F7>;
  - `VK_F8` — клавиша <F8>;
  - `VK_F9` — клавиша <F9>;
  - `VK_F10` — клавиша <F10>;
  - `VK_F11` — клавиша <F11>;
  - `VK_F12` — клавиша <F12>;

- VK_NUMLOCK — клавиша <NumLock>;
- VK_SCROLL — клавиша <ScrollLock>.

Результат работы программы (при нажатии клавиши <z>) показан на рис. 5.8.

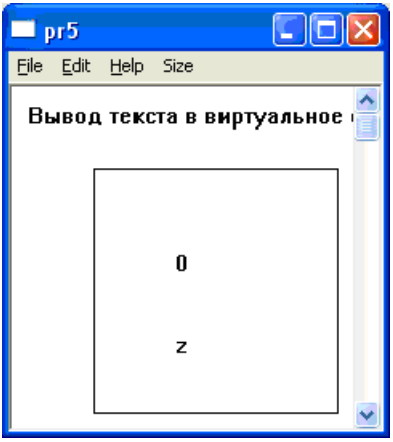


Рис. 5.8. Работа программы
с обработкой нажатия символьных клавиш

Добавим в класс представления обработку сообщения WM_KEYDOWN (рис. 5.9). Сделаем так, чтобы при нажатии на клавиши со стрелками вверх и вниз ползунок линейки прокрутки сдвигался соответственно на одну позицию вверх или вниз (используется обработка SB_LINEUP и SB_LINEDOWN функции CChildView::OnVScroll(), *см. разд. 5.1.3*).

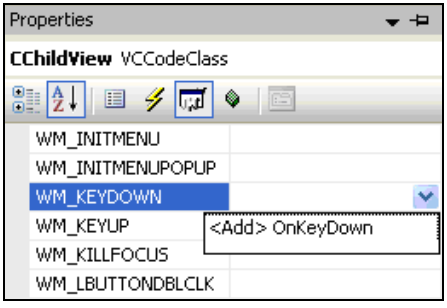


Рис. 5.9. Добавление обработки
сообщения нажатия клавиши

Изменения в файлах приведены в листингах 5.13 и 5.14.

Листинг 5.13. Изменения в файле ChildView.h для обработки сообщения о нажатии специальной клавиши (клавиши со стрелкой вверх или вниз)

```
// ...
class CChildView : public CWnd
{
// ...
public:
   afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
};
```

Листинг 5.14. Изменения в файле ChildView.cpp для обработки сообщения о нажатии специальной клавиши (клавиши со стрелкой вверх или вниз)

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
// ...
   ON_WM_KEYDOWN()
END_MESSAGE_MAP()
// ...
// Обработка нажатия специальных клавиш
void CChildView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
   // TODO: Add your message handler code here and/or call default

   switch(nChar)
   {
      // Нажата клавиша "СТРЕЛКА ВВЕРХ"
      case VK_UP:
                  // Послать сообщение линейке прокрутки для сдвига на
                  //одну позицию вверх
                  SendMessage(WM_VSCROLL, SB_LINEUP);
                  break;
```

```
        // Нажата клавиша "СТРЕЛКА ВНИЗ"
        case VK_DOWN:
                        // Послать сообщение линейке прокрутки для сдвига на
                        // одну позицию вниз
                        SendMessage(WM_VSCROLL, SB_LINEDOWN);
                        break;
    }
    CWnd::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

Послать сообщение в окно `CWnd` можно с помощью функции:

```
LRESULT CWnd::SendMessage(
   UINT message,              // Посылаемое сообщение
   WPARAM wParam = 0,         // Дополнительная информация
   LPARAM lParam = 0);        // Дополнительная информация
```

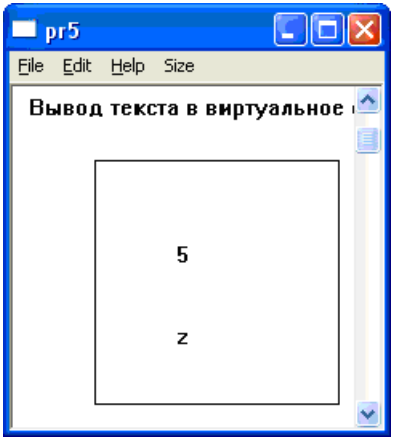


Рис. 5.10. Работа с линейкой прокрутки с помощью клавиш

Функция `SendMessage()` посылает сообщение, вызывая оконную процедуру для указанного окна, и *не возвращает управление*, пока сообщение не будет обработано. Для менее "требовательного" сообщения используется функция:

```
BOOL CWnd::PostMessage(
   UINT message,              // Посылаемое сообщение
```

```
   WPARAM wParam = 0,       // Дополнительная информация
   LPARAM lParam = 0);      // Дополнительная информация
```

Функция `PostMessage()` помещает сообщение в очередь и заканчивает работу без ожидания его обработки.

Результат работы программы показан на рис. 5.10.

5.1.5. Создание дочернего окна

Добавим в меню новый пункт **Child | new** (рис. 5.11) и его обработку в класс окна представления. При выборе данного пункта должно создаваться дочернее окно.

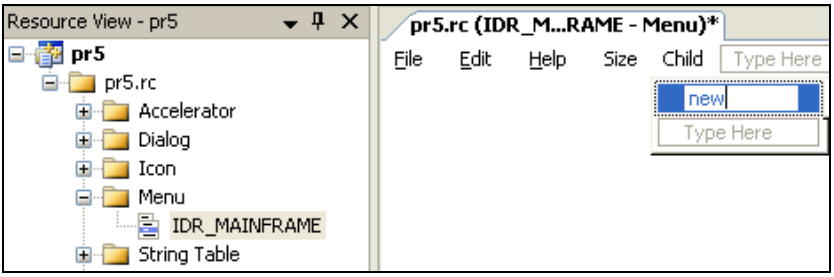


Рис. 5.11. Меню для создания дочернего окна

Изменения в файлах приведены в листингах 5.15 и 5.16.

Листинг 5.15. Изменения в файле ChildView.h для создания нового дочернего окна

```
// ...
class CChildView : public CWnd
{
// ...
public:
   afx_msg void OnChildNew();
};
```

Листинг 5.16. Изменения в файле ChildView.cpp для создания нового дочернего окна

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   // ...
   ON_COMMAND(ID_CHILD_NEW, &CChildView::OnChildNew)
END_MESSAGE_MAP()

// Создание нового окна
void CChildView::OnChildNew()
{
   // TODO: Add your command handler code here
   static CWnd *pwnd;            // Указатель на новое (создаваемое) окно
   if(pwnd)                      // Может быть создано только одно окно
      delete pwnd;
   pwnd = new CWnd;
   // Создание нового окна
   pwnd->Create(NULL, L"Windows", WS_VISIBLE | WS_CHILD |
                WS_OVERLAPPEDWINDOW, CRect( 50,50,500,200 ), this,NULL);
}
```

Результат работы программы показан на рис. 5.12. При выборе пункта меню **Child | new** создается окно. Оно может быть свернуто, развернуто или закрыто. Но оно является "прозрачным" и сохраняет у себя картинку окна, которое находится за ним. Чтобы этого избежать, надо добавить свой класс для окна (чтобы изменить параметры регистрации класса окна и иметь возможность выводить текст в окно).

Для добавления нового класса (`MyChild`) в проект надо:

- в окне **Class View** вызвать контекстное меню для проекта **pr5** и выполнить команду **Add | Class** (Добавить | Класс), как показано на рис. 5.13, *а*;
- в появившемся окне **Add Class** надо выделить в дереве **Categories** (Категории) узел **MFC** (MFC), в правом списке **Templates** (Шаблоны) выбрать **MFC Class** (MFC Класс) и нажать кнопку **Add** (рис. 5.13, *б*);

- в открывшемся окне **MFC Class Wizard** ввести в поле **Class name** (Имя добавляемого класса) "MyChild", в поле **Base class** (Базовый класс) можно выбрать из списка нужный класс, но мы оставим **CWnd**. Так же оставим без изменения имена новых файлов, которые будут добавлены в проект: MyChild.h (объявление нового класса) и MyChild.cpp (определение нового класса) (рис. 5.13, *в*);
- нажать кнопку **Finish**.

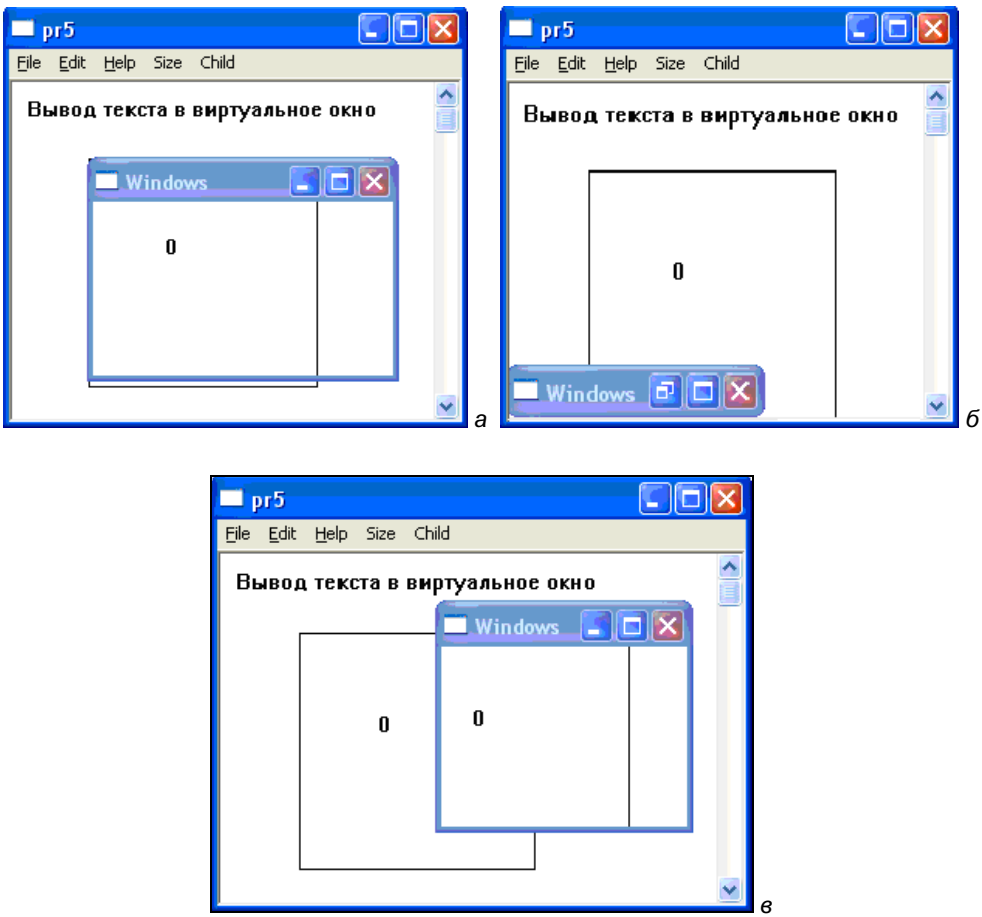
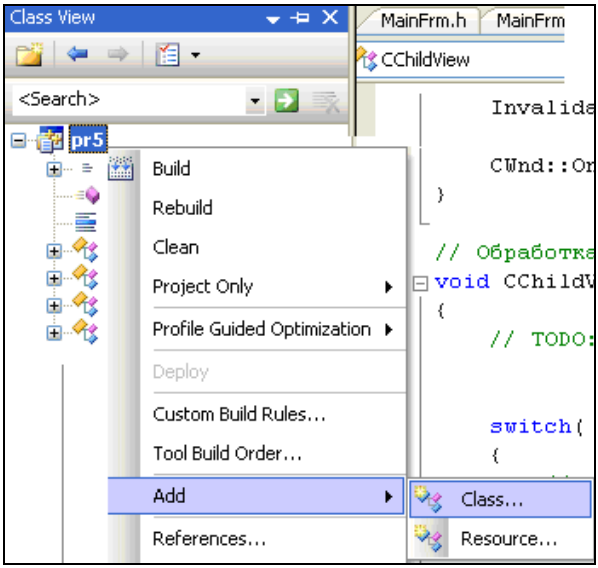
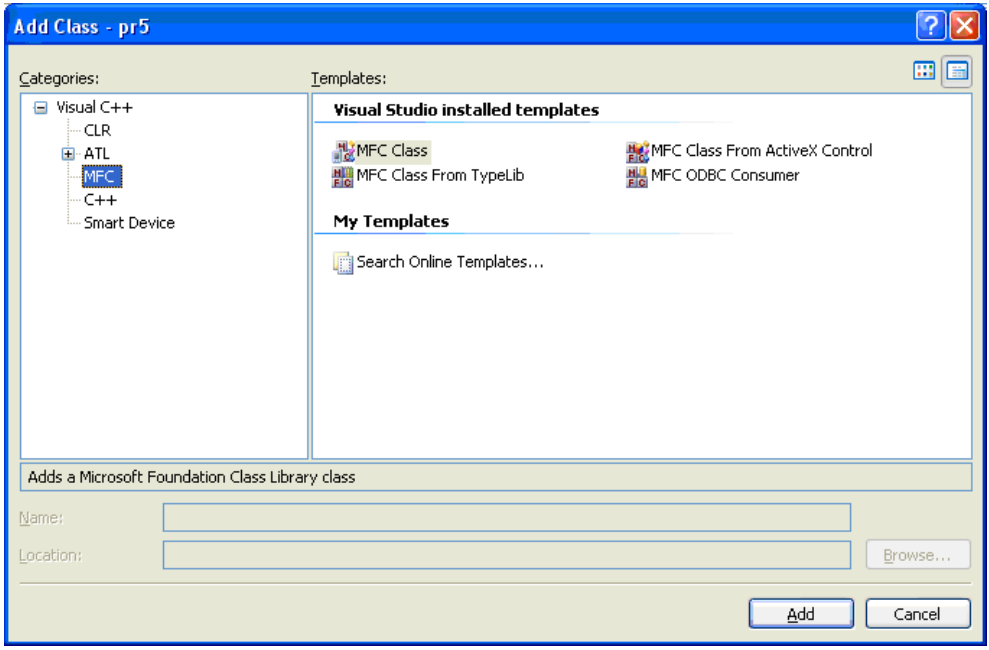


Рис. 5.12. Создание дочернего окна:
при выборе пункта меню **Child | new** (*а*),
при сворачивании дочернего окна (*б*), при перемещении дочернего окна (*в*)

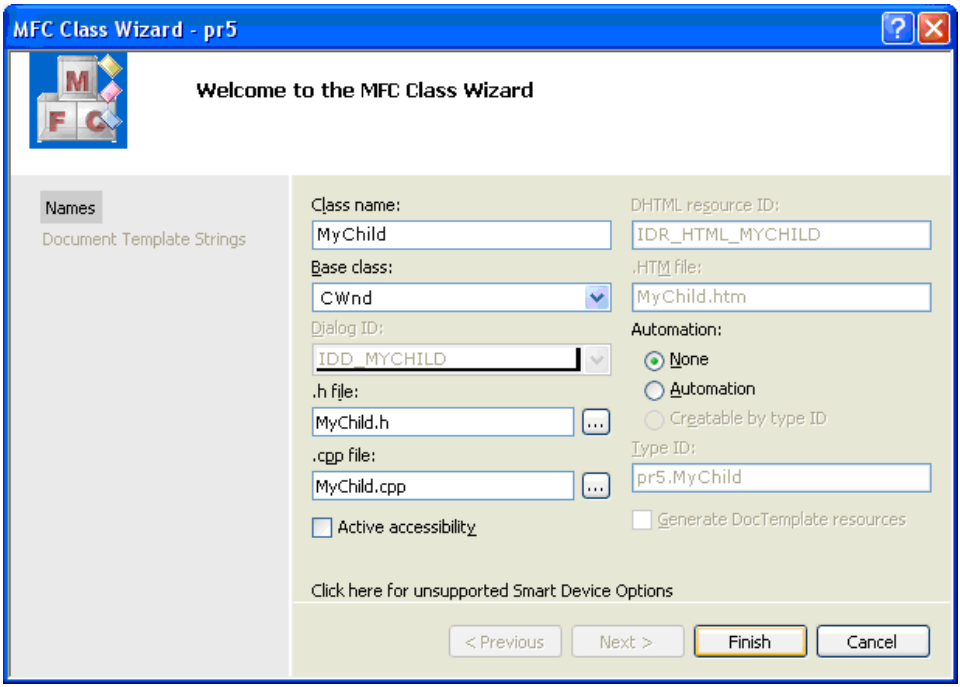


а



б

Рис. 5.13. Добавление нового класса в проект (*а*), выбор типа добавляемого класса (*б*)



в

Рис. 5.13. Выбор нового и базового класса (*в*)

Изменения в программе приведены в листингах 5.17 и 5.18.

Листинг 5.17. Изменения в файле MyChild.h для работы с новым окном

```
#pragma once

// MyChild
class MyChild : public CWnd
{
   DECLARE_DYNAMIC(MyChild)
public:
   MyChild();
   virtual ~MyChild();
protected:
   DECLARE_MESSAGE_MAP()
};
```

Листинг 5.18. Изменения в файле MyChild.cpp для работы с новым окном

```
#include "stdafx.h"
#include "pr5.h"
#include "MyChild.h"

// MyChild
IMPLEMENT_DYNAMIC(MyChild, CWnd)

MyChild::MyChild()
{
}
MyChild::~MyChild()
{
}
BEGIN_MESSAGE_MAP(MyChild, CWnd)
END_MESSAGE_MAP()

// MyChild message handlers
```

Добавим пункт меню **Clild | new2** и его обработку аналогично **new** (рис. 5.14).

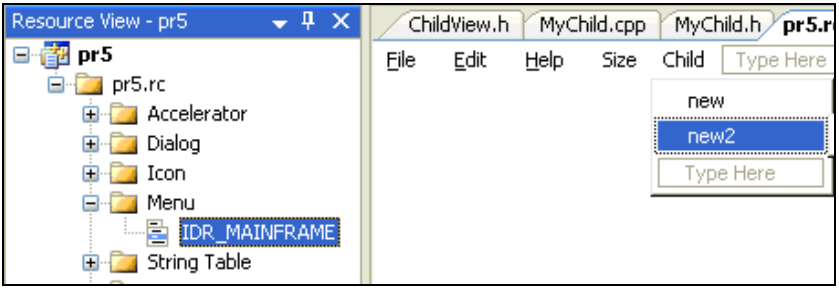


Рис. 5.14. Добавление пункта меню для работы с новым дочерним окном

Изменения в файлах приведены в листингах 5.19 и 5.20.

Листинг 5.19. Изменения в файле ChildView.h для обработки пункта меню создания нового окна

```
// ...
#pragma once
#include "MyChild.h"
class CChildView : public CWnd
{
// ...
public:
   afx_msg void OnChildNew2();
};
```

Листинг 5.20. Изменения в файле ChildView.cpp для обработки пункта меню создания нового окна

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
// ...
   ON_COMMAND(ID_CHILD_NEW2, &CChildView::OnChildNew2)
END_MESSAGE_MAP()
// Создание нового окна
void CChildView::OnChildNew2()
{
   // TODO: Add your command handler code here
   static MyChild *pwnd2;
   if(pwnd2)
      delete pwnd2;
   pwnd2 = new MyChild;
   pwnd2->Create(NULL, L"Windows2",  WS_VISIBLE| WS_CHILD |
                WS_OVERLAPPEDWINDOW, CRect(50,50,500,200), this,NULL);
}
```

Теперь надо переопределить функцию `PreCreateWindow()` — регистрацию класса окна. Для этого надо в окне **Class View** вызвать контекстное меню для класса `MyChild` и выполнить команду **Properties**. В окне **Properties** выбрать

вкладку (шестую слева) **Overrides** (Переопределить), в списке стандартных функций класса найти `PreCreateWindow` и выполнить команду контекстного меню **<Add> PreCreateWindow** (рис. 5.15).

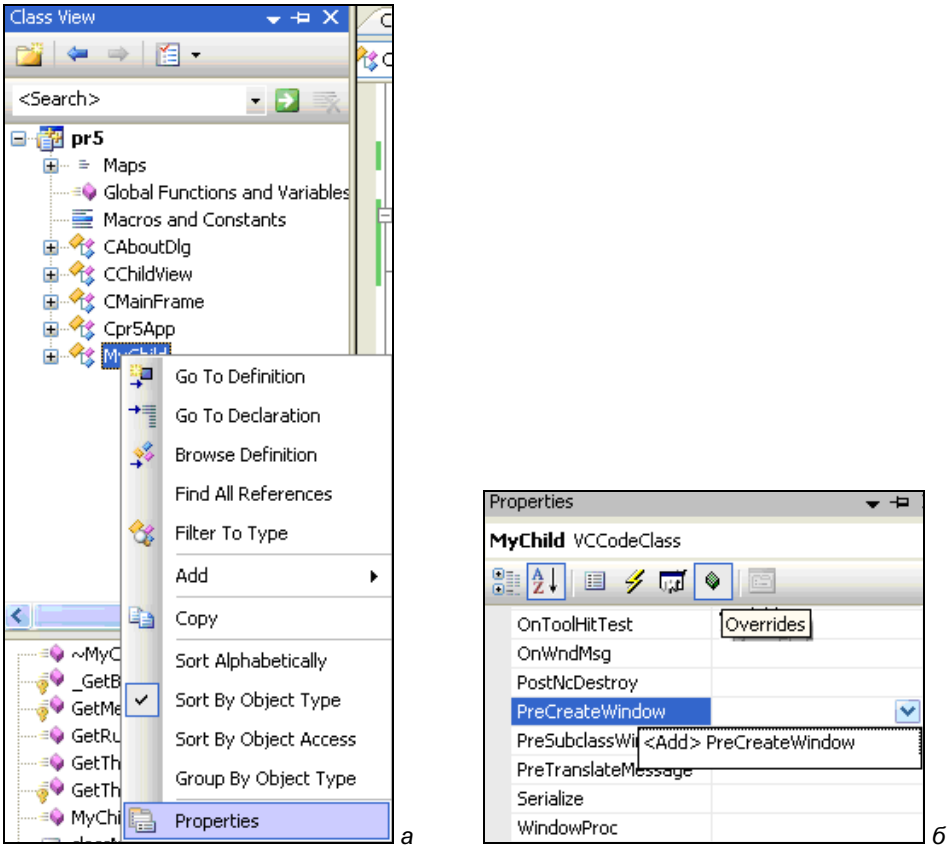


Рис. 5.15. Открытие окна свойств для класса нового окна (*а*) и добавление предопределенной функции регистрации класса этого окна (*б*)

Зададим значения для регистрации класса нового окна. Изменения в файлах приведены в листингах 5.21 и 5.22.

Листинг 5.21. Изменения в файле MyChild.h для задания белого фона нового окна

```
class MyChild : public CWnd
{
```

```
// ...
protected:
   virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
};
```

Листинг 5.22. Изменения в файле MyChild.cpp для задания белого фона нового окна

```
// ...
BOOL MyChild::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Add your specialized code here and/or call the base class
   cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW |
                        CS_DBLCLKS, ::LoadCursor(NULL, IDC_ARROW),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return CWnd::PreCreateWindow(cs);
}
```

Добавим вывод текста в новое окно (обработку сообщения `WM_PAINT`). Для этого надо в окне **Class View** вызвать контекстное меню для класса `MyChild` и выполнить команду **Properties**. В окне **Properties** выбрать вкладку **Messages**, в списке сообщений найти `WM_PAINT` и выполнить команду контекстного меню **<Add> OnPaint** (рис. 5.16).

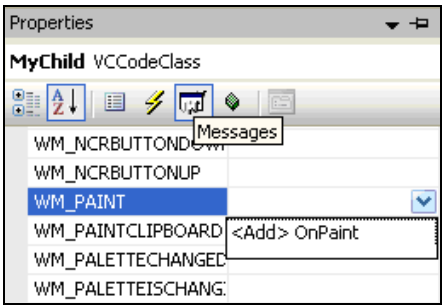


Рис. 5.16. Добавление обработки сообщения о перерисовке окна

Изменения в файлах приведены в листингах 5.23 и 5.24.

Листинг 5.23. Изменения в файле MyChild.h для вывода текста в новое окно

```
class MyChild : public CWnd
{
   // ...
public:
   afx_msg void OnPaint();
};
```

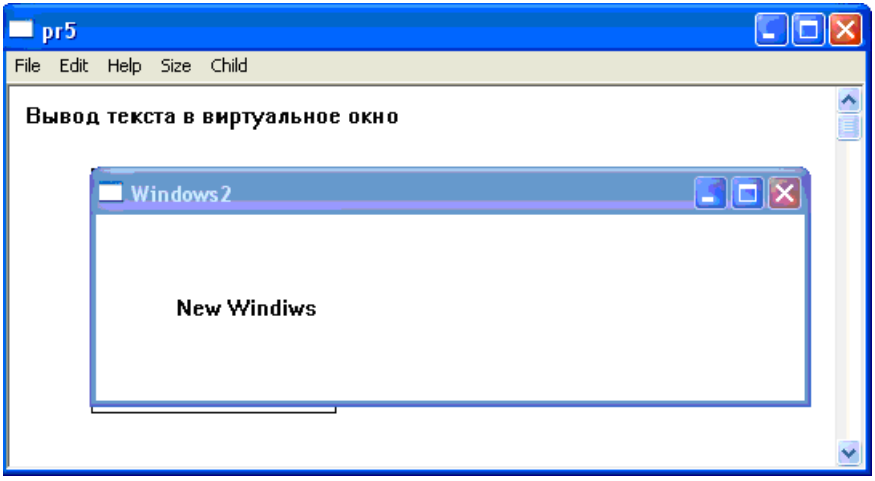


Рис. 5.17. Вывод текста в дочернее окно

Листинг 5.24. Изменения в файле MyChild.cpp для вывода текста в новое окно

```
// ...
BEGIN_MESSAGE_MAP(MyChild, CWnd)
   ON_WM_PAINT()
END_MESSAGE_MAP()
// ...
void MyChild::OnPaint()
{
   CPaintDC dc(this);          // device context for painting
```

```
    // TODO: Add your message handler code here
    // Do not call CWnd::OnPaint() for painting messages
    dc.TextOutW(50,50, L"New Windiws");
}
```

Результат работы программы показан на рис. 5.17.

5.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 5.25. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define ID_SIZE_BIG             32771
#define ID_SIZE_SMALL           32772
#define ID_CHILD_NEW            32773
#define ID_CHILD_NEW2           32774
// ...
```

Листинг 5.26. Файл ChildView.h (объявление класса представления)

```
// ...
#pragma once
#include "MyChild.h"
class CChildView : public CWnd
{
// Construction
public:
    CChildView();
    CBitmap virt_wnd;           // Битовый рисунок - виртуальное окно
    CDC dc_virt;                // Контекст виртуального окна
    int maxX, maxY;             // Ширина и высота экрана
    void create_virt(void);     // Создание виртуального окна
```

```
   int x, y;                    // Размеры области вывода
   int z;                       // Текущая позиция ползунка на линейке
                                // прокрутки
   CString s;                   // Значение z для вывода в окно
// ...
protected:
   afx_msg void OnPaint();
   DECLARE_MESSAGE_MAP()
public:
   afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
public:
   afx_msg void OnSizeBig();
public:
   afx_msg void OnSizeSmall();
public:
   afx_msg void OnVScroll(UINT nSBCode, UINT nPos,
                          CScrollBar* pScrollBar);
public:
   afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
public:
   afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
public:
   afx_msg void OnChildNew();
public:
   afx_msg void OnChildNew2();
};
```

Листинг 5.27. Файл ChildView.cpp (определение класса представления)

```
// ...
CChildView::CChildView()
{
   x = 10;
   y = 10;
   z = 0;
}
```

```
CChildView::~CChildView()
{
   dc_virt.DeleteDC();
}
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_CREATE()
   ON_COMMAND(ID_SIZE_BIG, &CChildView::OnSizeBig)
   ON_COMMAND(ID_SIZE_SMALL, &CChildView::OnSizeSmall)
   ON_WM_VSCROLL()
   ON_WM_CHAR()
   ON_WM_KEYDOWN()
   ON_COMMAND(ID_CHILD_NEW, &CChildView::OnChildNew)
   ON_COMMAND(ID_CHILD_NEW2, &CChildView::OnChildNew2)
END_MESSAGE_MAP()
// ...
void CChildView::OnPaint()
{
   CPaintDC dc(this);        // device context for painting
   // TODO: Add your message handler code here

   // Устанавливаем режим отображения
   dc.SetMapMode(MM_ANISOTROPIC);
   // Устанавливаем размеры логической области вывода
   dc.SetWindowExt(500, 500);
   // Устанавливаем размеры физической области вывода
   dc.SetViewportExt(x+500, y+500);
   // Копирование содержимого виртуального окна (из dc_virt)
   // в реальное окно представления (в dc)
   dc.BitBlt(0,0, maxX, maxY, &dc_virt, 0, 0+z,SRCCOPY);

   // Do not call CWnd::OnPaint() for painting messages
}

void CChildView::create_virt( void )
{
```

```
   // Получение размеров экрана
   maxX = GetSystemMetrics(SM_CXSCREEN);
   maxY = GetSystemMetrics(SM_CYSCREEN);
   //Строим растр (виртуальное окно) совместимый с реальным окном
   CDC *dc;                          // Контекст устройства окна
   dc = GetDC();                     // Получаем контекст устройства окна
   dc_virt.CreateCompatibleDC(dc);   // Строим растр, совместимый с окном
   // Создаем в памяти растровое изображение (виртуальное окно),
   // совместимое с окном представления
   virt_wnd.CreateCompatibleBitmap(dc, maxX, maxY);
   dc_virt.SelectObject(&virt_wnd);  // Подключаем виртуальное окно
   // Закрашиваем виртуальное окно текущей кистью (чтобы оно было
   // идентично реальному окну)
   dc_virt.PatBlt(0, 0, maxX, maxY, PATCOPY);
   ReleaseDC(dc);                    // Освобождаем контекст устройства
}

int CChildView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CWnd::OnCreate(lpCreateStruct) == -1)
      return -1;

   // TODO:  Add your specialized creation code here
   create_virt();
   dc_virt.TextOutW(10, 10, L"Вывод текста в виртуальное окно");
   dc_virt.Rectangle(50, 50, 200, 200);
   dc_virt.TextOutW(100,100, L"0");
   return 0;
}

// Увеличение изображения
void CChildView::OnSizeBig()
{
   // TODO: Add your command handler code here
   x += 500;
```

```
   y += 500;
   Invalidate();                    // Перерисовать окно представления.
}

// Уменьшение изображения
void CChildView::OnSizeSmall()
{
   // TODO: Add your command handler code here
   x -= 500;
   y -= 500;
   Invalidate();                    // Перерисовать окно представления
}

// Обработка сообщения линейки прокрутки
void CChildView::OnVScroll(UINT nSBCode, UINT nPos,
                           CScrollBar* pScrollBar)
{
   // TODO: Add your message handler code here and/or call default

   // Установка значений линейки (от 0 до 100)
   SetScrollRange(SB_VERT, 0, 100);
   z = nPos;                        // Сохранение текущей позиции ползунка
   switch(nSBCode)                  // Что было нажато на линейке
   {
      // Зацепили и потащили ползунок
      case SB_THUMBPOSITION:
                                    // Выставить ползунок в текущую позицию
                                    SetScrollPos(SB_VERT, nPos);
                                    break;
      // Закончили работу с линейкой
      case SB_ENDSCROLL:
                                    // Получить текущую позицию ползунка
                                    z = GetScrollPos(SB_VERT);
                                    break;
```

```
        // Нажали на стрелку вверх (наверху линейки)
        case SB_LINEUP:
                                nPos = GetScrollPos(SB_VERT);
                                nPos--;
                                if(nPos < 0)
                                   nPos = 0;
                                SetScrollPos(SB_VERT, nPos);
                                z = nPos;
                                break;
        // Нажали на стрелку вниз (внизу линейки)
        case SB_LINEDOWN:
                                nPos = GetScrollPos(SB_VERT);
                                nPos++;
                                if(nPos > 100)
                                   nPos = 100;
                                SetScrollPos(SB_VERT, nPos);
                                z = nPos;
                                break;
    }
    // Записать текущую позицию в строку для вывода в окно
    s.Format(L"%d", z);
    // Затереть старую строку
    dc_virt.TextOutW(100,100, L"    ");
    // Записать новую
    dc_virt.TextOutW(100,100, s);
    // Перерисовать окно
    Invalidate();

    CWnd::OnVScroll(nSBCode, nPos, pScrollBar);
}

// Обработка нажатия клавиши
void CChildView::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
```

```
    CString ch;                                  // Для вывода в окно
    ch.Format(L"%c", nChar);
    // "Затираем" старый символ
    dc_virt.TextOutW(100,150, L"  ");
    // Выводим новый
    dc_virt.TextOutW(100,150, ch);
    // Перерисовать окно
    Invalidate();

    CWnd::OnChar(nChar, nRepCnt, nFlags);
}

// Обработка нажатия специальных клавиш
void CChildView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
    switch(nChar)
    {
       // Нажата клавиша "СТРЕЛКА ВВЕРХ"
       case VK_UP:
                       // Послать сообщение линейке прокрутки
                       // для сдвига на одну позицию вверх
                       SendMessage(WM_VSCROLL, SB_LINEUP);
                       break;
       // Нажата клавиша "СТРЕЛКА ВНИЗ"
       case VK_DOWN:
                       // Послать сообщение линейке прокрутки
                       // для сдвига на одну позицию вниз
                       SendMessage(WM_VSCROLL, SB_LINEDOWN);
                       break;
    }

    CWnd::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

```
// Создание нового окна
void CChildView::OnChildNew()
{
   // TODO: Add your command handler code here
   static CWnd *pwnd;            // Указатель на новое (создаваемое) окно
   if(pwnd)                      // Может быть создано только одно окно
      delete pwnd;
   pwnd = new CWnd;
   // Создание нового окна
   pwnd->Create(NULL, L"Windows", WS_VISIBLE | WS_CHILD |
                WS_OVERLAPPEDWINDOW, CRect( 50,50,500,200 ), this,NULL);
}

// Создание нового окна с выводом в него текста
void CChildView::OnChildNew2()
{
   // TODO: Add your command handler code here
   static MyChild *pwnd2;
   if(pwnd2)
      delete pwnd2;
   pwnd2 = new MyChild;
   pwnd2->Create(NULL, L"Windows2", WS_VISIBLE| WS_CHILD |
                WS_OVERLAPPEDWINDOW, CRect( 50,50,500,200 ), this, NULL);
}
```

Листинг 5.28. Файл MyChild.h (объявление класса дочернего окна), добавленный в проект с помощью мастера

```
#pragma once

// MyChild
class MyChild : public CWnd
{
   DECLARE_DYNAMIC(MyChild)
```

```
public:
   MyChild();
   virtual ~MyChild();
protected:
   DECLARE_MESSAGE_MAP()
protected:
   virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
public:
   afx_msg void OnPaint();
};
```

Листинг 5.29. Файл MyChild.cpp (определение класса дочернего окна), добавленный в проект с помощью мастера

```
// MyChild.cpp : implementation file
//
#include "stdafx.h"
#include "pr5.h"
#include "MyChild.h"

// MyChild
IMPLEMENT_DYNAMIC(MyChild, CWnd)

MyChild::MyChild()
{
}

MyChild::~MyChild()
{
}

BEGIN_MESSAGE_MAP(MyChild, CWnd)
   ON_WM_PAINT()
END_MESSAGE_MAP()
```

```
// MyChild message handlers

void MyChild::OnPaint()
{
   CPaintDC dc(this          // device context for painting
   // TODO: Add your message handler code here
   // Do not call CWnd::OnPaint() for painting messages
   dc.TextOutW(50,50, L"New Windiws");
}

BOOL MyChild::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Add your specialized code here and/or call the base class
cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS,
                        ::LoadCursor(NULL, IDC_ARROW),
                         reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);

   return CWnd::PreCreateWindow(cs);
}
```

ГЛАВА 6

Основные элементы управления диалоговых окон

Создадим новый проект, на примере которого рассмотрим способы создания диалоговых окон, добавления к ним элементов управления и работу с ними.

Создайте проект **pr6** аналогично проекту **pr1** *(см. разд. 1.1)*. Надо изменить следующие опции относительно изначально предложенных мастером:

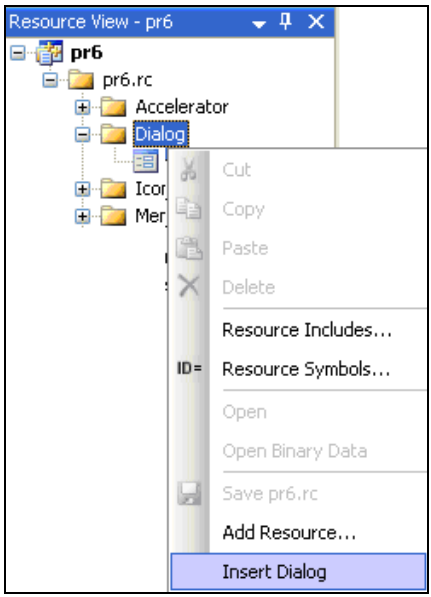
- на вкладке **Application Type**:
  - SDI-документ;
  - без поддержки архитектуры документ/представление;
- на вкладке **User Interface Features**:
  - без строки статуса;
  - без панели инструментов.

6.1. Описание программы

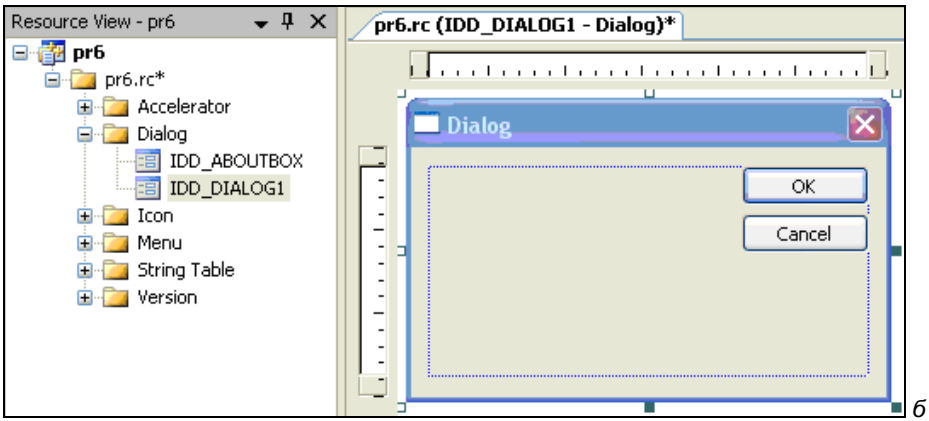
6.1.1. Добавление окна диалога

Добавим новый ресурс диалогового окна. Для этого надо в файле ресурсов вызвать контекстное меню для ресурса **Dialog** и выполнить команду **Insert Dialog** (Вставить диалог) (рис. 6.1, *а*). После этого в ресурсах появится новое окно диалога (по умолчанию ему дается идентификатор `IDD_DIALOG1`) (рис. 6.1, *б*). После создания нового ресурса диалога откроется окно **Toolbox** (Инструменты) с элементами управления, которые можно добавлять в окно

диалога (рис. 6.2). Для добавления элемента надо "захватить" его левой кнопкой мыши в окне **Toolbox** и перетащить в окно диалога.



а



б

Рис. 6.1. Добавление ресурса диалога с помощью контекстного меню (*а*) и шаблон нового окна диалога (*б*)

Далее перечислены элементы управления, а также приведены названия классов для тех элементов управления, которые будут рассмотрены в этой главе (остальные элементы будут подробно рассмотрены в следующих главах), и

изображения, как будут выглядеть элементы управления при их установке в окно диалога:

- **Button** — кнопка (`class CButton : public CWnd`);

- **Check Box** — флажок (для установки или снятия определенного параметра) (`class CButton : public CWnd`);

- **Edit Control** — текстовое поле (позволяет вводить и редактировать текст) (`class CEdit : public CWnd`);

- **Combo Box** — поле с раскрывающимся списком (позволяет ввести в поле или выбрать в раскрывающемся списке нужное значение) (`class CComboBox : public CWnd`);

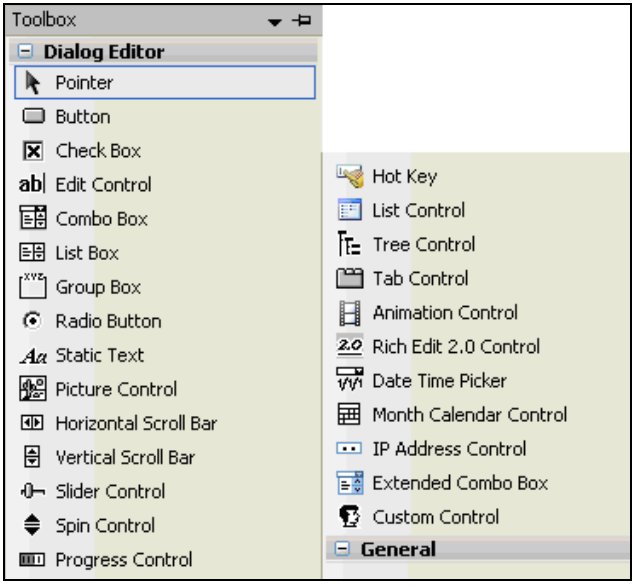
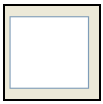
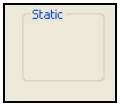


Рис. 6.2. Окно **Toolbox** с элементами управления

- **List Box** — список (позволяет выбрать в списке нужное значение) (`class CListBox : public CWnd`);



- **Group Box** — групповой блок (рамка с надписью для группирования элементов управления) (`class CStatic : public CWnd`);



- **Radio Button** — переключатель (для выбора одного из заданных значений) (`class CButton : public CWnd`);

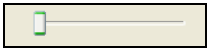
- **Static Text** — надпись (небольшой поясняющий текст) (`class CStatic : public CWnd`);

- **Picture Control** — рисунок;

- **Horizontal Scroll Bar** — горизонтальная полоса прокрутки (позволяет настраивать значения в диапазоне, например, интенсивность цвета, скорость перемещения курсора мыши и т. п.).

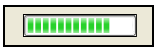
- **Vertical Scroll Bar** — вертикальная полоса прокрутки;

- **Slider Control** — регулятор (шкала с бегунком для выбора нужного значения);



- **Spin Control** — счетчик (пара кнопок со стрелками вверх и вниз для выбора нужного значения);

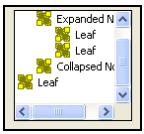
- **Progress Control** — индикатор (показывает степень завершенности определенного процесса);



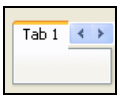
- **Hot Key** — быстрая клавиша (предоставляет пользователю окно для ввода комбинаций быстрых клавиш);

- **List Control** — список (набор значков и текста для выбора нужного значения);

- **Tree Control** — дерево (для выбора нужного объекта);



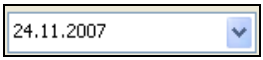
- **Tab Control** — набор вкладок (для выбора в одном окне диалога разных вкладок — независимых страниц);



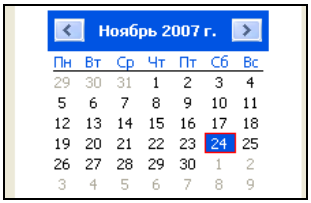
- **Animation Control** — анимация (небольшая область, отображающая видеоклип);

- **Rich Edit 2.0 Control** — расширенное текстовое поле (с возможностью форматирования символов и абзацев текста);

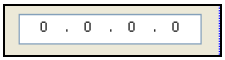
- **Date Time Picker** — дата и время (поле для ввода даты и времени);



- **Month Calendar Control** — календарь (для выбора даты);



- **IP Address Control** — IP-адрес (поле с маской для ввода IP-адреса);

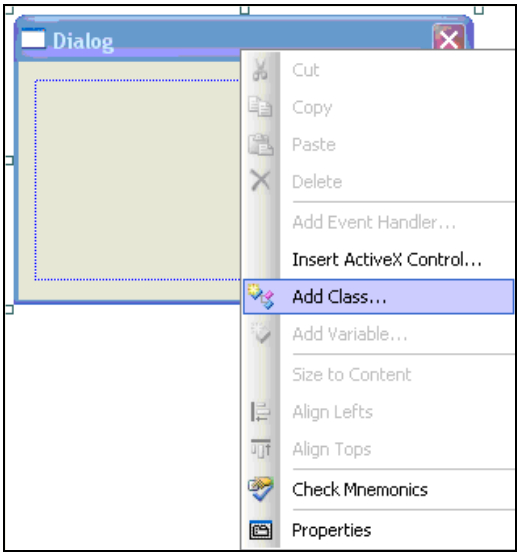


- **Extended Combo Box** — расширенное поле с раскрывающимся списком (можно в качестве элементов списка использовать рисунки);

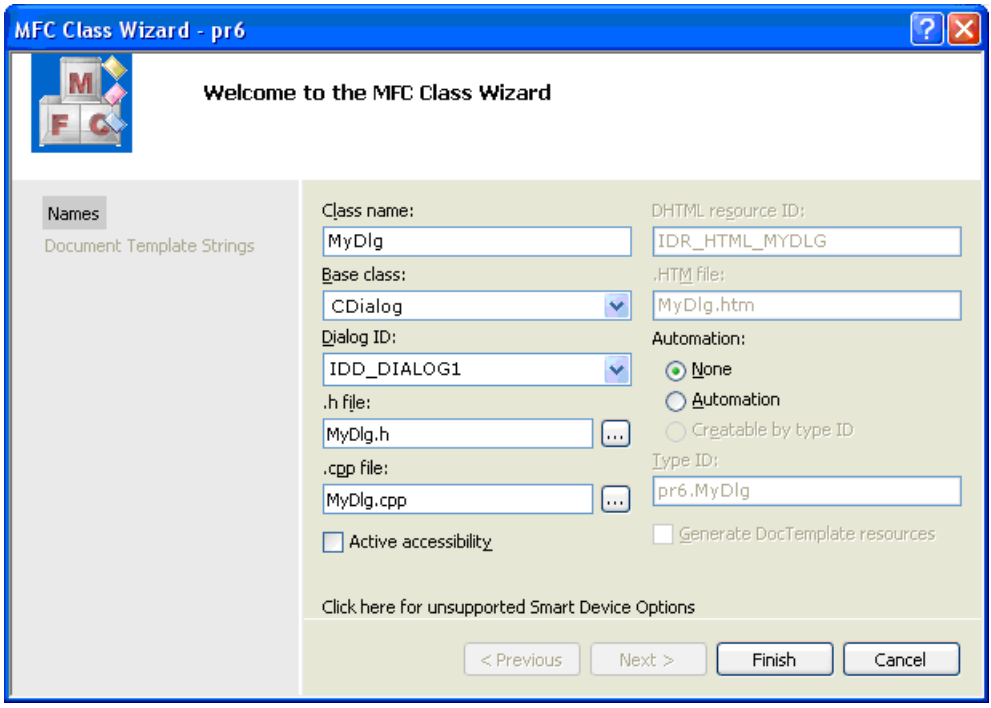
- **Custom Control** — пользовательский элемент (элемент управления, созданный и запрограммированный пользователем).

Чтобы можно было работать с окном диалога, надо создать для него класс. Для этого необходимо сделать следующее:

- вызвать контекстное меню для нового окна диалога и выполнить команду **Add Class** (Добавить класс) (рис. 6.3, *а*);
- в появившемся окне **Add Class** надо в дереве **Categories** выбрать **MFC**, в списке **Templates** выбрать **MFC Class** и нажать кнопку **Add** (см. рис. 5.13, *б*);
- в открывшемся окне **MFC Class Wizard** надо задать (рис. 6.3, *б*):
  - в поле **Class name** ввести имя класса "MyDlg";
  - в поле **Base class** оставить значение по умолчанию **CDialog**;
  - в поле **Dialog ID** (Идентификатор диалога) оставить значение по умолчанию **IDD_DIALOG1**;
  - оставить файлы по умолчанию — MyDlg.h (объявление класса диалога) и MyDlg.cpp (определение класса диалога);
- нажать кнопку **Finish**.



а



б

Рис. 6.3. Добавление класса окна диалога с помощью контекстного меню (*а*) и выбор значений для класса диалога (*б*)

Изменения в программе и в файлах показаны в листингах 6.1—6.3.

Листинг 6.1. Изменения в файле Resource.h для работы с окном диалога

```
// ...
#define IDD_DIALOG1                 130
// ...
```

Листинг 6.2. Файл MyDlg.h для работы с окном диалога

```
#pragma once

// MyDlg dialog
class MyDlg : public CDialog
{
   DECLARE_DYNAMIC(MyDlg)
public:
   MyDlg(CWnd* pParent = NULL);                  // standard constructor
   virtual ~MyDlg();
// Dialog Data
   enum { IDD = IDD_DIALOG1 };
protected:
   virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

   DECLARE_MESSAGE_MAP()
};
```

Листинг 6.3. Файл MyDlg.cpp для работы с окном диалога

```
// MyDlg.cpp : implementation file
//
#include "stdafx.h"
#include "pr6.h"
#include "MyDlg.h"
```

```
// MyDlg dialog
IMPLEMENT_DYNAMIC(MyDlg, CDialog)

MyDlg::MyDlg(CWnd* pParent /*=NULL*/): CDialog(MyDlg::IDD, pParent)
{
}
MyDlg::~MyDlg()
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(MyDlg, CDialog)
END_MESSAGE_MAP()

// MyDlg message handlers
```

Для работы с окном диалога существует класс:

```
class CDialog : public CWnd
```

Конструктор класса выглядит следующим образом:

```
explicit CDialog::CDialog(
   LPCTSTR lpszTemplateName,     // Указатель на строку с именем шаблона
   CWnd* pParentWnd = NULL);     // Указатель на родительское окно
```

или

```
explicit CDialog::CDialog(
   UINT nIDTemplate,             // Номер идентификатора ресурса
   CWnd* pParentWnd = NULL);
```

или

```
CDialog::CDialog();
```

Если `pParentWnd = NULL`, то родительским окном является главное окно приложения.

Функция `DoDataExchange()` вызывается автоматически для обмена данными (и для проверки корректности данных), передаваемых между объектами классов диалогового окна и включенных в него объектов классов элементов управления (переключателей, полей редактирования, списков и т. п.):

```
virtual void CWnd::DoDataExchange(
   CDataExchange* pDX);              // Указатель на изменяемый объект
```

Добавим пункт меню **My | Dlg** (рис. 6.4), по которому будет появляться окно диалога, и его обработку в класс представления `CChildView` (добавление обработки выбора пункта меню *см. в разд. 4.1.1*).

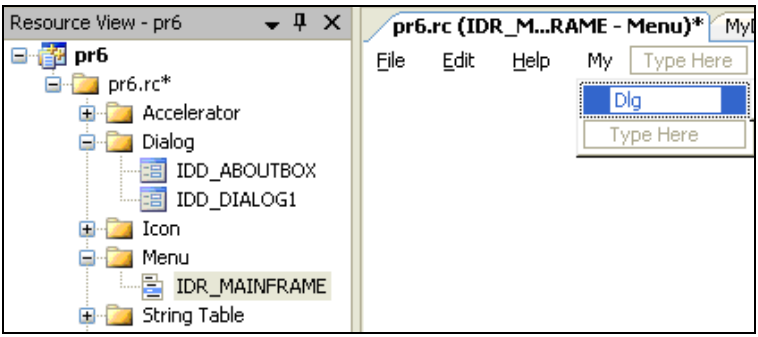


Рис. 6.4. Добавление пункта меню для вызова окна диалога

Изменения в программе показаны в листингах 6.4—6.6.

Листинг 6.4. Изменения в файле Resource.h для обработки пункта меню открытия окна диалога

```
// ...
#define ID_MY_DLG                       32771
// ...
```

Листинг 6.5. Изменения в файле ChildView.h для обработки пункта меню открытия окна диалога

```
// ...
class CChildView : public CWnd
{
```

```
   // ...
public:
   afx_msg void OnMyDlg();
};
```

Листинг 6.6. Изменения в файле ChildView.cpp для обработки пункта меню открытия окна диалога

```
// ...
#include "ChildView.h"
#include "MyDlg.h"
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_COMMAND(ID_MY_DLG, &CChildView::OnMyDlg)
END_MESSAGE_MAP()
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;                  // Объект окна диалога
   INT_PTR nRet;
   nRet = dlg.DoModal();       // Создание окна диалога
   switch (nRet)               // Как было закрыто окно диалога
   {
      case IDOK:               // Кнопкой OK
                  AfxMessageBox(L"OK");
                  break;
      case IDCANCEL:           // Кнопкой Cancel
                  AfxMessageBox(L"CANCEL");
                  break;
   }
}
```

Для создания окна модального диалога (приложение не будет ни на что реагировать, пока окно диалога не будет закрыто) существует функция

`DoModal()`, которая возвращает идентификатор нажатой кнопки в окне диалога или `-1` при ошибке создания окна:

```
virtual INT_PTR CDialog::DoModal();
```

Основные идентификаторы нажатых кнопок:

- `IDOK` — была нажата кнопка **OK**;
- `IDCANCEL` — пользователь выполнил одно из действий:
  - нажал кнопку **Cancel**;
  - нажал кнопку  в левом верхнем углу окна диалога;
  - нажал клавишу <Esc>;
  - нажал комбинацию клавиш <Ctrl>+<Break>.

Результаты работы программы показаны на рис. 6.5.

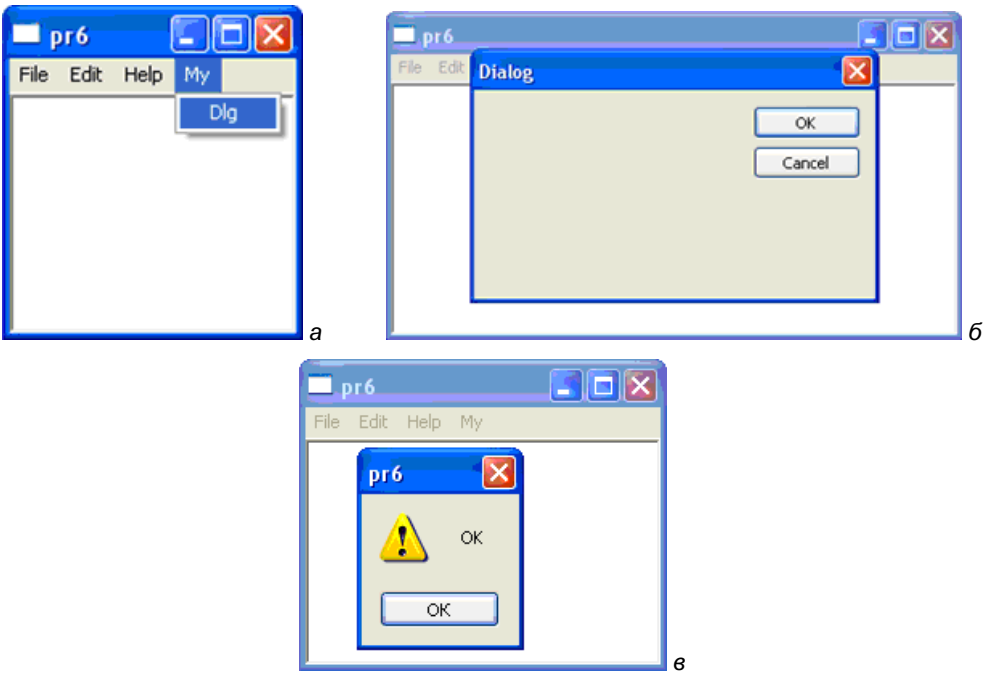


Рис. 6.5. Вызов модального диалога с помощью меню (*а*), вид модального диалога (*б*) и обработка нажатия кнопки **OK** при закрытии диалога (*в*)

Для создания окна немодального диалога существует функция:

```
virtual BOOL CDialog::Create(
   LPCTSTR lpszTemplateName,     // Указатель на строку с именем шаблона
   CWnd* pParentWnd = NULL);     // Указатель на родительское окно
```

или

```
virtual BOOL CDialog::Create(
   UINT nIDTemplate,             // Номер идентификатора ресурса
   CWnd* pParentWnd = NULL);
```

После создания немодального диалога надо обязательно вызвать функцию `CWnd::ShowWindow()` (т. к. немодальный диалог, в отличие от модального, не становится видимым автоматически). Для создания немодального диалога нельзя пользоваться локальными переменными, т. к. он (в отличие от модального) не ожидает закрытия окна диалога и отдает управление программе дальше (локальная переменная выходит за пределы времени жизни, и диалог удаляется, вы его даже не успеете рассмотреть). Для нашей программы можно было создать немодальный диалог следующим образом:

```
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   static MyDlg *pdl;
   if(pdl)
      delete pdl;
   pdl = new MyDlg;
   // Можно и так: pdl->Create(MyDlg::IDD);
   pdl->Create(IDD_DIALOG1);
   pdl->ShowWindow(SW_SHOW);
}
```

Результат работы такого немодального диалога показан на рис. 6.6. Сначала был открыт немодальный диалог, а потом (не закрывая его) модальный диалог из меню **Help | About**.

Как изменить заголовок и размер окна диалога, названия и размер кнопок, было описано в *разд. 1.7.2*. К диалогу можно подключить меню. Для этого в окне свойств диалога надо выбрать в поле **Menu** идентификатор ресурса меню (рис. 6.7). В меню будут работать только построенные мастером пункты, для своих пунктов меню надо добавлять обработку.

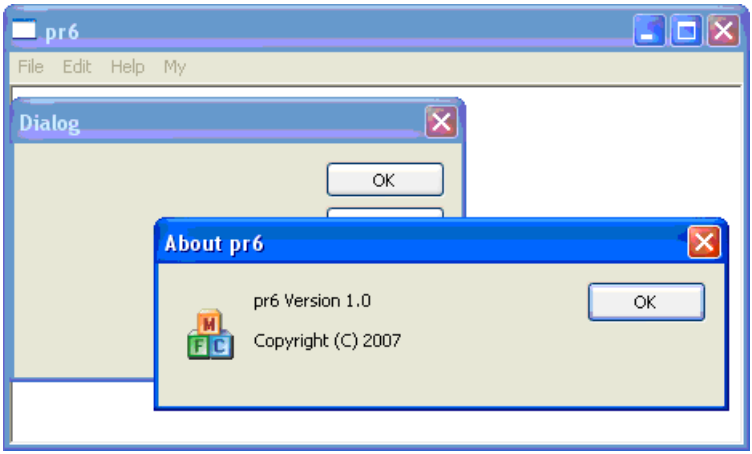


Рис. 6.6. Создание немодального диалога

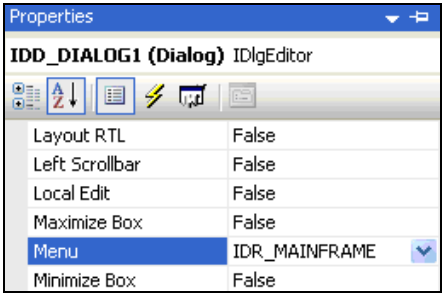


Рис. 6.7. Подключение меню к окну диалога

По умолчанию размер всех диалоговых окон в процессе работы программы изменять нельзя.

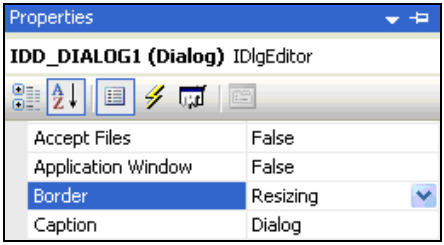


Рис. 6.8. Изменение рамки окна диалога

Чтобы можно было менять размер диалогового окна, надо изменить стиль рамки. В окне свойств диалога в поле **Border** (Рамка) значение **Dialog Frame** (Диалогового окна) нужно заменить на **Resizing** (Изменяющая размеры) (рис. 6.8). Результат работы с меню и меняющимися размерами показан на рис. 6.9.

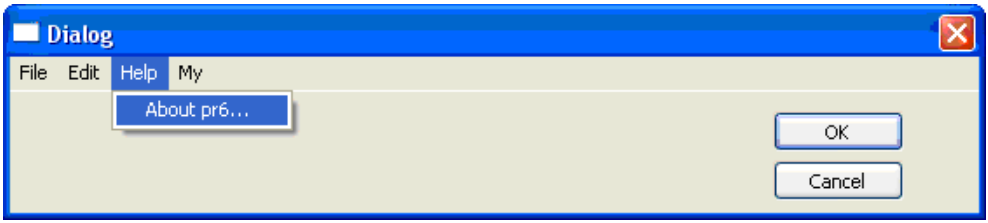


Рис. 6.9. Диалоговое окно с меню и с измененными размерами

6.1.2. Кнопка (*Button*)

Добавить любой элемент управления можно, "захватив" его в окне **Toolbox** и перетащив в окно диалога. Добавим кнопку **Button** (рис. 6.10) и обработку сообщения о ее нажатии. Добавить обработку сообщения о нажатия кнопки можно двумя способами:

- ❒ дважды щелкнув левой кнопкой мыши на кнопке в окне ресурса диалогового окна;

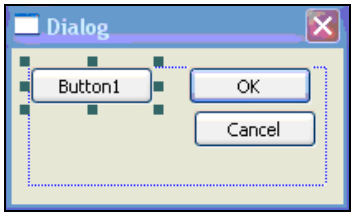
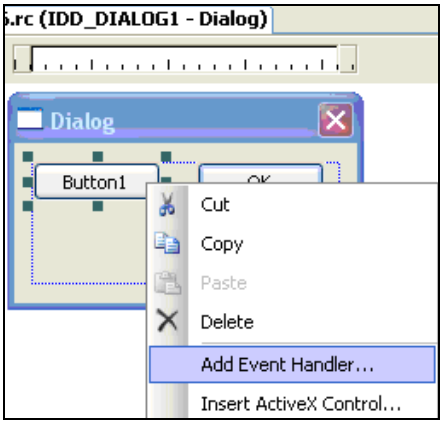


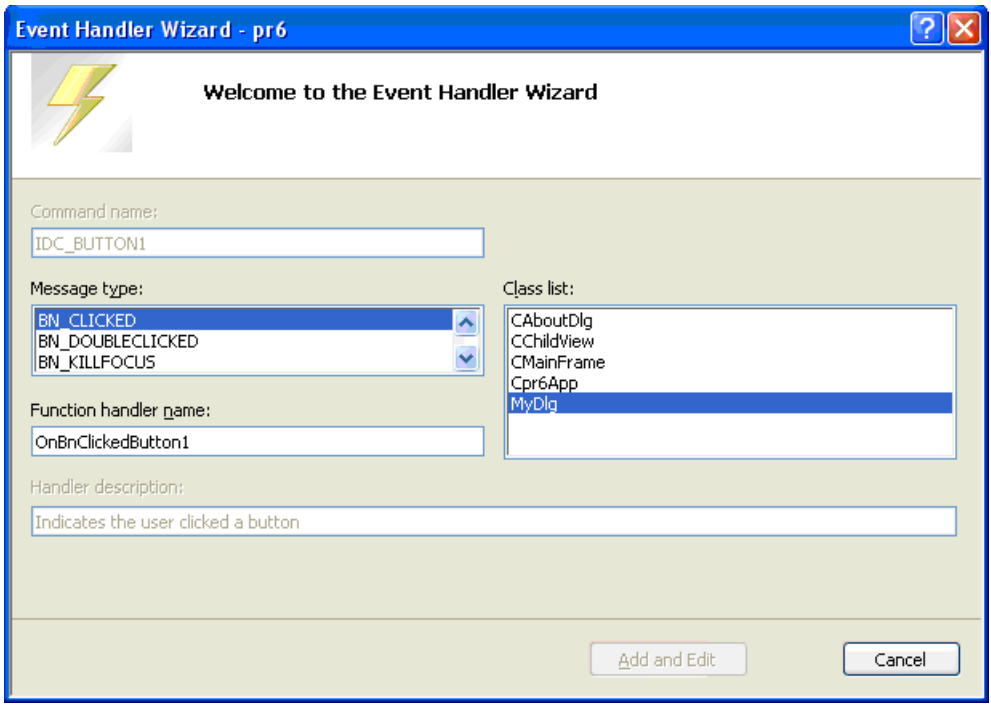
Рис. 6.10. Добавление кнопки в окно диалога

- ❒ выполнить действия, аналогичные добавлению обработки пункта меню:
  - для кнопки вызвать контекстное меню и выполнить команду **Add Event Handler** (рис. 6.11, *а*);

- в окне **Event Handler Wizard** выбрать в **Message type** значение **BN_CLICKED** и в списке **Class list** класс **MyDlg** (рис. 6.11, *б*);



а



б

Рис. 6.11. Добавление обработки нажатия кнопки с помощью контекстного меню (*а*) и выбор типа сообщения от кнопки (*б*)

Изменения в файлах показаны в листингах 6.7—6.9.

Листинг 6.7. Изменения в файле Resource.h для работы с кнопкой

```
// ...
#define IDC_BUTTON1                     1001
// ...
```

Листинг 6.8. Изменения в файле MyDlg.h для обработки нажатия кнопки

```
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnBnClickedButton1();
};
```

Листинг 6.9. Изменения в файле MyDlg.cpp для обработки нажатия кнопки

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
END_MESSAGE_MAP()

// MyDlg message handlers
void MyDlg::OnBnClickedButton1()
{
   // TODO: Add your control notification handler code here
   AfxMessageBox(L"Button1");
}
```

Результат работы программы показан на рис. 6.12.

Для каждого элемента управления в окне диалога можно выбрать точное расположение и размеры с помощью пункта главного меню **Format** (Форматирование) (рис. 6.13). С помощью него можно также выравнивать выделен-

ные элементы управления (элементы управления выделяются прямоугольной областью, аналогично графическому редактору).

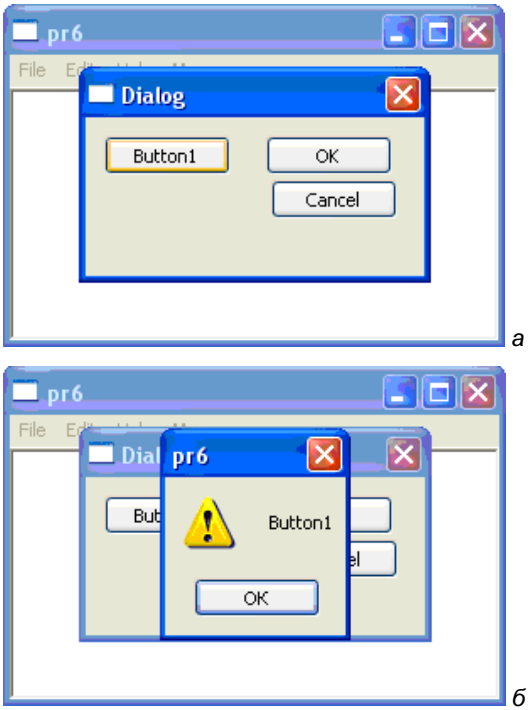


Рис. 6.12. Вид окна диалога с элементом управления **Button** (*а*) и обработка нажатия кнопки (*б*)

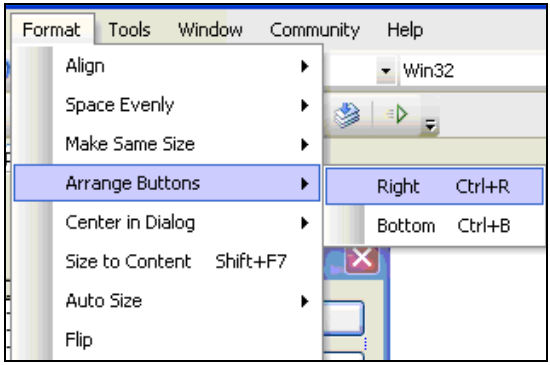


Рис. 6.13. Форматирование расположения элементов в окне диалога

В процессе работы окна диалога при нажатии клавиши <Tab> управление передается поочередно всем элементам окна диалога. Посмотреть или изменить эту очередность можно с помощью команды меню **Format | Tab Order** (Форматирование | Порядок табуляторов) (рис. 6.14). Для изменения номера очередности надо по нему щелкнуть левой кнопкой мыши.

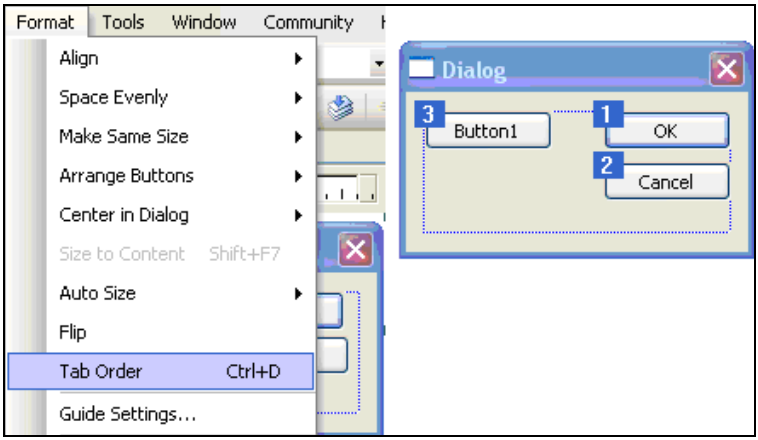


Рис. 6.14. Изменение порядка табуляторов в окне диалога

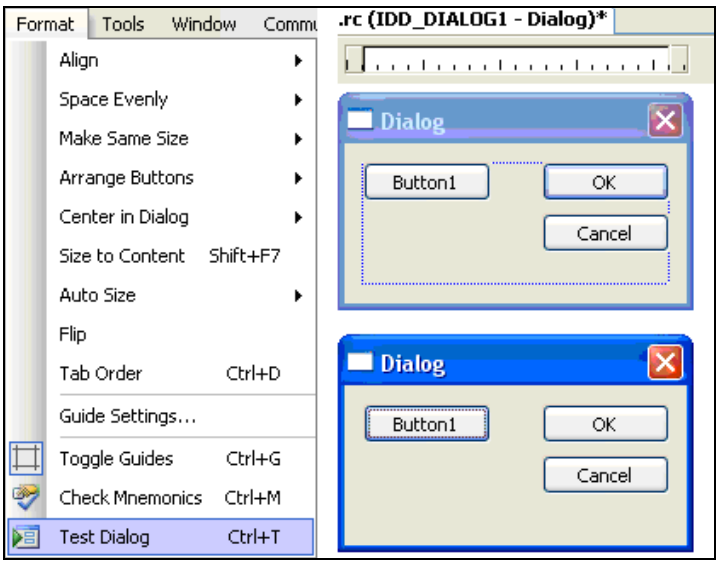


Рис. 6.15. Тестирование работы окна диалога

Для того чтобы проверить работу окна диалога, необязательно каждый раз строить и запускать программу. Это можно сделать, выполнив команду меню **Format | Test Dialog** (Форматирование | Тестирование диалога) (рис. 6.15).

6.1.3. Флажок (*Check Box*)

Поместим в окно диалога элемент управления **Check Box** (рис. 6.16).

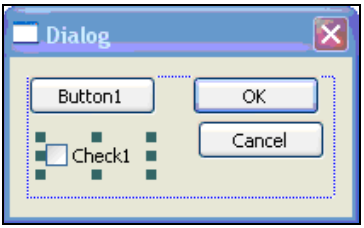
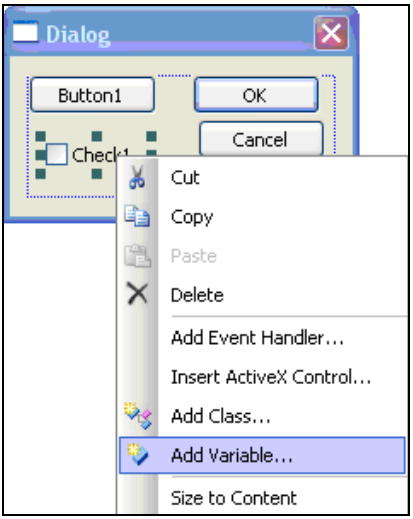


Рис. 6.16. Добавление флажка в окно диалога

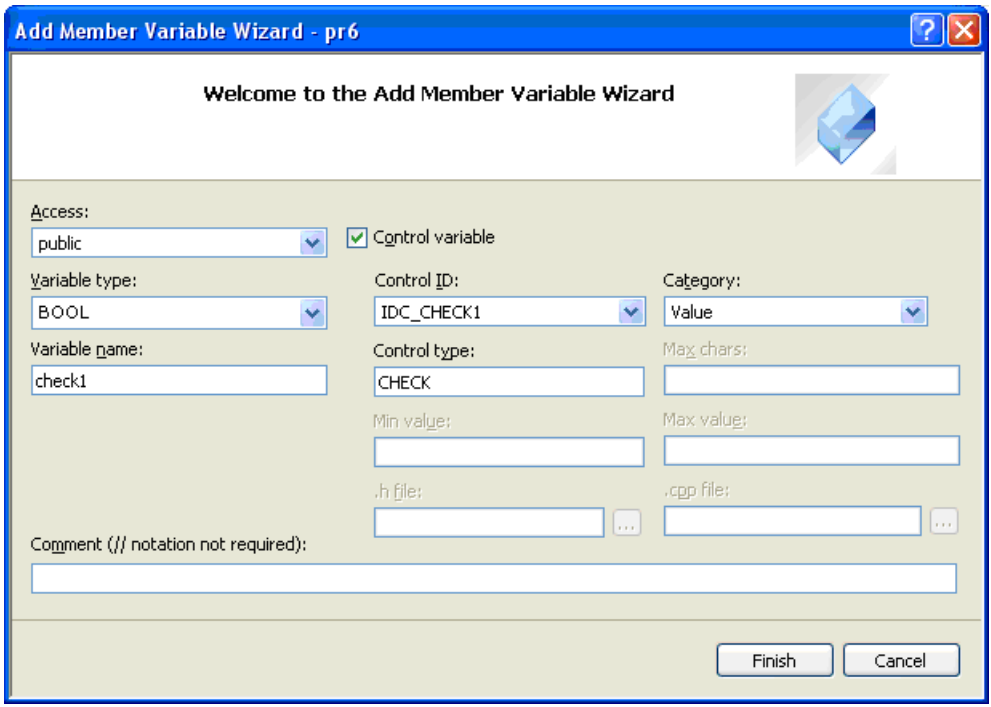
Добавим в класс диалога переменную `check1`, которая будет связана с этим элементом управления. Для этого нужно:

- вызвать для элемента **Check Box** контекстное меню и выполнить команду **Add Variable** (Добавить переменную) (рис. 6.17, *а*);
- в окне **Add Member Variable Wizard** (Мастер добавления переменных) надо:
  - в поле **Category** (Категория) изменить значение с **Control** (Управление) на **Value** (Значение) (рис. 6.17, *б*);
  - ввести в поле **Variable name** (Имя переменной): "check1" (рис. 6.17, *б*).

> ***Примечание***
>
> При выборе категории для добавляемой переменной надо руководствоваться следующим правилом: переменная **Control** используется *только при работе* диалогового окна и становится недоступной после закрытия окна диалога; переменная **Value** инициализируется и используется *только после закрытия* окна диалога по кнопке **OK** (во время работы окна диалога она не инициализирована и бесполезна).



а



б

Рис. 6.17. Добавление переменной, связанной с элементом управления **Check Box**, с помощью контекстного меню (*а*) и параметры в окне Мастера добавления переменных (*б*)

Изменения в файлах в показаны в листингах 6.10—6.12.

Листинг 6.10. Изменения в файле Resource.h для работы с флажком

```
// ...
#define IDC_CHECK1                      1003
// ...
```

Листинг 6.11. Изменения в файле MyDlg.h для работы с флажком

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   BOOL check1;
};
```

Листинг 6.12. Изменения в файле MyDlg.cpp для работы с флажком

```
//
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , check1(FALSE)              // По умолчанию флажок не отмечен
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   DDX_Check(pDX, IDC_CHECK1, check1);
}
```

Функция динамического обмена данными (Dynamic Data Exchange) автоматически синхронизирует данные самого элемента управления и члена класса, связанного с ним. Эта функция объявлена как:

```
void AFXAPI DDX_Check(
   CDataExchange* pDX,
```

```
   int nIDC,                  // Идентификатор элемента управления
   BOOL & value);             // Член класса, участвующий в обмене
                              // (обычный флажок)
```

или

```
void AFXAPI DDX_Check(
   CDataExchange* pDX,
   int nIDC,
   int& value);               // Флажок с тремя состояниями
```

Задать нужный тип параметра `value` можно в поле **Variable type** (Тип переменной) окна **Add Member Variable Wizard** (см. рис. 6.17, *б*).

Добавим код программы так, чтобы при закрытии диалога (при нажатии кнопки **OK**), если был установлен флажок `check1`, выдавалось сообщение. Изменения в файле показаны в листинге 6.13.

Листинг 6.13. Изменения в файле ChildView.cpp для обработки состояния флажка в момент закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;
   INT_PTR nRet;
   nRet = dlg.DoModal();
   switch (nRet)
   {
      case IDOK:
                     // AfxMessageBox(L"OK");
                     if(dlg.check1)      // Был выбран флажок
                        AfxMessageBox(L"check1");
                     break;
      case IDCANCEL:
                     AfxMessageBox(L"CANCEL");
                     break;
   }
}
```

Результаты работы программы показаны на рис. 6.18.

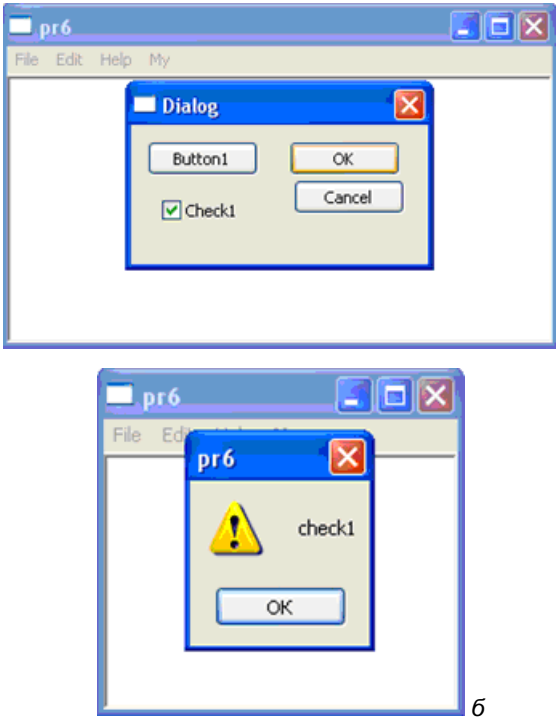


Рис. 6.18. Вид окна диалога с элементом управления **Check Box** (*а*) и обработка нажатия кнопки **OK** при установленном флажке (*б*)

Узнать или задать состояние флажка до закрытия окна диалога (предварительно создав для нее переменную категории **Control**) можно с помощью функций `GetCheck()` и `SetCheck()`.

Текущее состояние флажка (для флажков с заданными стилями `BS_AUTOCHECKBOX`, `BS_AUTORADIOBUTTON`, `BS_AUTO3STATE`, `BS_CHECKBOX`, `BS_RADIOBUTTON`, `BS_3STATE`) можно с помощью функции:

```
int CButton::GetCheck() const;
```

Функция возвращает одно из следующих значений:

- `BST_UNCHECKED` (0) — флажок не установлен;
- `BST_CHECKED` (1) — флажок установлен;

- `BST_INDETERMINATE` (2) — состояние флажка не определено (флажок не активного серого цвета). Это значение может возвращаться только для кнопок со стилем `BS_3STATE` или `BS_AUTO3STATE`.

Задать состояние кнопки можно с помощью функции:

```
void CButton::SetCheck(
   int nCheck);          // См. возвращаемые значения функции GetCheck()
```

6.1.4. Текстовое поле (*Edit Control*)

Добавим текстовое поле **Edit Control** (рис. 6.19) и переменную `edit1` (категории **Value**), которая будет связана с этим элементом. В поле **Max chars** окна **Add Member Variable Wizard** дополнительно можно задать максимальное количество символов при вводе текста (при длине больше 7 символов текст просто перестанет вводиться) (рис. 6.20).

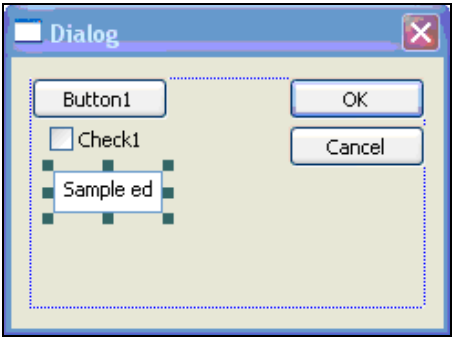


Рис. 6.19. Добавление **Edit Control** в окно диалога

Изменения в файлах приведены в листингах 6.14—6.16.

Листинг 6.14. Изменения в файле Resource.h для работы с текстовым полем

```
// ...
#define IDC_EDIT1                       1004
// ...
```

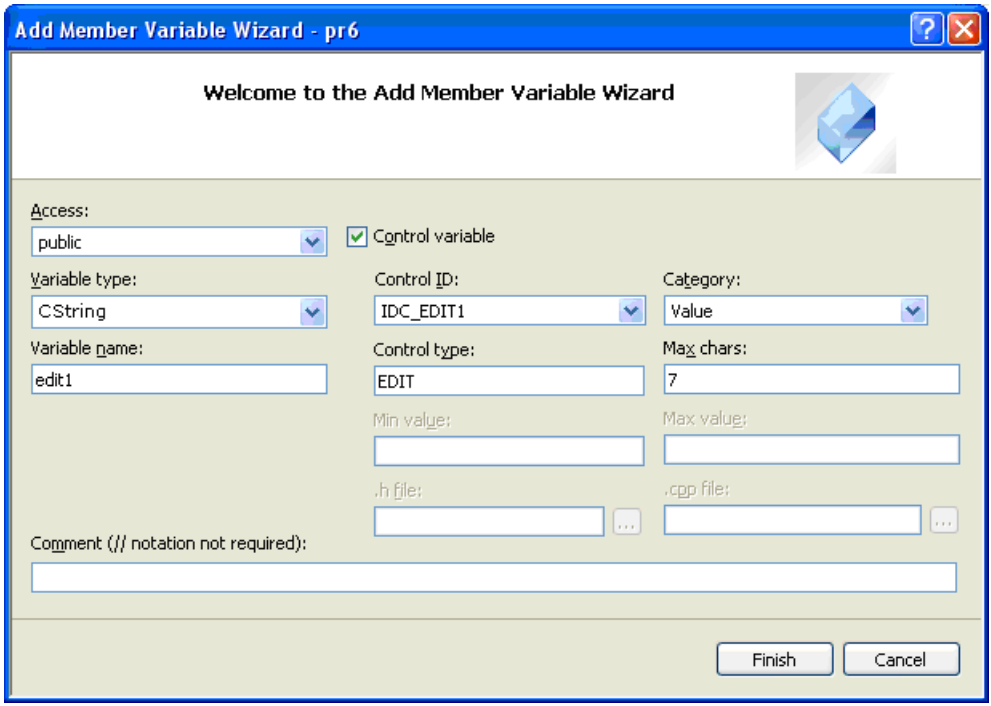


Рис. 6.20. Добавление переменной, связанной с **Edit Control**

Листинг 6.15. Изменения в файле MyDlg.h для работы с текстовым полем

```
class MyDlg : public CDialog
{
// ...
public:
   CString edit1;
};
```

Листинг 6.16. Изменения в файле MyDlg.cpp для работы с текстовым полем

```
MyDlg::MyDlg(CWnd* pParent /*=NULL*/): CDialog(MyDlg::IDD, pParent)
   , check1(FALSE)
   , edit1(_T(""))          // Изначально в поле редактирования пусто
```

```
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   DDX_Check(pDX, IDC_CHECK1, check1);
   DDX_Text(pDX, IDC_EDIT1, edit1);
   DDV_MaxChars(pDX, edit1, 7);
}
```

Для динамического обмена данным между текстовым полем и связанным с ним членом класса существуют следующие функции:

```
void AFXAPI DDX_Text(
   CDataExchange* pDX,
   int nIDC,                // Идентификатор элемента управления
   BYTE& value);            // Член класса, участвующий в обмене
```

или

```
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, short& value));
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, int& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, UINT& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, long& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, DWORD& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, CString& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, float& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, double& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, COleCurrency& value);
void AFXAPI DDX_Text(CDataExchange* pDX, int nIDC, COleDateTime& value);
```

Функция проверки корректности данных (Dynamic Data Validation) *не может быть* использована без вызова соответствующей функции `DDX`.

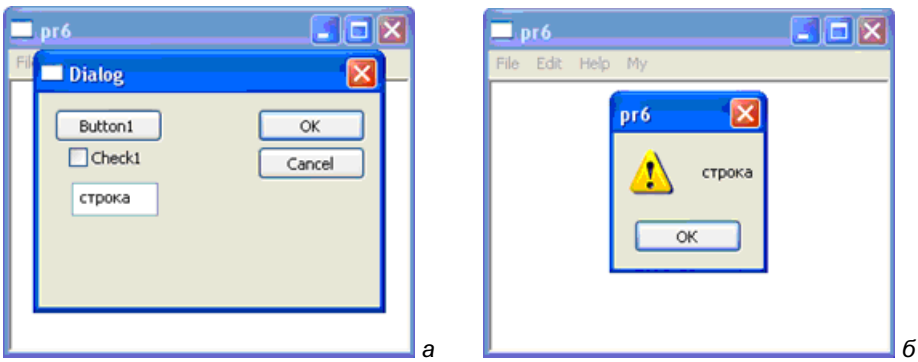
Контроль количества символов в текстовом поле осуществляется с помощью функции:

```
void AFXAPI DDV_MaxChars(
   CDataExchange* pDX,
   CString const& value,          // Член класса, участвующий в обмене
   int nChars);                   // Контрольное значение
```

Добавим код программы так, чтобы при закрытии диалога (нажатием кнопки **OK**), если был введен текст, выдавалось сообщение. Изменения в файле приведены в листинге 6.17.

Листинг 6.17. Изменение в файле ChildView.cpp для обработки содержимого текстового поля после закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
// ...
   switch (nRet)
   {
      case IDOK:
                    // ...
                    if(dlg.edit1.GetLength())
                       AfxMessageBox(dlg.edit1);
// ...
   }
}
```



а *б*

Рис. 6.21. Вид окна диалога с элементом управления **Edit Control** (*а*) и обработка нажатия кнопки **OK** (*б*)

Получение длины строки (без завершающего символа `'\0'`) выполняется с помощью функции:

```
int CSimpleStringT::GetLength() const throw();
```

Для работы со строками существуют следующие классы:

```
typedef CStringT CString
class CStringT : public CSimpleStringT
```

Результат работы программы показан на рис. 6.21.

Сделаем так, чтобы пользователь обязан был ввести текст в поле редактирования (диалоговое окно не закроется по кнопке **OK**, если поле пустое). Для этого надо добавить переменную `cedit1` (категории **Control**) (рис. 6.22) и обработку сообщения нажатия кнопки **OK** (дважды щелкнув по кнопке **OK** в ресурсе окна диалога).

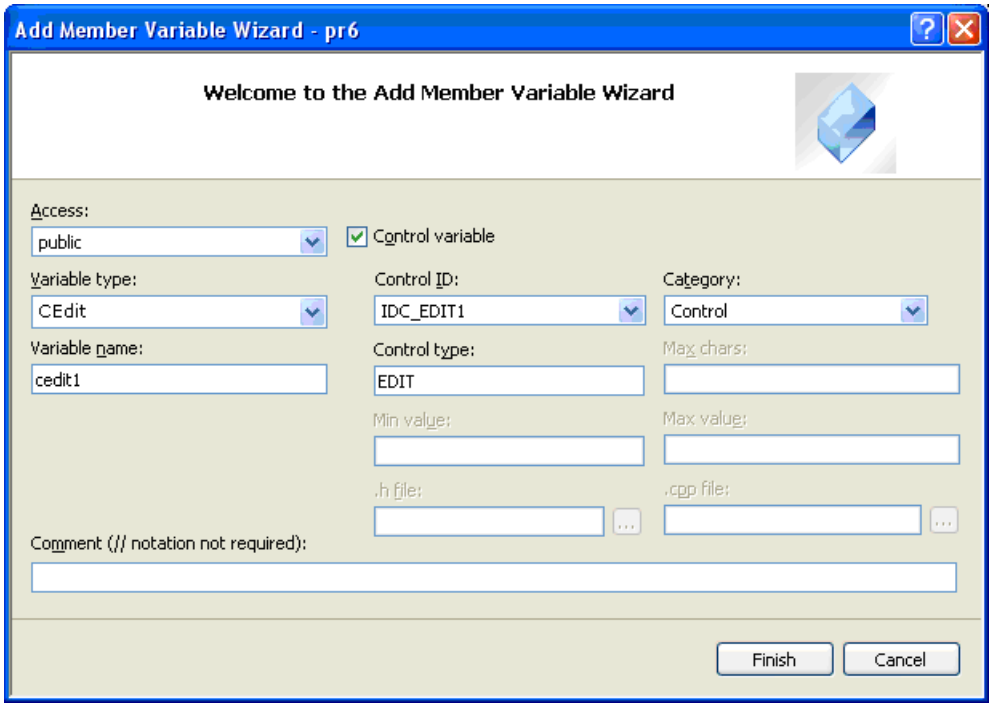


Рис. 6.22. Добавление управляющей переменной, связанной с **Edit Control**

Изменения в файлах приведены в листингах 6.18 и 6.19.

Листинг 6.18. Изменение в файле MyDlg.h для контроля содержимого текстового поля до закрытия окна диалога

```
class MyDlg : public CDialog
{
// ...
public:
   CEdit cedit1;
public:
   afx_msg void OnBnClickedOk();
};
```

Листинг 6.19. Изменение в файле MyDlg.cpp для контроля содержимого текстового поля до закрытия окна диалога

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_EDIT1, cedit1);
}
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
   ON_BN_CLICKED(IDOK, &MyDlg::OnBnClickedOk)
END_MESSAGE_MAP()
// ...
void MyDlg::OnBnClickedOk()
{
   // TODO: Add your control notification handler code here
   if(!cedit1.GetWindowTextLengthW())
   {
      AfxMessageBox(L"Заполните edit1");
      // Передать управление edit1
      GotoDlgCtrl(&cedit1);
```

```
        return;         // Не закрывать окно диалога
    }
    OnOK();             // Произвести обмен данными и закрыть окно диалога
}
```

Получить длину текста в окне редактирования (без завершающего символа `'\0'`) можно с помощью функции:

```
int CWnd::GetWindowTextLength() const;
```

> ***ПРИМЕЧАНИЕ***
>
> В данном случае нельзя использовать функцию `edit1.GetLength()`, т. к. диалоговое окно еще не закрыто и значение `edit1` не инициализировано.

Передать фокус элементу управления в диалоговом окне можно функцией:

```
void CDialog::GotoDlgCtrl(
   CWnd* pWndCtrl);              // Указатель на элемент управления
```

Перемещение фокуса на предыдущий элемент управления:

```
void CDialog::PrevDlgCtrl() const;
```

Перемещение фокуса на следующий элемент управления:

```
void CDialog::NextDlgCtrl() const;
```

Результат работы программы показан на рис. 6.23.

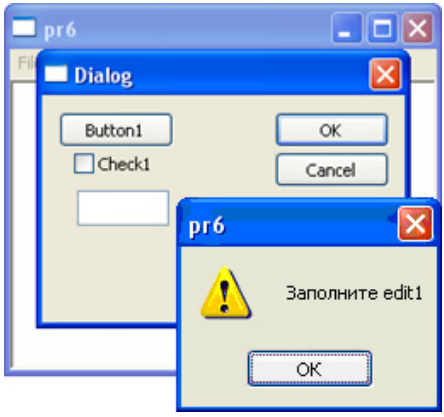
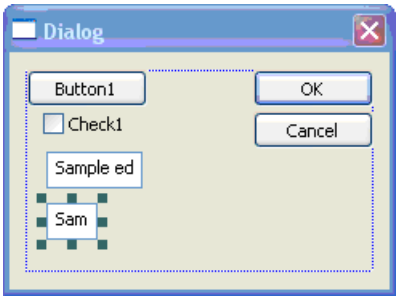


Рис. 6.23. Контроль содержимого **Edit Control** до закрытия окна диалога

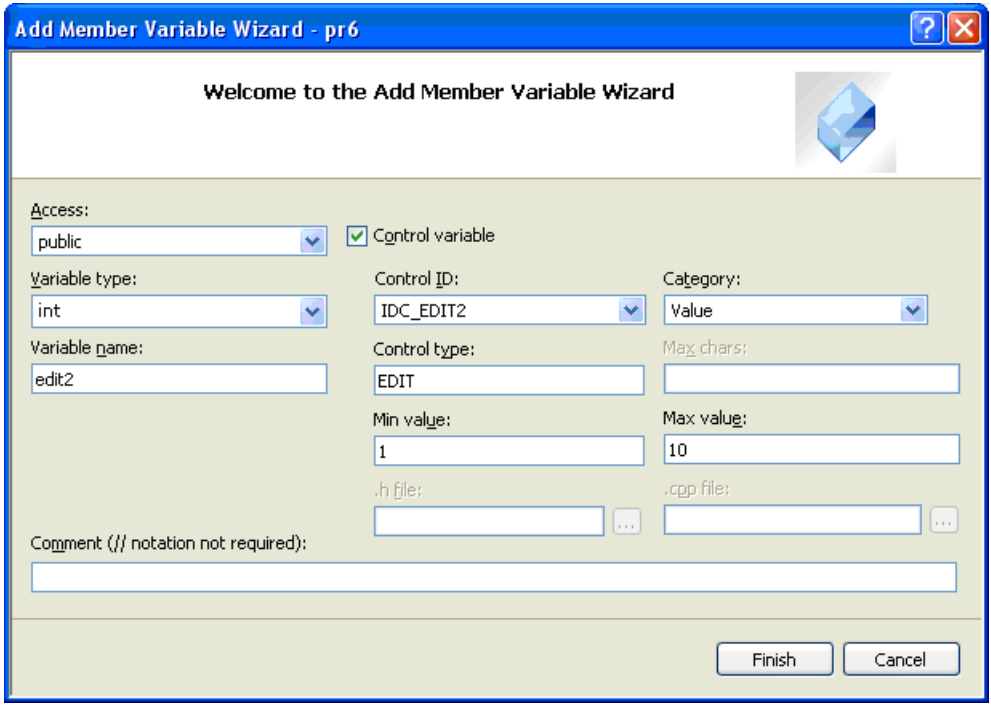
Примечание

Можно было и не заводить переменную `cedit1`. Тогда работа с элементом управления выглядела бы так:

```
CEdit *pedit1 = (CEdit *)(GetDlgItem(IDC_EDIT1));
pedit1->GetWindowTextLengthW();
```



а



б

Рис. 6.24. Добавление еще одного текстового поля (*а*) и переменной для ввода целых чисел (*б*)

Функция получения указателя на элемент управления в окне диалога выглядит следующим образом:

```
CWnd* CWnd::GetDlgItem(   // Указатель на элемент управления или
                          // 0 - при ошибке
   int nID) const;        // Идентификатор элемента управления
```

или

```
void CWnd::GetDlgItem(
   int nID,               // Идентификатор объекта
   HWND* phWnd) const     // Дескриптор окна объекта (возвращается)
```

Добавим еще одно текстовое поле (рис. 6.24, *а*) и переменную `edit2` для работы с целыми числами в диапазоне от 1 до 10. В окне **Add Member Variable Wizard** (рис. 6.24, *б*) надо задать следующие параметры:

- в списке **Variable type** выбрать **int**;
- в поле **Variable name** ввести "edit2";
- в списке **Category** выбрать **Value**;
- в поле **Min value** ввести "1";
- в поле **Max value** — "10".

Изменения в файлах приведены в листингах 6.20—6.22.

Листинг 6.20. Изменения в файле Resource.h для работы с целыми числами в текстовом поле

```
// ...
#define IDC_EDIT2                       1005
// ...
```

Листинг 6.21. Изменения в файле MyDlg.h для работы с целыми числами в текстовом поле

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   int edit2;
};
```

Листинг 6.22. Изменения в файле MyDlg.cpp для работы с целыми числами в текстовом поле

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/): CDialog(MyDlg::IDD, pParent)
   , check1(FALSE), edit1(_T(""))
   , edit2(0)                    // 0 - начальное значение по умолчанию
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Text(pDX, IDC_EDIT2, edit2);
   DDV_MinMaxInt(pDX, edit2, 1, 10);
}
```

Контроль значения целых чисел выполняется с помощью функции:

```
void AFXAPI DDV_MinMaxInt(
   CDataExchange* pDX,
   int value,                // Член класса, участвующий в обмене
   int minVal,               // Минимальное значение
   int maxVal);              // Максимальное значение
```

Для нашего случая изменим начальное значение в `edit2` с 0 на 1. Изменения в файле приведены в листинге 6.23.

Листинг 6.23. Изменения в файле MyDlg.cpp для задания начального значения в текстовом поле

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/): CDialog(MyDlg::IDD, pParent)
   , check1(FALSE), edit1(_T(""))
   , edit2(1)
{
}
```

Результат работы программы показан на рис. 6.25.

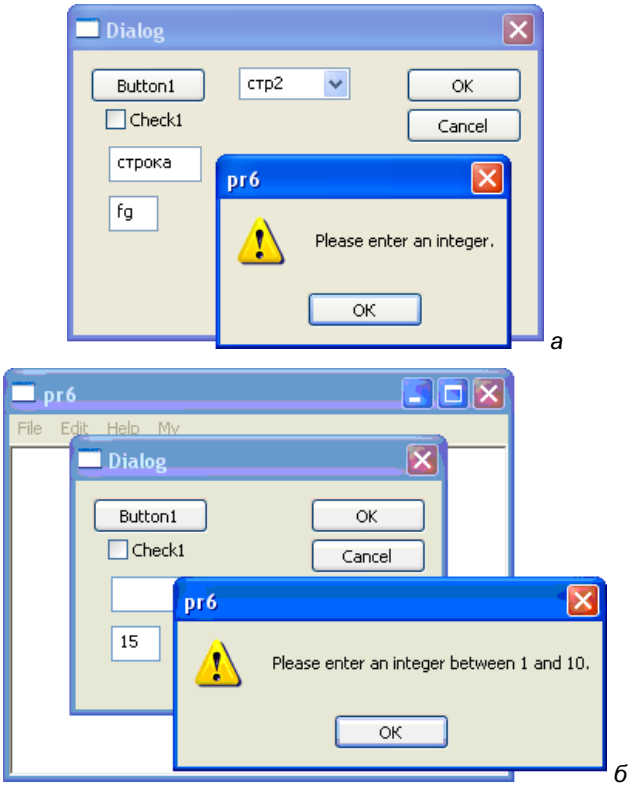


Рис. 6.25. Контроль числа в текстовом поле (*а*) и контроль диапазона числа в текстовом поле (*б*)

Контроль того, что в текстовом поле находится число, можно выполнять не только при нажатии кнопки **OK**, но и в момент ввода в текстовое поле. Для этого надо изменить в свойствах текстового поля значение **Number** (Цифры) с `False` на `True` (рис. 6.26). Теперь можно будет вводить только цифры. Все остальные символы (включая знак минуса) будут игнорироваться, поэтому это годится только для ввода положительных чисел.

Еще одно важное свойство — **Read Only** (Только для чтения) (рис. 6.26). Если задать его `True`, то в текстовом поле нельзя будет редактировать текст. Занести текст туда можно будет только из программы, используя переменную категории **Control** (`CEdit cedit2`) и функцию `cedit2.SetWindowTextW()`.

Задать текст в окне можно с помощью функции:

```
void CWnd::SetWindowText(
   LPCTSTR lpszString);          // Строка, выводимая в окно
```

А получить текст из окна:

```
void CWnd::GetWindowText(
   CString& rString) const;      // Строка, куда заносится текст
                                 // из окна
```

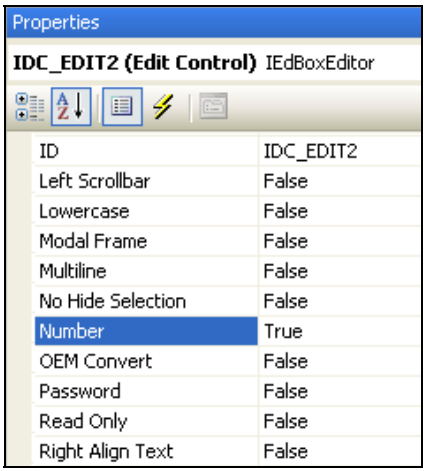


Рис. 6.26. Изменение свойств текстового поля

6.1.5. Поле со списком (*Combo Box*)

Добавим в окно диалога элемент управления **Combo Box**.

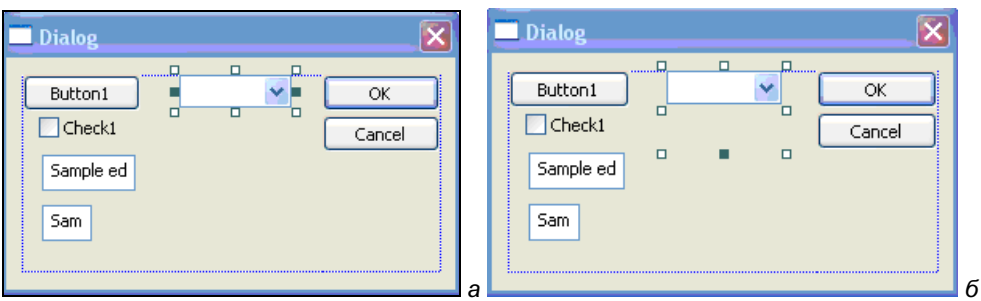


Рис. 6.27. Добавление поля с выпадающим списком (*а*) и изменение размеров выпадающего списка (*б*)

При нажатии кнопки со стрелкой (справа у элемента), можно посмотреть и отредактировать размеры открывающегося списка (рис. 6.27).

Добавим переменную (категории **Value**) `combo1`, которая будет связана с этим элементом управления (рис. 6.28).

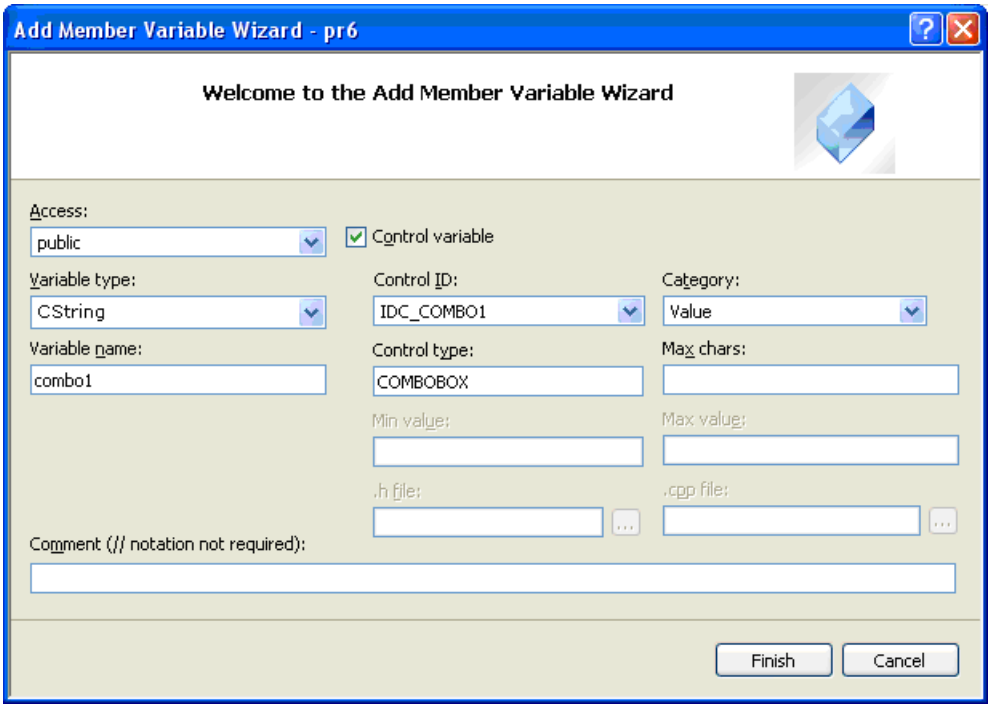


Рис. 6.28. Добавление переменной, связанной со списком

Изменения в файлах приведены в листингах 6.24—6.26.

Листинг 6.24. Изменения в файле Resource.h для работы с выпадающим списком

```
// ...
#define IDC_COMBO1                      1006
// ...
```

Листинг 6.25. Изменения в файле MyDlg.h для работы с выпадающим списком

```
class MyDlg : public CDialog
{
// ...
public:
   CString combo1;
};
```

Листинг 6.26. Изменения в файле MyDlg.cpp для работы с выпадающим списком

```
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , check1(FALSE), edit1(_T("")), edit2(0)
   , combo1(_T(""))
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_CBString(pDX, IDC_COMBO1, combo1);
}
```

Используется следующая функция обмена данными:

```
void AFXAPI DDX_CBString(
   CDataExchange* pDX,
   int nIDC,                // Идентификатор элемента управления
   CString& value);         // Член класса, участвующий в обмене
```

Теперь надо заполнить список. Для этого необходимо переопределить функцию инициализации диалога `OnInitDialog()`, которая вызывается автоматически перед созданием окна диалога:

- в окне **Class View** вызвать контекстное меню для класса `MyDlg` и выполнить команду **Properties** (рис. 6.29, *а*);
- в окне **Properties** выбрать вкладку **Overrides**;

- найти в списке функцию **OnInitDialog** и выбрать **<Add> OnInitDialog** (рис. 6.29, *б*).

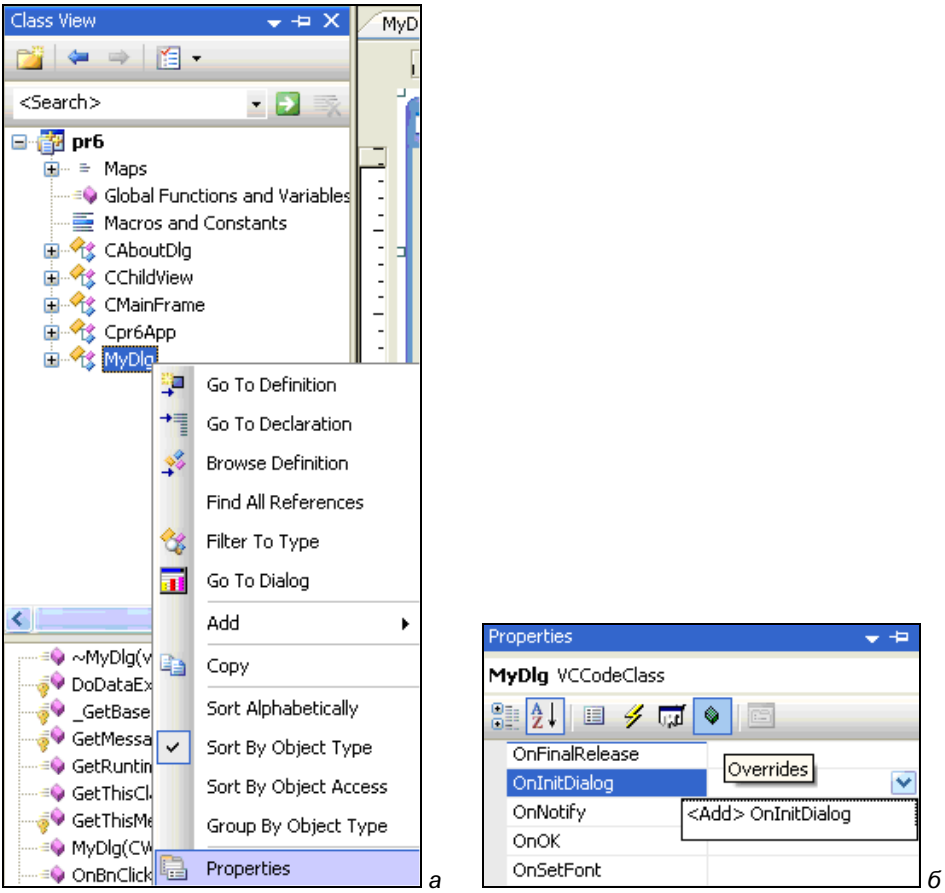


Рис. 6.29. Добавление функции инициализации диалога с помощью контекстного меню (*а*) и окна свойств (*б*)

Изменения в файлах приведены в листингах 6.27 и 6.28.

Листинг 6.27. Изменения в файле MyDlg.h для начальной инициализации окна диалога

```
class MyDlg : public CDialog
{
```

```
// ...
public:
   virtual BOOL OnInitDialog();
};
```

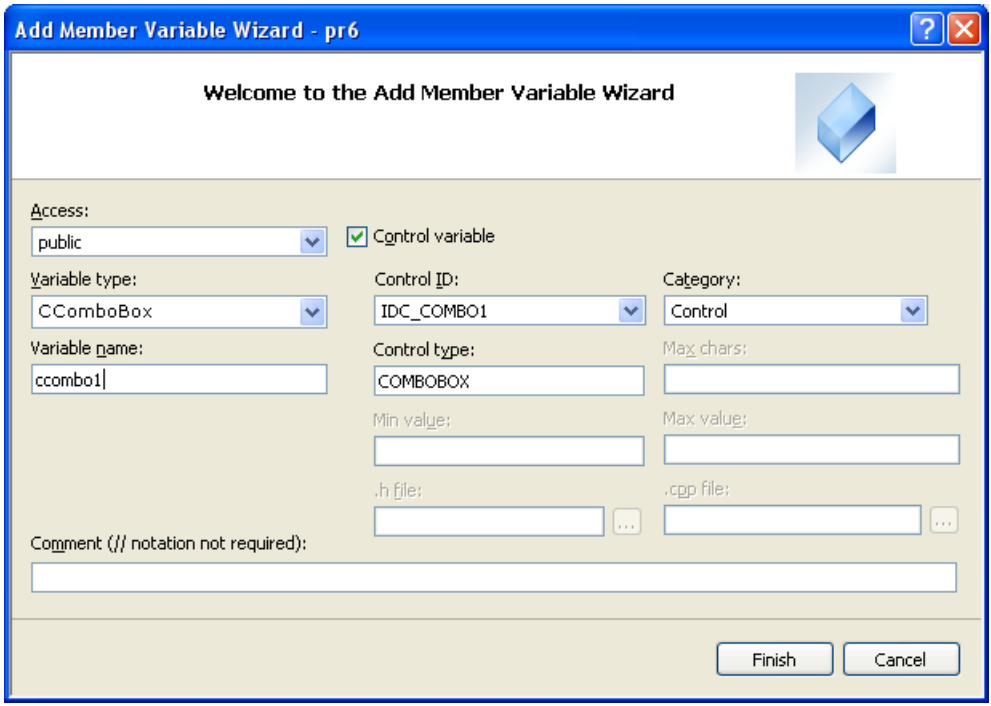


Рис. 6.30. Добавление переменной для заполнения списка

Листинг 6.28. Изменения в файле MyDlg.cpp для начальной инициализации окна диалога

```
// ...
BOOL MyDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // TODO:  Add extra initialization here
```

```
    return TRUE;  // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}
```

Теперь надо добавить переменную `ccombo1` (категории **Control**), связанную со списком (рис. 6.30), и заполнить список.

Изменения в файлах приведены в листингах 6.29 и 6.30.

Листинг 6.29. Изменения в файле MyDlg.h для заполнения выпадающего списка

```
class MyDlg : public CDialog
{
// ...
public:
    CComboBox ccombo1;
};
```

Листинг 6.30. Изменения в файле MyDlg.cpp для заполнения выпадающего списка

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
    // ...
    DDX_CBString(pDX, IDC_COMBO1, combo1);
    DDX_Control(pDX, IDC_COMBO1, ccombo1);
    // ...
}
// ...
BOOL MyDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO:  Add extra initialization here
    // Заполнить список
```

```
    ccombo1.AddString(L"стр1");
    ccombo1.AddString(L"стр2");
    ccombo1.AddString(L"стр3");
    // Сделать по умолчанию "стр2"
    ccombo1.SetCurSel(1);
    return TRUE;  // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}
```

Добавление строки в список (список сразу сортируется в алфавитном порядке) выполняется с помощью функции:

```
int CComboBox::AddString(           // Индекс новой строки в списке
   LPCTSTR lpszString);             // Добавляемая строка
```

Если надо создать неупорядоченный список, то используется функция:

```
int CComboBox::InsertString(        // Индекс новой строки в списке
   int nIndex,                      // Индекс строки в списке
   LPCTSTR lpszString);             // Добавляемая строка
```

Если `nIndex = -1`, то строка добавляется в конец списка.

> ***Примечание***
>
> Индексация строк начинается с 0 и по порядку. Если задать неправильный индекс, например:
>
> ```
> ccombo1.InsertString(0, L"стр1");
> ccombo1.InsertString(1, L"стр2");
> ccombo1.InsertString(3, L"стр3");
> ```
>
> то строка `"стр3"` в списке не появится.

Получить количество элементов в списке можно с помощью функции:

```
int CComboBox::GetCount() const;
```

Изначально в списке не выбрана ни одна строка. Если надо установить начальное значение, то используется функция:

```
int CComboBox::SetCurSel(           // Индекс строки в списке или
                                    // -1 при ошибке
   int nSelect);                    // Индекс строки в списке
```

Удалить строку из списка можно функцией:

```
int CComboBox::DeleteString(     // Количество оставшихся элементов
   UINT nIndex);                 // Индекс удаляемого элемента
```

Удалить все элементы из списка:

```
void CComboBox::ResetContent();
```

Найти строку в списке можно с помощью функции:

```
int CComboBox::FindString(       // Индекс найденной строки или -1
   int nStartAfter,              // Индекс начала поиска
   LPCTSTR lpszString) const;    // Строка для поиска
```

Параметр `nStartAfter` должен задавать индекс строки, стоящей перед той, с которой будет начат поиск. При достижении конца списка поиск будет продолжаться от начала до строки с индексом `nStartAfter` включительно. Если задать `nStartAfter = -1`, то поиск будет проходить от начала по всему списку.

Поиск и выделение строки списка выполняется функцией:

```
int CComboBox::SelectString(      // Индекс найденной строки или -1
   int nStartAfter,               // Индекс начала поиска
   LPCTSTR lpszString);           // Строка для поиска
```

Сделаем так, чтобы при закрытии диалога (при нажатии кнопки **OK**) выдавалось сообщение с текстом выбранной строки. Изменения в файле приведены в листинге 6.31.

Листинг 6.31. Изменения в файле ChildView.cpp для обработки выбранной строки выпадающего списка после закрытия окна диалога

```
void CChildView::OnMyDlg()
{
   // ...
   switch (nRet)
   {
      case IDOK:
                     // ...
                     if(dlg.combo1.GetLength())
                        AfxMessageBox(dlg.combo1);
                     // ...
   }
}
```

Результат работы программы показан на рис. 6.31.

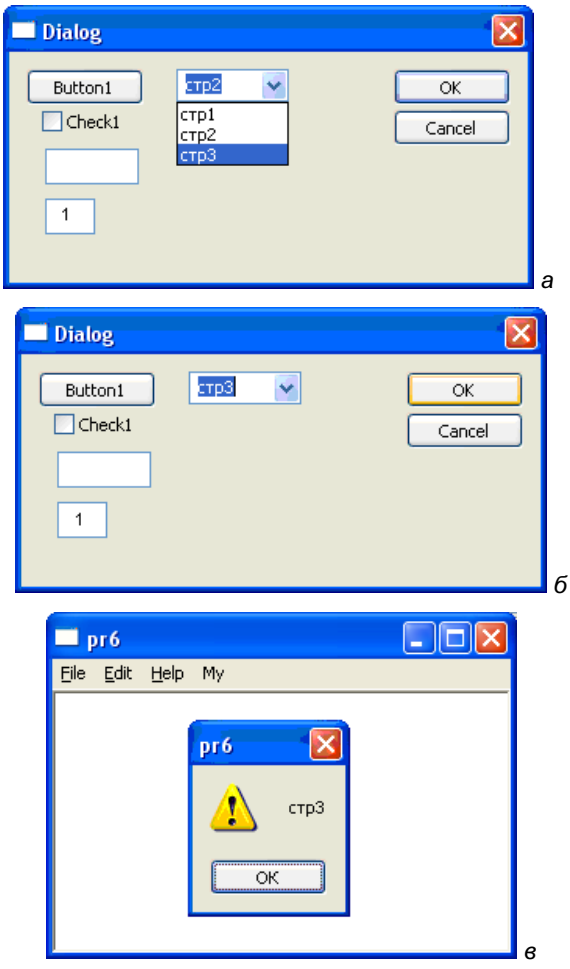


Рис. 6.31. Выбор элемента в выпадающем списке (*а*, *б*) и обработка выбранного элемента после закрытия окна диалога (*в*)

6.1.6. Список (*List Box*)

Добавим в окно диалога элемент управления **List Box** (рис. 6.32) и переменную (категории **Value**) `list1`, которая связана с этим элементом управления (рис. 6.33).

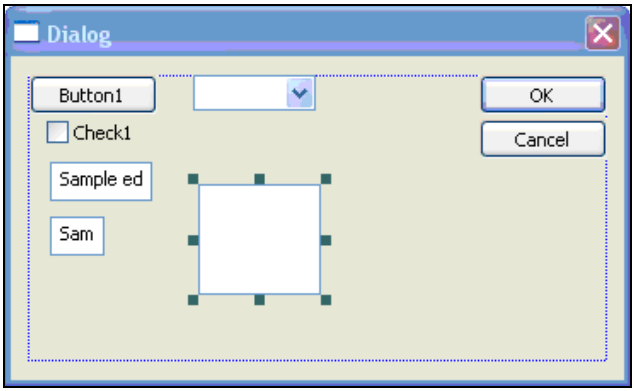


Рис. 6.32. Добавление списка в окно диалога

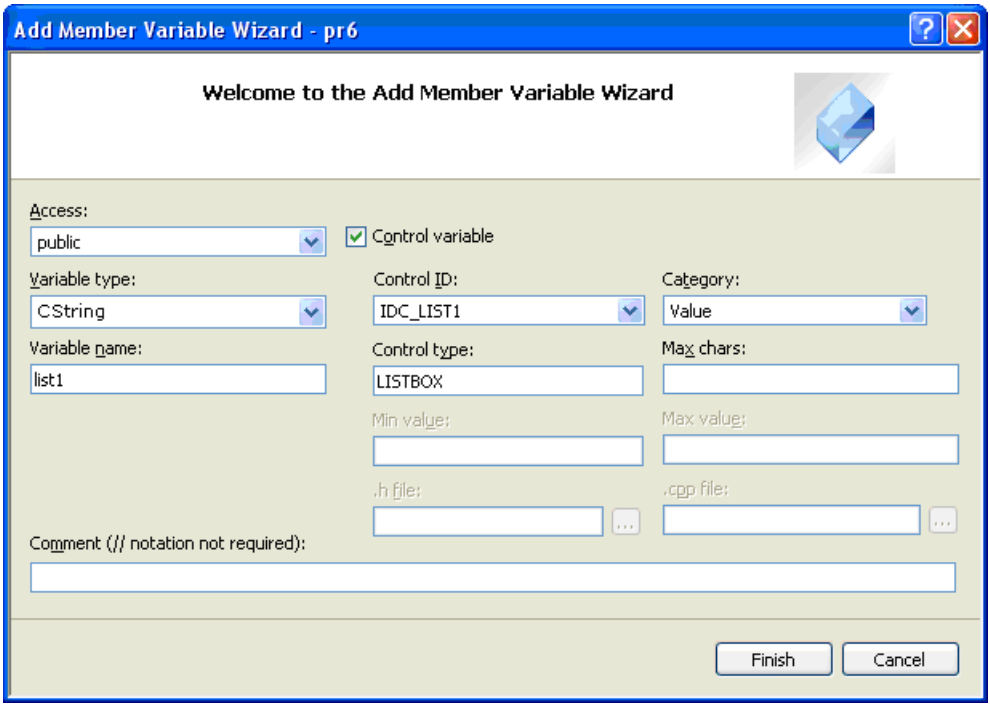


Рис. 6.33. Добавление переменной, связанной со списком

Изменения в файлах приведены в листингах 6.32—6.34.

Листинг 6.32. Изменения в файле Resource.h для работы со списком

```
// ...
#define IDC_LIST1                       1007
// ...
```

Листинг 6.33. Изменения в файле MyDlg.h для работы со списком

```
class MyDlg : public CDialog
{
// ...
public:
   CString list1;
};
```

Листинг 6.34. Изменения в файле MyDlg.cpp для работы со списком

```
MyDlg::MyDlg(CWnd* pParent /*=NULL*/): CDialog(MyDlg::IDD, pParent)
   , check1(FALSE), edit1(_T("")), edit2(0), combo1(_T(""))
   , list1(_T(""))
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_LBString(pDX, IDC_LIST1, list1);
}
```

Функция обмена данными следующая:

```
void AFXAPI DDX_LBString(
   CDataExchange* pDX,
   int nIDC,                // Идентификатор элемента управления
   CString& value);         // Член класса, участвующий в обмене
```

Работа со списком **List Box** аналогична работе с выпадающим списком **Combo Box** *(см. разд. 6.1.5)*. Для заполнения списка надо добавить переменную (категории **Control**) `clist1` (рис. 6.34).

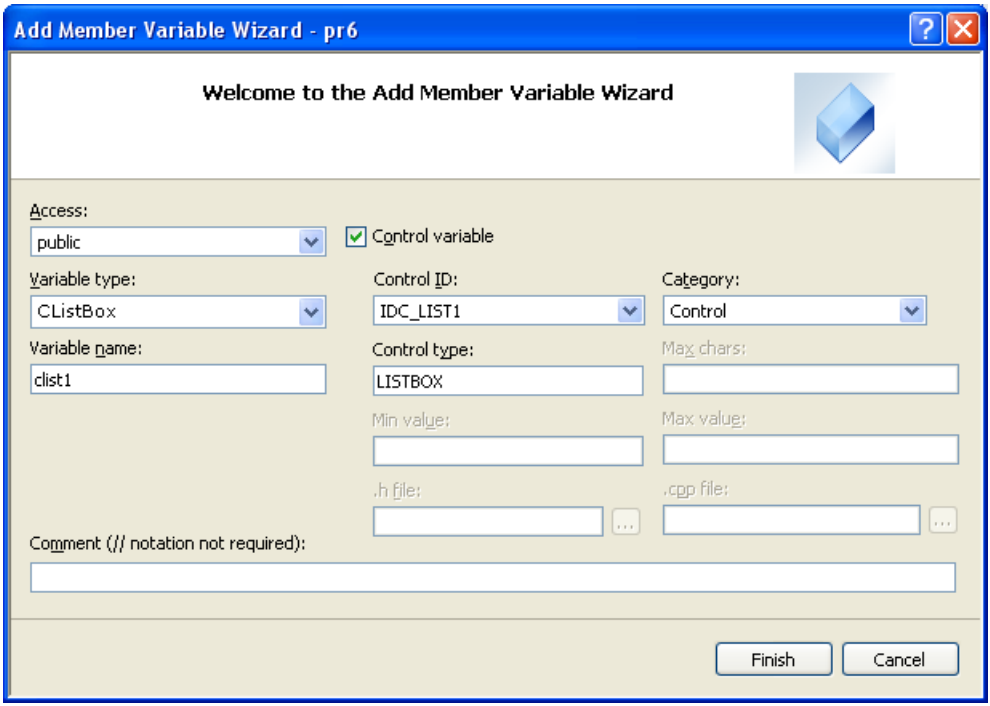


Рис. 6.34. Создание переменной для заполнения списка

Изменения в файлах приведены в листингах 6.35—6.37.

Листинг 6.35. Изменения в файле MyDlg.h для начальной инициализации списка

```
class MyDlg : public CDialog
{
// ...
public:
   CListBox clist1;
};
```

Листинг 6.36. Изменения в файле MyDlg.cpp для начальной инициализации списка

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_LBString(pDX, IDC_LIST1, list1);
   DDX_Control(pDX, IDC_LIST1, clist1);
}
// ...
BOOL MyDlg::OnInitDialog()
{
   CDialog::OnInitDialog();
   // ...
   // Заполнить список
   clist1.AddString(L"один");
   clist1.AddString(L"два");
   clist1.AddString(L"три");
   // Сделать по умолчанию "два" (список автоматически отсортирован по
   // алфавиту: два [0], один [1], три [2])
   clist1.SetCurSel(0);

   return TRUE;
}
```

Листинг 6.37. Изменения в файле ChildView.cpp для контроля выбранного элемента списка после закрытия окна диалога

```
void CChildView::OnMyDlg()
{
   // ...
   switch (nRet)
   {
      case IDOK:
```

```
                    // ...
                    if(dlg.list1.GetLength())
                       AfxMessageBox(dlg.list1);
                    // ...
   }
}
```

Результат работы программы показан на рис. 6.35.

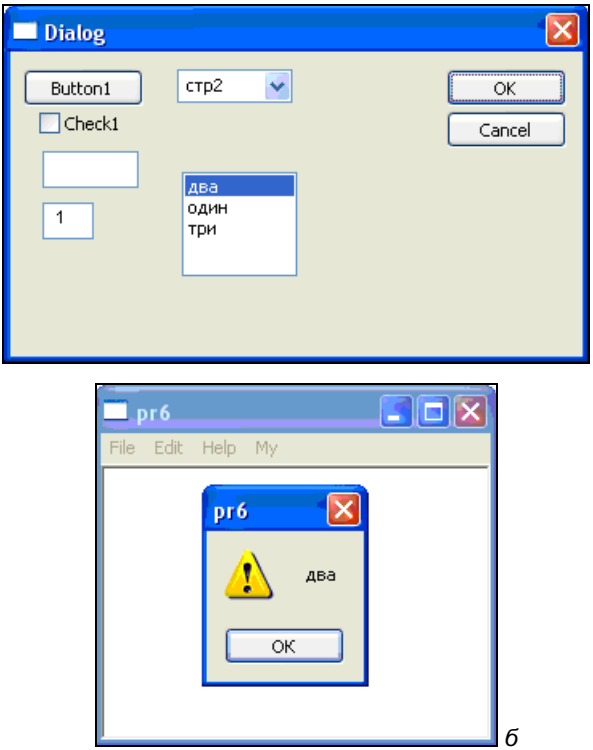


Рис. 6.35. Вид окна диалога со списком (*а*)
и обработка нажатия кнопки **OK** (*б*)

Функция для добавления элемента в список определена следующим образом:

```
int CListBox::AddString(          // Индекс новой строки в списке
   LPCTSTR lpszString);           // Добавляемая строка
```

Если надо создать неупорядоченный список, по используется функция:

```
int CListBox::InsertString(        // Индекс новой строки в списке
   int nIndex,                      // Индекс строки в списке
   LPCTSTR lpszString);             // Добавляемая строка
```

Если `nIndex = -1`, то строка добавляется в конец списка.

Получить количество элементов в списке можно с помощью функции:

```
int CListBox::GetCount() const;
```

Изначально в списке не выбрана ни одна строка. Если надо установить начальное значение, то используется функция:

```
int CListBox::SetCurSel(        // Индекс строки в списке или
                                 // -1 при ошибке
   int nSelect);                 // Индекс строки в списке
```

Удалить строку из списка можно функцией:

```
int CListBox::DeleteString(      // Количество оставшихся элементов
   UINT nIndex);                 // Индекс удаляемого элемента
```

Удалить все элементы из списка:

```
void CComboBox::ResetContent();
```

Найти строку в списке:

```
int CListBox::FindString(       // Индекс найденной строки или -1
   int nStartAfter,              // Индекс начала поиска
   LPCTSTR lpszString) const;    // Строка для поиска
```

Параметр `nStartAfter` должен задавать индекс строки, стоящей перед той, с которой будет начат поиск. При достижении конца списка поиск будет продолжаться от начала до строки с индексом `nStartAfter` включительно. Если задать `nStartAfter = -1`, то поиск будет проходить от начала по всему списку.

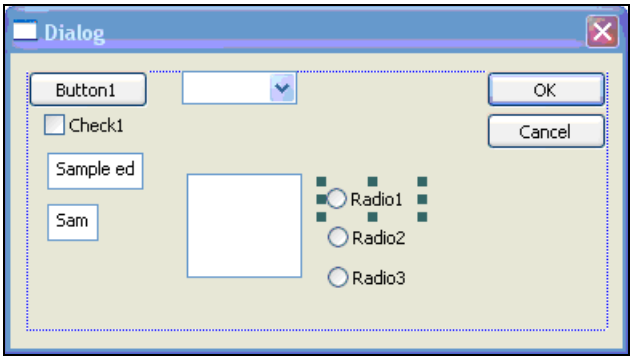
Поиск и выделение строки списка выполняется с помощью функции:

```
int CListBox::SelectString(       // Индекс найденной строки или -1
   int nStartAfter,               // Индекс начала поиска
   LPCTSTR lpszString);           // Строка для поиска
```

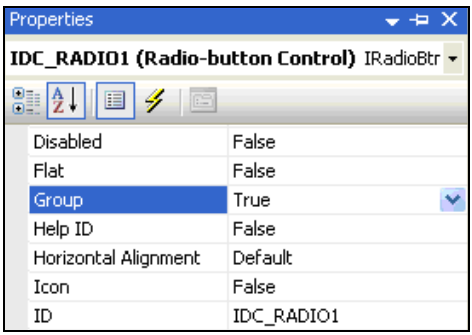
6.1.7. Переключатель (*Radio Button*)

Добавим в окно диалога три элемента **Radio Button** (рис. 6.36, *а*). Объединим эти кнопки в группу (так, чтобы можно было выбрать одну кнопку из трех).

Для этого *только у первой кнопки группы* (**Radio1**) надо в окне свойств изменить в поле **Group** значение `False` на `True` (рис. 6.36, *б*). Добавим в класс диалога переменную (категории **Value**) `radio1`, которая будет связана с группой **Radio1**, **Radio2**, **Radio3** (рис. 6.37).



а



б

Рис. 6.36. Добавление группы кнопок (*а*) и их объединение с помощью окна свойств (*б*)

Изменения в файлах приведены в листингах 6.38—6.40.

Листинг 6.38. Изменения в файле Resource.h для работы с группой переключателей

```
// ...
#define IDC_RADIO1        1008
#define IDC_RADIO2        1009
#define IDC_RADIO3        1010
// ...
```

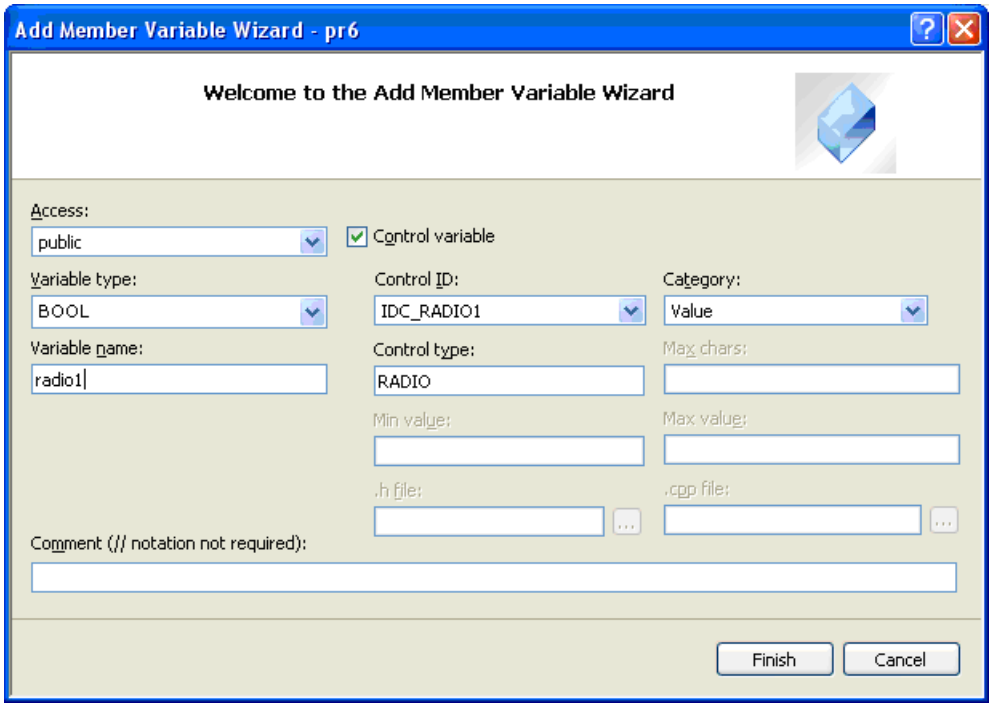


Рис. 6.37. Добавление переменной, связанной с группой кнопок

Листинг 6.39. Изменения в файле MyDlg.h для работы с группой переключателей

```
class MyDlg : public CDialog
{
// ...
public:
   BOOL radio1;
};
```

Листинг 6.40. Изменения в файле MyDlg.cpp для работы с группой переключателей

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
```

```
   , check1(FALSE), edit1(_T("")), edit2(0), combo1(_T(""))
   , list1(_T(""))
   , radio1(FALSE)
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Radio(pDX, IDC_RADIO1, radio1);
}
```

Функция обмена данными в данном случае следующая:

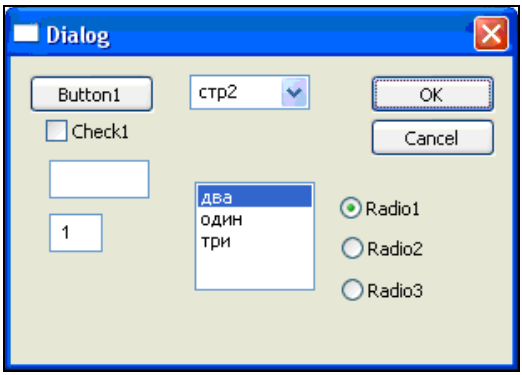
```
void AFXAPI DDX_Radio(
   CDataExchange* pDX,
   int nIDC,                  // Идентификатор элемента управления
   int& value);               // Состояние кнопки или
                              // индекс нажатой кнопки в группе
```

Добавим обработку состояния переключателей таким образом, чтобы после закрытия окна диалога по кнопке **OK** выдавалось сообщение о выбранном переключателе. Изменения в файле приведены в листинге 6.41.

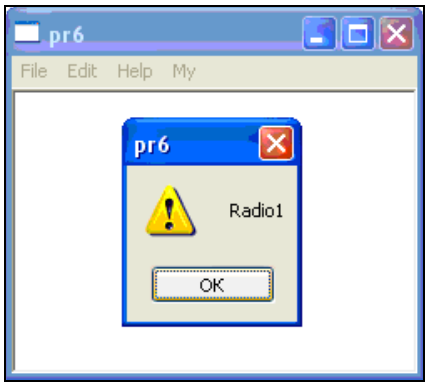
Листинг 6.41. Изменения в файле ChildView.cpp для обработки значений переключателей после закрытия окна диалога

```
void CChildView::OnMyDlg()
{
   // ...
   switch (nRet)
   {
      case IDOK:
                     // ...
                     switch(dlg.radio1)
                     {
                        case 0:
                                 AfxMessageBox(L"Radio1");
                                 break;
```

```
                    case 1:
                        AfxMessageBox(L"Radio2");
                        break;
                    case 2:
                        AfxMessageBox(L"Radio3");
                        break;
                }
                break;
        case IDCANCEL:
                AfxMessageBox(L"CANCEL");
                break;
    }
}
```



а



б

Рис. 6.38. Работа с группой переключателей (*а*) и обработка нажатия кнопки **OK** (*б*)

Примечание

Хотя переменная `radio1` имеет тип `BOOL`, она содержит индекс (с 0) выбранной кнопки в группе.

Результат работы программы показан на рис. 6.38.

6.1.8. Элементы оформления: надпись (*Static Text*) и групповой блок (*Group Box*)

Добавим элемент управления **Static Text** (рис. 6.39, *а*). В окне свойств в поле **Caption** зададим текст надписи: `1-10` (это границы значения элемента `edit2`) (рис. 6.39, *б*).

Добавим рамку для группового блока **Group Box** (рис. 6.40, *а*). В окне свойств в поле **Caption** зададим текст надписи `Группа` (рис. 6.40, *б*).

Окончательный вид окна диалога при запуске программы показан на рис. 6.41.

Примечание

Элементы **Static Text** и **Group Box** служат только для оформления. Их идентификаторы даже не добавляются в файл Resource.h.

Список всех используемых в этой главе элементов управления показан на рис. 6.42.

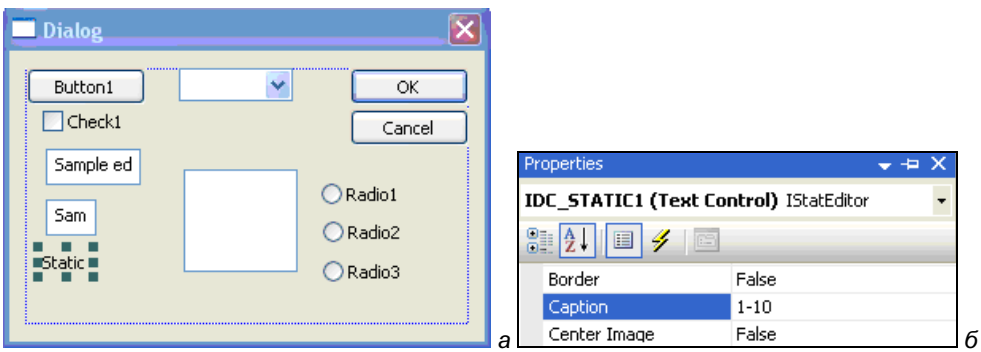
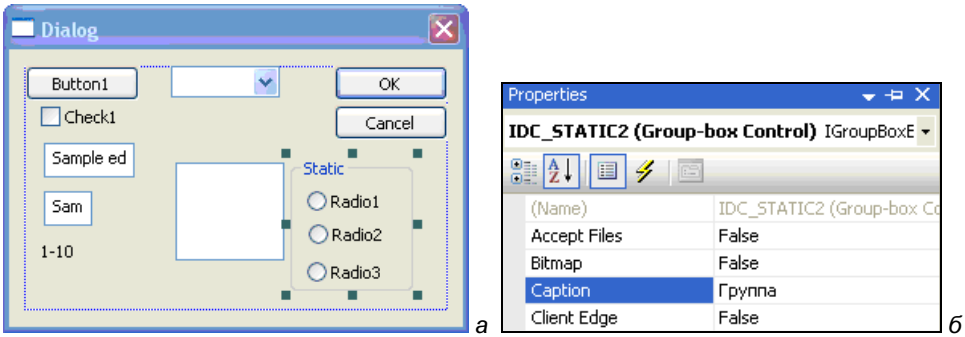


Рис. 6.39. Добавление статической надписи (*а*) и изменение ее текста с помощью окна свойств (*б*)



а *б*

Рис. 6.40. Добавление рамки группового блока (*а*) и изменение текста надписи с помощью окна свойств (*б*)

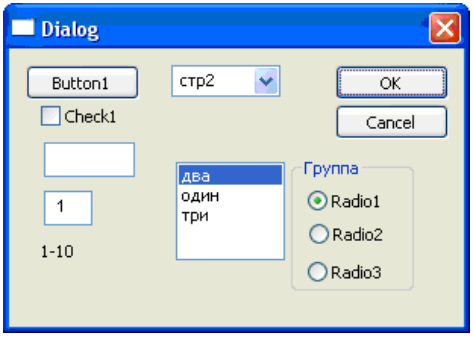


Рис. 6.41. Окончательный вид окна диалога

Рис. 6.42. Список всех используемых в окне диалога элементов

6.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 6.42. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDD_DIALOG1        130
#define IDC_BUTTON1        1001
#define IDC_CHECK1         1003
#define IDC_EDIT1          1004
#define IDC_EDIT2          1005
#define IDC_COMBO1         1006
#define IDC_LIST1          1007
#define IDC_RADIO1         1008
#define IDC_RADIO2         1009
#define IDC_RADIO3         1010
#define ID_MY_DLG          32771
// ...
```

Листинг 6.43. Файл MyDlg.h (объявление класса диалога, добавлен мастером)

```
// MyDlg.h
#pragma once
#include "afxwin.h"

// MyDlg dialog
class MyDlg : public CDialog
{
   DECLARE_DYNAMIC(MyDlg)

public:
   MyDlg(CWnd* pParent = NULL);   // standard constructor
   virtual ~MyDlg();
```

```
// Dialog Data
   enum { IDD = IDD_DIALOG1 };

protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

   DECLARE_MESSAGE_MAP()
public:
   afx_msg void OnBnClickedButton1();
public:
   BOOL check1;
public:
   CString edit1;
public:
   CEdit cedit1;
public:
   afx_msg void OnBnClickedOk();
public:
   int edit2;
public:
   CString combo1;
public:
   virtual BOOL OnInitDialog();
public:
   CComboBox ccombo1;
public:
   CString list1;
public:
   CListBox clist1;
public:
   BOOL radio1;
};
```

Листинг 6.44. Файл MyDlg.cpp (определение класса диалога, добавлен мастером)

```
// MyDlg.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "pr6.h"
#include "MyDlg.h"

// MyDlg dialog

IMPLEMENT_DYNAMIC(MyDlg, CDialog)

MyDlg::MyDlg(CWnd* pParent /*=NULL*/): CDialog(MyDlg::IDD, pParent)
   , check1(FALSE)
   , edit1(_T(""))
   , edit2(1)
   , combo1(_T(""))
   , list1(_T(""))
   , radio1(FALSE)
{
}

MyDlg::~MyDlg()
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   DDX_Check(pDX, IDC_CHECK1, check1);
   DDX_Text(pDX, IDC_EDIT1, edit1);
   DDV_MaxChars(pDX, edit1, 7);
   DDX_Control(pDX, IDC_EDIT1, cedit1);
   DDX_Text(pDX, IDC_EDIT2, edit2);
   DDV_MinMaxInt(pDX, edit2, 1, 10);
   DDX_CBString(pDX, IDC_COMBO1, combo1);
   DDX_Control(pDX, IDC_COMBO1, ccombo1);
   DDX_LBString(pDX, IDC_LIST1, list1);
   DDX_Control(pDX, IDC_LIST1, clist1);
   DDX_Radio(pDX, IDC_RADIO1, radio1);
}
```

```
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
   ON_BN_CLICKED(IDOK, &MyDlg::OnBnClickedOk)
END_MESSAGE_MAP()

// MyDlg message handlers

void MyDlg::OnBnClickedButton1()
{
   // TODO: Add your control notification handler code here
   AfxMessageBox(L"Button1");
}

BOOL MyDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // TODO:  Add extra initialization here

   // Заполнить выпадающий список
   ccombo1.AddString(L"стр1");
   ccombo1.AddString(L"стр2");
   ccombo1.AddString(L"стр3");
   // Сделать по умолчанию "стр2"
   ccombo1.SetCurSel(1);

   // Заполнить список
   clist1.AddString(L"один");
   clist1.AddString(L"два");
   clist1.AddString(L"три");
   // Сделать по умолчанию "два" (список автоматически отсортирован по
   // алфавиту: два [0], один [1],три [2])
   clist1.SetCurSel(0);
   return TRUE;  // return TRUE unless you set the focus to a control
   // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
void MyDlg::OnBnClickedOk()
{
   // TODO: Add your control notification handler code here
   if(!cedit1.GetWindowTextLengthW())
   {
      AfxMessageBox(L"Заполните edit1" );
      // Передать управление edit1
      GotoDlgCtrl(&cedit1);
      return;
   }
   OnOK();
}
```

Листинг 6.45. Файл ChildView.h (объявление класса представления)

```
// ...
class CChildView : public CWnd
{
// ...
public:
   afx_msg void OnMyDlg();
};
```

Листинг 6.46. Файл ChildView.cpp (определение класса представления)

```
// ...
#include "ChildView.h"
#include "MyDlg.h"
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_COMMAND(ID_MY_DLG, &CChildView::OnMyDlg)
END_MESSAGE_MAP()
// ...
void CChildView::OnMyDlg()
{
```

```
    // TODO: Add your command handler code here
    MyDlg dlg;
    INT_PTR nRet;
    nRet = dlg.DoModal();
    switch (nRet)
    {
       case IDOK:
                    // AfxMessageBox(L"OK");
                    if(dlg.check1)
                       AfxMessageBox(L"check1");
                    if(dlg.edit1.GetLength())
                       AfxMessageBox(dlg.edit1);
                    if(dlg.combo1.GetLength())
                       AfxMessageBox(dlg.combo1);
                    if(dlg.list1.GetLength() )
                       AfxMessageBox(dlg.list1);
                    switch(dlg.radio1)
                    {
                       case 0:
                                AfxMessageBox(L"Radio1");
                                break;
                       case 1:
                                AfxMessageBox(L"Radio2");
                                break;
                       case 2:
                                AfxMessageBox(L"Radio3");
                                break;
                    }
                    break;
       case IDCANCEL:
                    AfxMessageBox(L"CANCEL");
                    break;
    }
}
```

ГЛАВА 7

Дополнительные элементы управления диалоговых окон

Создайте проект **pr7** аналогично проекту **pr1** *(см. разд. 1.1)*. Надо изменить следующие опции относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ;
  - без поддержки архитектуры документ/представление;
- на вкладке **User Interface Features**:
  - без строки статуса;
  - без панели инструментов.

7.1. Описание программы

Добавить в проект диалоговое окно `IDD_DIALOG1`, класс диалога `MyDlg`, меню для его вызова **My | Dlg** *(см. разд. 6.1.1)* и функцию `OnInitDialog()` *(см. разд. 6.1.5)*.

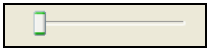
В этой главе рассмотрим следующие элементы управления:

- **Picture Control** — рисунок (`class CStatic: public CWnd`);

- **Horizontal Scroll Bar** — горизонтальная полоса прокрутки (`class CScrollBar : public CWnd`);

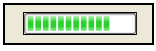
- **Vertical Scroll Bar** — вертикальная полоса прокрутки (`class CScrollBar : public CWnd`);

- **Slider Control** — регулятор (`class CSliderCtrl : public CWnd`);



- **Spin Control** — счетчик (`class CSpinButtonCtrl : public CWnd`);

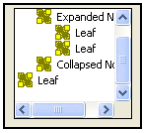
- **Progress Control** — индикатор (`class CProgressCtrl : public CWnd`);



- **Hot Key** — быстрая клавиша (`class CHotKeyCtrl : public CWnd`);

- **List Control** — список (`class CListCtrl : public CWnd`);

- **Tree Control** — дерево (`class CTreeCtrl : public CWnd`).

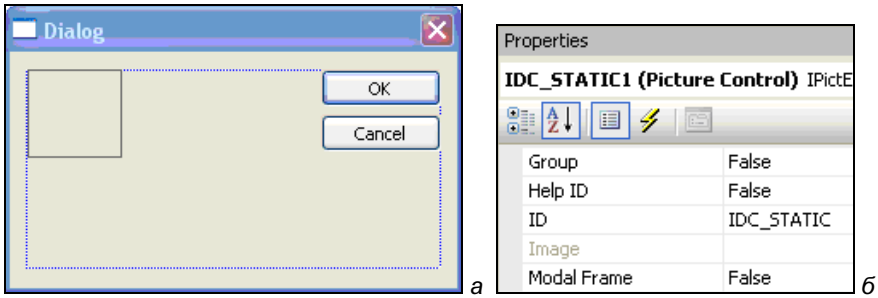


7.1.1. Рисунок (*Picture Control*)

Добавим на окно диалога рисунок **Picture Control** (рис. 7.1).

Теперь надо создать новый ресурс для рисунка размером 35 × 35. Для этого нужно:

- в окне **Resource View** вызвать контекстное меню для папки **pr7.rc** и выполнить команду **Add Resource** (рис. 7.2, *а*);

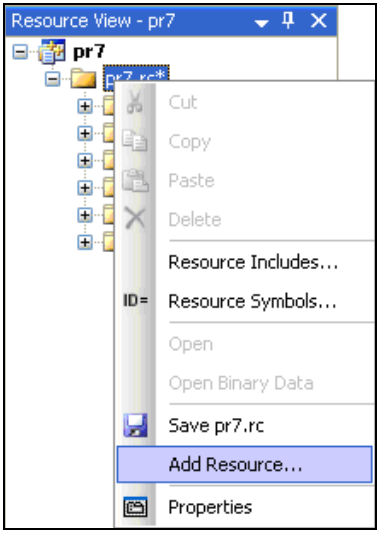


а *б*

Рис. 7.1. Добавление элемента управления **Picture Control** в окно диалога (*а*) и просмотр его свойств (*б*)

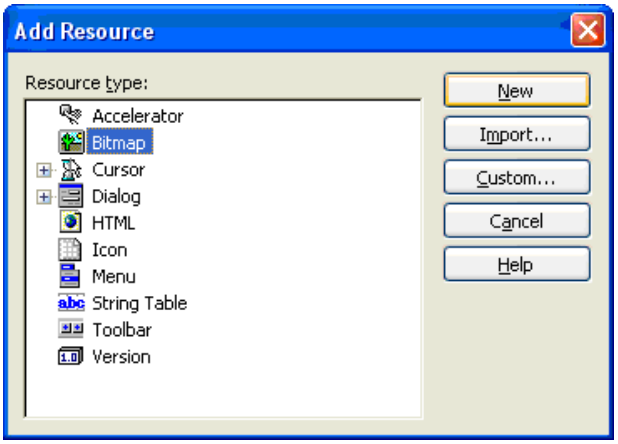
- в окне **Add Resource** выбрать тип ресурса **Bitmap** и нажать кнопку **New** (рис. 7.2, *б*);
- в окне свойств нового ресурса (`IDB_BITMAP1`) в поле **Height** задать `35`, в поле **Width** тоже задать `35` (рис. 7.2, *в*);
- нарисовать рисунок (рис. 7.2, *в*).

Для работы с рисунком надо в окне свойств ресурса `IDB_BITMAP1` изменить значение поля **Type** (Тип) с `Frame` на `Bitmap` (рис. 7.3).

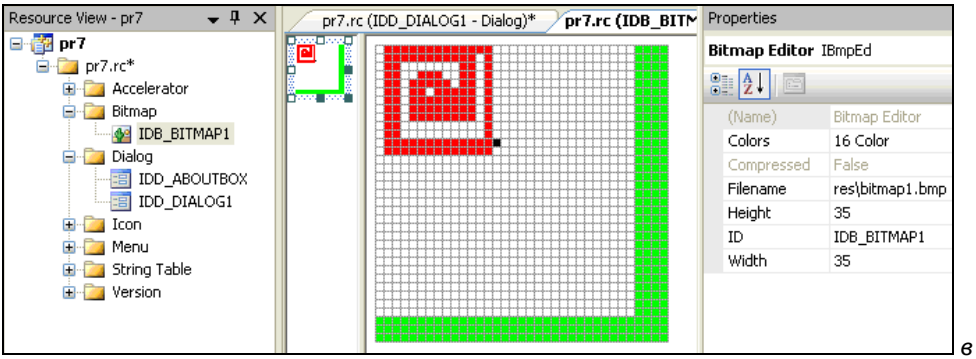


а

Рис. 7.2. Добавление нового ресурса для рисунка с помощью контекстного меню (*а*)



б



в

Рис. 7.2. Добавление нового ресурса для рисунка с помощью окна **Add Resource** (*б*), задание его свойств и самого рисунка (*в*)

По умолчанию предлагается простая рамка заданного цвета (`Frame`) и, если это свойство не изменить, *рисунок отображаться не будет*. Теперь надо связать элемент **Picture Control** (`IDC_STATIC`) с рисунком (`IDB_BITMAP1`). Изменения в файлах приведены в листингах 7.1—7.3.

Листинг 7.1. Изменения в файле Resource.h для работы с рисунком

```
// ...
#define IDB_BITMAP1                     131
// ...
```

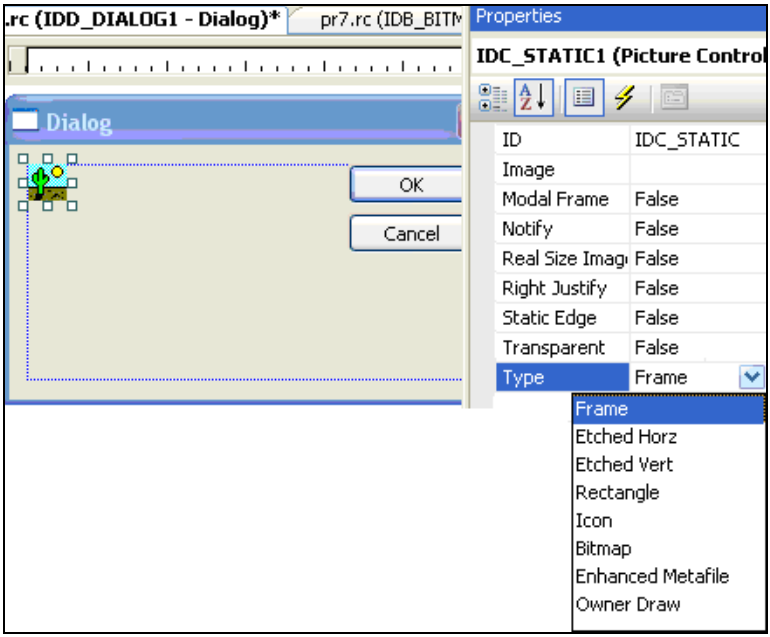


Рис. 7.3. Изменение типа рисунка

Листинг 7.2. Изменения в файле MyDlg.h для работы с рисунком

```
// ...
class MyDlg : public CDialog
{
//...
public:
   CBitmap bmp;
};
```

Листинг 7.3. Изменения в файле MyDlg.cpp для работы с рисунком

```
BOOL MyDlg::OnInitDialog()
{
   CDialog::OnInitDialog();
   // TODO:  Add extra initialization here
```

```
    CStatic *pbmp = (CStatic *)this->GetDlgItem(IDC_STATIC);
    bmp.LoadBitmapW(IDB_BITMAP1);
    pbmp->SetBitmap(bmp.operator HBITMAP());

    return TRUE;  // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}
```

Функции `GetDlgItem()` и `LoadBitmapW()` были описаны в *гл. 6* и в *гл. 3*, соответственно. Функции для работы с битовым массивом (рисунком) описаны далее.

Задать дескриптор битового массива для его вывода в рамках статического элемента можно с помощью функции:

```
HBITMAP CStatic::SetBitmap(    // Дескриптор предыдущего битового массива
                               // (NULL - если его нет)
   HBITMAP hBitmap);           // Дескриптор задаваемого битового массива
```

Получить текущий дескриптор, установленный с помощью `SetBitmap()`, можно функцией:

```
HBITMAP CStatic::GetBitmap() const;
```

Дескриптор битового массива (возвращает его или 0 при ошибке) можно получить с помощью функции:

```
operator CBitmap::HBITMAP() const;
```

Функции для отображения иконок в рамках статического элемента следующие:

```
HICON CStatic::SetIcon(HICON hIcon);
HICON CStatic::GetIcon() const;
```

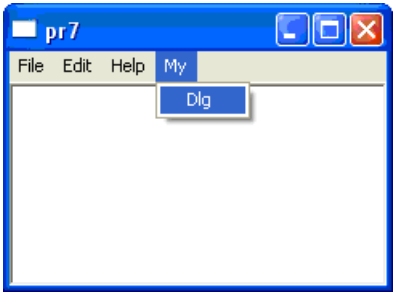
Функции для отображения курсоров в рамках статического элемента:

```
HCURSOR CStatic::SetCursor(HCURSOR hCursor);
HCURSOR CStatic::GetCursor();
```

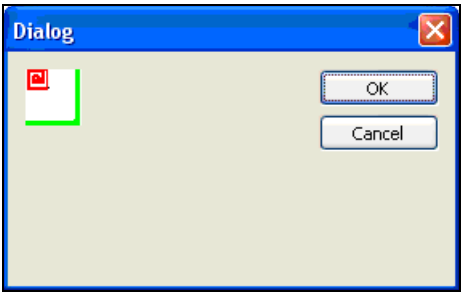
Результат работы программы показан на рис. 7.4.

Работать с элементами управления (любыми) можно не только в диалоговом окне. Разница в том, что в диалоговом окне они создаются мастером, а для других окон их надо создавать "вручную" (с помощью функции `Create()`). Сделаем так, чтобы при нажатии кнопки **OK** в окне диалога этот же рисунок

появлялся в окне представления, а при нажатии кнопки **Cancel** в окне диалога — исчезал. Изменения в файлах приведены в листингах 7.4 и 7.5.



а



б

Рис. 7.4. Открытие окна диалога (*а*) и вид рисунка в окне диалога (*б*)

Листинг 7.4. Изменения в файле ChildView.h для добавления рисунка в окно представления

```
// ...
class CChildView : public CWnd
{
// ...
public:
   CBitmap bmpv;
   CStatic sbmp;
};
```

Листинг 7.5. Изменения в файле ChildView.cpp для добавления рисунка в окно представления

```
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;
   INT_PTR nRet;
   nRet = dlg.DoModal();
```

```
   static int n;        // Флаг показа картинки (0 - создать и показать,
                        // 1 - стереть и удалить
   if(nRet == IDOK)
   {
      // Работа с рисунком
      if(!n)
      {
         bmpv.LoadBitmapW(IDB_BITMAP1);
         sbmp.Create(NULL, WS_CHILD | WS_VISIBLE |
                     SS_BITMAP | SS_CENTERIMAGE,
                     CRect(50,50,85,85), this, IDC_STATIC);
         sbmp.SetBitmap(bmpv.operator HBITMAP());
         n = 1;
      }
   }
   else                          // if(nRet == IDCANCEL)
   {
      bmpv.DeleteObject();
      sbmp.DestroyWindow();
      n = 0;
   }
}
```

Функция создания статического элемента определена следующим образом:

```
virtual BOOL CStatic::Create(   // 0 - ошибка
   LPCTSTR lpszText,            // Текст, выводимый в элемент управления
   DWORD dwStyle,               // Стиль статического элемента
   const RECT& rect,            // Положение и размер статического
                                // элемента
   CWnd* pParentWnd,            // Указатель на родительское окно
   UINT nID = 0xffff);          // Идентификатор статического элемента
```

Стили статического элемента (`dwStyle`) могут быть такими:

- `SS_BITMAP` — обязательный стиль для битовых массивов;
- `SS_BLACKFRAME` — задает прямоугольник того же цвета, что и рамка окна (по умолчанию это черный цвет);

- `SS_BLACKRECT` — задает прямоугольник, закрашенный цветом, совпадающим с цветом рамки окна (по умолчанию это черный цвет);
- `SS_CENTER` — располагает выводимый текст в центре статического элемента. Слова, не помещающиеся на одной строке, автоматически переносятся на следующую строку;
- `SS_CENTERIMAGE` — располагает изображение в центре статического элемента;
- `SS_GRAYFRAME` — задает прямоугольник с рамкой того же цвета, что и цвет фона клиентской области родительского окна (для диалога этот цвет по умолчанию серый);
- `SS_GRAYRECT` — задает прямоугольник, закрашенный цветом, совпадающим с цветом фона клиентской области родительского окна (для диалога этот цвет по умолчанию серый);
- `SS_ICON` — обязательнй стиль для курсоров и иконок;
- `SS_LEFT` — располагает выводимый текст, выравнивая его по левому краю статического элемента. Слова, не помещающиеся на одной строке, автоматически переносятся на следующую строку;
- `SS_LEFTNOWORDWRAP` — располагает выводимый текст, выравнивая его по левому краю статического элемента. Слова, не помещающиеся на одной строке, не переносятся на следующую строчку, а обрезаются;
- `SS_NOPREFIX` — если этот стиль не определен, то в тексте буква, следующая за символом `'&'`, не подчеркивается (альтернативная комбинация клавиш с <Alt>);
- `SS_RIGHT` — располагает выводимый текст, выравнивая его по правому краю статического элемента. Слова, не помещающиеся на одной строке, автоматически переносятся на следующую строку;
- `SS_RIGHTJUST` — определяет, что правый нижний угол статического элемента управления со стилем `SS_BITMAP` или `SS_ICON` должен остаться фиксированным, когда элемент управления изменит размеры. Только верхняя и левая стороны могут быть скорректированы;
- `SS_WHITEFRAME` — задает прямоугольник с рамкой, которая по умолчанию белого цвета;
- `SS_WHITERECT` — задает прямоугольник, закрашенный белым цветом.

Размер статического элемента должен соответствовать рисунку (или быть больше). Если задать размер меньше, то рисунок будет "обрезан". Для нашей

программы положение рисунка относительно родительского окна (представления) — (50, 50). Размер рисунка 35 × 35, поэтому при создании рисунка в функции `Create()` используются размеры области `CRect(50,50,85,85)` (`50+35=85`, `50+35=85`). Параметр `pParentWnd` должен быть обязательно определен (он не может быть равен `NULL`).

Удаление загруженного (подключенного) объекта выполняется с помощью функции:

```
BOOL CGdiObject::DeleteObject();                    // 0 - ошибка
```

Разрушение окна (элемента управления) выполняет функция:

```
virtual BOOL CWnd::DestroyWindow();                 // 0 - ошибка
```

Результат работы программы показан на рис. 7.5.

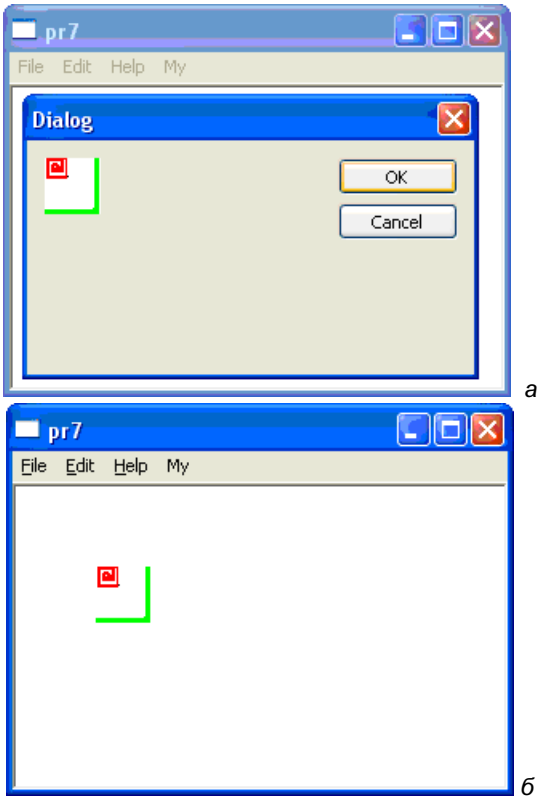
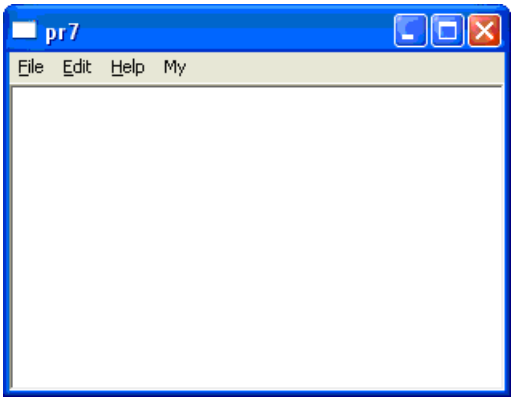


Рис. 7.5. Вид окна диалога с элементом управления **Picture Control** (*а*), появление рисунка в окне представления при нажатии кнопки **OK** в окне диалога (*б*)

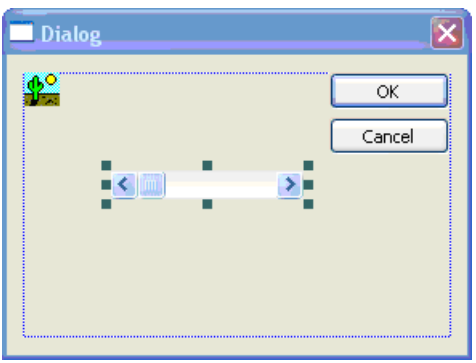


в

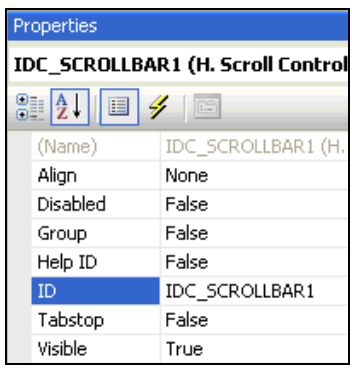
Рис. 7.5. Удаление рисунка из окна представления при нажатии кнопки **Cancel** в окне диалога (*в*)

7.1.2. Горизонтальная полоса прокрутки (*Horizontal Scroll Bar*)

Добавим горизонтальную полосу прокрутки **Horizontal Scroll Bar** (рис. 7.6) и переменную `scrollbar1` (категории **Value**), которая будет связана с этим элементом управления (рис. 7.7). Изменения в файлах приведены в листингах 7.6—7.8.



а



б

Рис. 7.6. Добавление горизонтальной полосы прокрутки в окно диалога (*а*) и просмотр ее свойств (*б*)

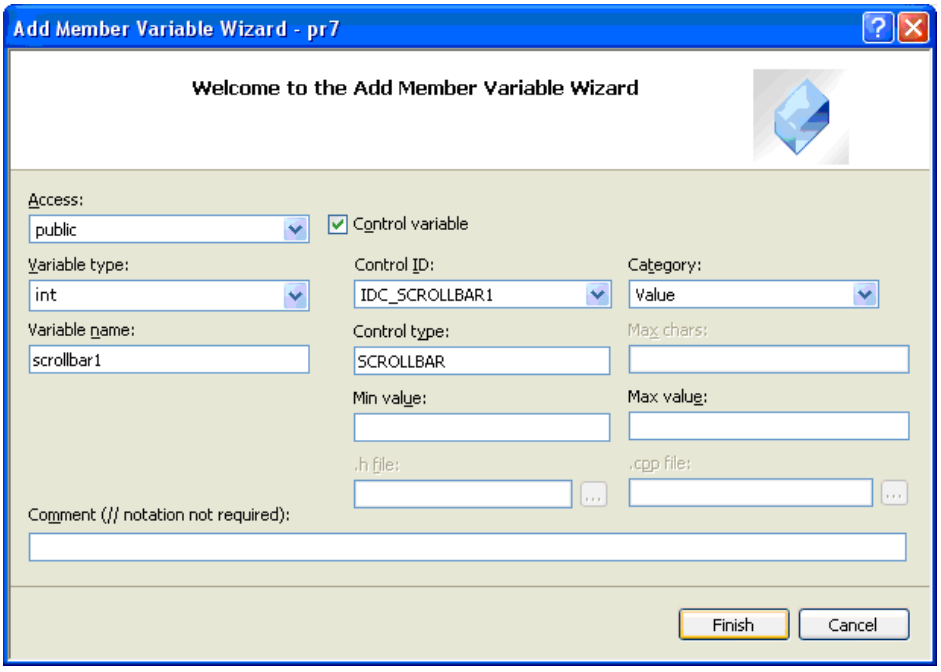


Рис. 7.7. Добавление переменной, связанной с горизонтальной полосой прокрутки

Листинг 7.6. Изменения в файле Resource.h для работы с горизонтальной полосой прокрутки

```
// ...
#define IDC_SCROLLBAR1                  1007
// ...
```

Листинг 7.7. Изменения в файле MyDlg.h для работы с горизонтальной полосой прокрутки

```
class MyDlg : public CDialog
{
// ...
public:
   int scrollbar1;
};
```

Листинг 7.8. Изменения в файле MyDlg для работы с горизонтальной полосой прокрутки

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , scrollbar1(0)
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   DDX_Scroll(pDX, IDC_SCROLLBAR1, scrollbar1);
}
```

Для обработки сообщений от линейки прокрутки надо в класс диалога добавить обработку сообщения `WM_HSCROLL` (рис. 7.8):

- в окне свойств окна диалога выбрать вкладку **Messages**;
- в списке сообщений найти `WM_HSCROLL` и выбрать **<Add> OnHScroll**.

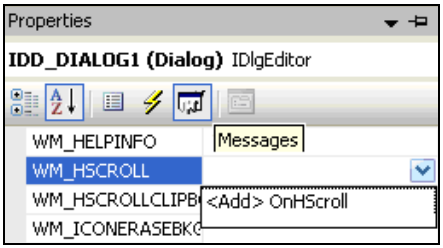


Рис. 7.8. Добавление обработки сообщения горизонтальной линейки прокрутки

Изменения в файлах приведены в листингах 7.9 и 7.10.

Листинг 7.9. Изменения в файле MyDlg.h для контроля сообщений горизонтальной полосы прокрутки

```
class MyDlg : public CDialog
{
// ...
```

```
public:
   afx_msg void OnHScroll(UINT nSBCode, UINT nPos,
                          CScrollBar* pScrollBar);
};
```

Листинг 7.10. Изменения в файле MyDlg.cpp для контроля сообщений горизонтальной полосы прокрутки

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_WM_HSCROLL()
END_MESSAGE_MAP()
// ...
void MyDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
   // TODO: Add your message handler code here and/or call default

   CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

Добавим обработку действий с линейкой прокрутки (сообщения и функции для линейки прокрутки описаны в *гл. 5*). Изменения в файле приведены в листинге 7.11.

Листинг 7.11. Изменения в файле MyDlg.cpp для обработки сообщений горизонтальной полосы прокрутки

```
// ...
void MyDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
   // TODO: Add your message handler code here and/or call default

   // Только для линейки прокрутки
   if(pScrollBar->m_hWnd == GetDlgItem(IDC_SCROLLBAR1)->m_hWnd)
   {
      // Установка значений линейки ( от 0 до 10 )
      pScrollBar->SetScrollRange(0, 10);
```

```
      switch(nSBCode)                  // Что было нажато на линейке
      {
         // Зацепили и потащили ползунок
         case SB_THUMBPOSITION:
                              // Выставить ползунок в текущую позицию
                              pScrollBar->SetScrollPos(nPos);
                              break;
         // Нажали на стрелку влево
         case SB_LINELEFT:
                              nPos = pScrollBar->GetScrollPos();
                              nPos--;
                              pScrollBar->SetScrollPos(nPos);
                              break;
         // Нажали на стрелку вправо
         case SB_LINERIGHT:
                              nPos = pScrollBar->GetScrollPos();
                              nPos++;
                              pScrollBar->SetScrollPos(nPos);
                              break;
      }
   }

   CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

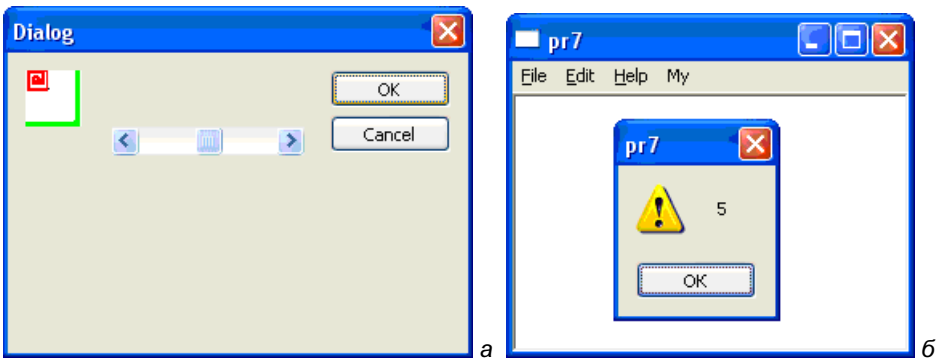
Теперь обработаем значение `scrollbar1`. Изменения в файле приведены в листинге 7.12.

Листинг 7.12. Изменения в файле ChildView.cpp для получения значения состояния полосы прокрутки в момент закрытия окна диалога

```
void CChildView::OnMyDlg()
{
   // ...
   if(nRet == IDOK)
```

```
    {
        // ...
        // Работа с линейкой прокрутки
        CString str1;
        str1.Format(L"%d", dlg.scrollbar1);
        AfxMessageBox(str1);
    }
    // ...
}
```

Результат работы программы показан на рис. 7.9.



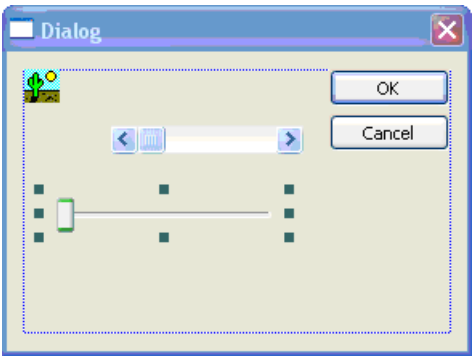
а *б*

Рис. 7.9. Вид окна диалога с элементом управления **Horizontal Scroll Bar** (*а*) и обработка нажатия кнопки **OK** (*б*)

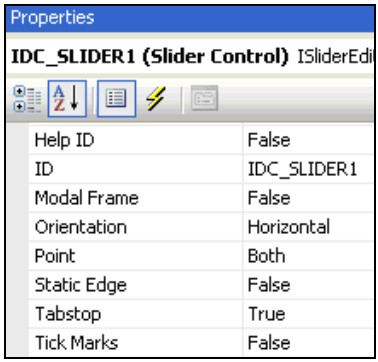
Добавление вертикальной полосы прокрутки **Vertical Scroll Bar** делается аналогично, только с заменой сообщения `WM_HSCROLL` на `WM_VSCROLL`, стиль `SB_LINELEFT` на `SB_LINEUP`, `SB_LINERIGHT` на `SB_LINEDOWN`.

7.1.3. Регулятор (*Slider Control*)

Добавим регулятор **Slider Control** (рис. 7.10) и переменную (категории **Value**) `slider1`, которая будет связана с этим элементом управления (рис. 7.11). Изменения в файлах приведены в листингах 7.13—7.15.



а



б

Рис. 7.10. Добавление регулятора в окно диалога (*а*) и просмотр его свойств (*б*)

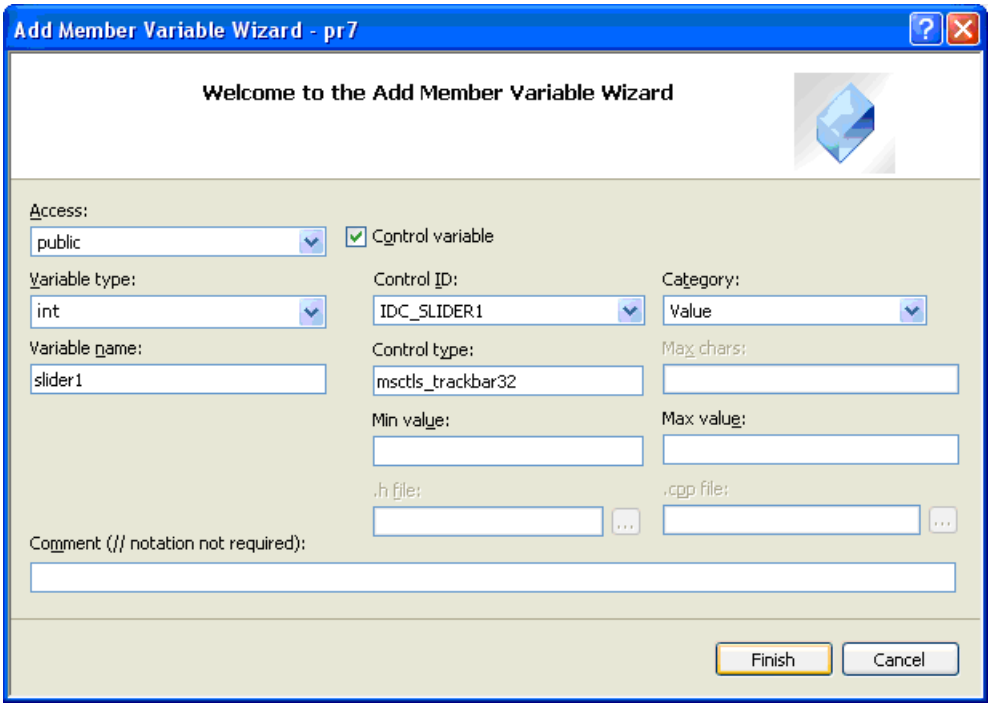


Рис. 7.11. Добавление переменной, связанной с регулятором

Листинг 7.13. Изменения в файле Resource.h для работы с регулятором

```
// ...
#define IDC_SLIDER1                     1009
// ...
```

Листинг 7.14. Изменения в файле MyDlg.h для работы с регулятором

```
class MyDlg : public CDialog
{
// ...
public:
   int slider1;
};
```

Листинг 7.15. Изменения в файле MyDlg.cpp для работы с регулятором

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , scrollbar1(0)
   , slider1(0)
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Slider(pDX, IDC_SLIDER1, slider1);
}
```

Зададим диапазон значений регулятора (от 0 до 10). Изменения в файле приведены в листинге 7.16.

Листинг 7.16. Изменения в файле MyDlg.cpp для установки диапазона значений регулятора

```
BOOL MyDlg::OnInitDialog()
{
```

```
   // ...
   // Регулятор
   CSliderCtrl *psl = (CSliderCtrl *)this->GetDlgItem(IDC_SLIDER1);
   psl->SetRangeMin(0);
   psl->SetRangeMax(10);
   // ...
}
```

Установка минимального значения регулятора выполняется с помощью функции:

```
void CSliderCtrl::SetRangeMin(
   int nMin,                  // Минимальное значение
   BOOL bRedraw = FALSE);     // TRUE - перерисовать сразу, FALSE - нет
```

Установка минимального значения регулятора:

```
void CSliderCtrl::SetRangeMax(
   int nMax,                  // Максимальное значение
   BOOL bRedraw = FALSE);     // TRUE - перерисовать сразу, FALSE - нет
```

Теперь обработаем значение `slider1`. Изменения в файле приведены в листинге 7.17.

Листинг 7.17. Изменения в файле ChildView.cpp для просмотра значения регулятора в момент закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
   // ...
   if(nRet == IDOK)
   {
      // ...
      // Работа с регулятором
      str1.Format(L"%d", dlg.slider1);
      str1 += " (slider1)";
      AfxMessageBox(str1);
   }
   // ...
}
```

Результат работы программы показан на рис. 7.12.

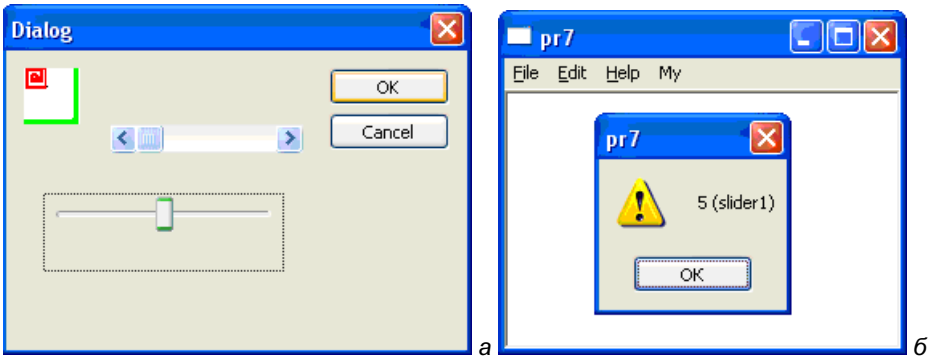


Рис. 7.12. Вид окна диалога с элементом управления **Slider Control** (*а*) и обработка нажатия кнопки **OK** (*б*)

Для удобства работы добавим шкалу разметки и сделаем так, чтобы над регулятором появлялось текущее значение в процентах. Для разметки надо изменить в окне свойств регулятора в поле **Tick Marks** (Разметка) значение с `False` на `True` и в поле **Auto Ticks** значение с `False` на `True` (количество делений шкалы автоматически рассчитывается из максимального и минимального значений регулятора, *если они заданы*), как показано на рис. 7.13. Для вывода текущего значения надо в класс диалога добавить обработку сообщения `WM_PAINT` (рис. 7.14):

- в окне свойств окна диалога выбрать вкладку **Messages**;
- в списке сообщений найти `WM_PAINT` и выбрать **<Add>OnPaint**.

Изменения в файлах приведены в листингах 7.18 и 7.19.

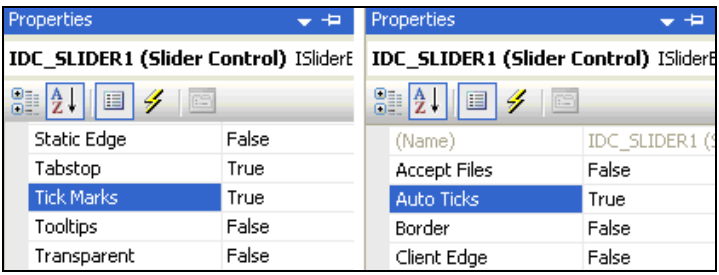


Рис. 7.13. Добавление разметки при работе с регулятором

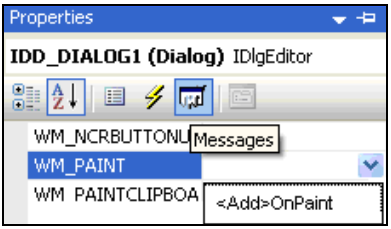


Рис. 7.14. Добавление обработки сообщения о перерисовке окна диалога

Листинг 7.18. Изменения в файле MyDlg.h для отображения текущего значения регулятора в окне диалога

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnPaint();
};
```

Листинг 7.19. Изменения в файле MyDlg.cpp для отображения текущего значения регулятора в окне диалога

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_WM_PAINT()
END_MESSAGE_MAP()
// ...
void MyDlg::OnPaint()
{
   CPaintDC dc(this); // device context for painting
   // TODO: Add your message handler code here
   // Вывод значений регулятора
   CSliderCtrl *psl = (CSliderCtrl *)this->GetDlgItem(IDC_SLIDER1);
   CString str1;
```

```
    str1.Format(L"%d", psl->GetPos());
    str1 += " %";
    // Закрашиваем ("стираем") поле вывода текста
    dc.FillSolidRect(30, 70, 35, 15,RGB(230, 230, 0));
    dc.TextOutW(30, 70, str1);

    // Do not call CDialog::OnPaint() for painting messages
}
```

Сообщения от регулятора обрабатываются при генерации сообщения WM_HSCROLL. Функция обработки этого сообщения уже есть (она задавалась при обработке линейки прокрутки). Изменения в файле приведены в листинге 7.20.

Листинг 7.20. Изменения в файле MyDlg.cpp для вывода значения регулятора сразу после его изменения

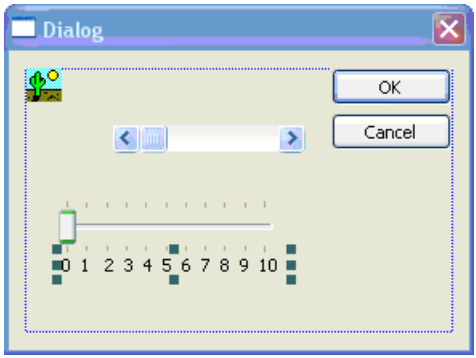
```
// ...
void MyDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // ...
    // Только для линейки прокрутки
    if(pScrollBar->m_hWnd == GetDlgItem(IDC_SCROLLBAR1)->m_hWnd)
    {
        // ...
    }
    else if(pScrollBar->m_hWnd == GetDlgItem(IDC_SLIDER1)->m_hWnd)
    {
        // Для вывода значения регулятора посылаем сообщений о перерисовке
        // окна (без перерисовки всего окна)
        this->Invalidate(FALSE);
    }

    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

Заполнения прямоугольной области заданным цветом выполняется функцией:

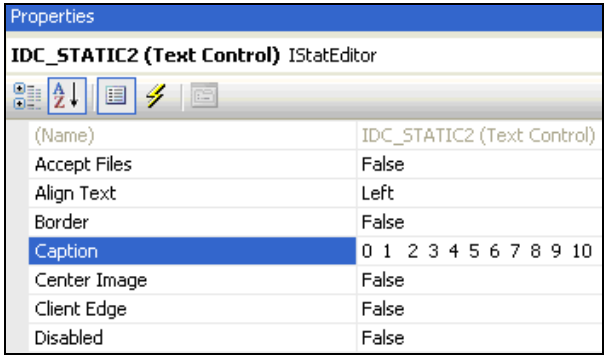
```
void CDC::FillSolidRect(
   LPCRECT lpRect,      // Координаты левого верхнего и правого нижнего
                        // углов области
   COLORREF clr);       // Цвет для заполнения
```

или

```
void CDC::FillSolidRect(
   int x,               // Координаты левого верхнего угла области
   int y,
   int cx,              // Ширина области
   int cy,              // Высота области
   COLORREF clr);       // Цвет для заполнения
```



а



б

Рис. 7.15. Добавление значений шкалы для регулятора с помощью элемента управления **Static Text** (*а*) и его свойства **Caption** (*б*)

Теперь для наглядности добавим в окне диалога значения шкалы (рис. 7.15). Для этого нужно:

- добавить в окно диалога элемент **Static Text** и расположить его под регулятором;
- в окне свойств **Static Text** в поле **Caption** ввести значения (через пробел) `"1 2 3 4 5 6 7 8 9 10"`.

Результат работы программы показан на рис. 7.16.

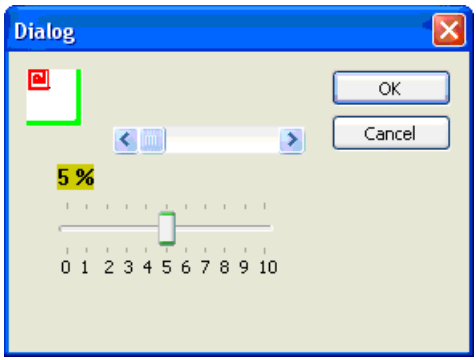


Рис. 7.16. Работа регулятора
с печатью его текущего значения в процентах

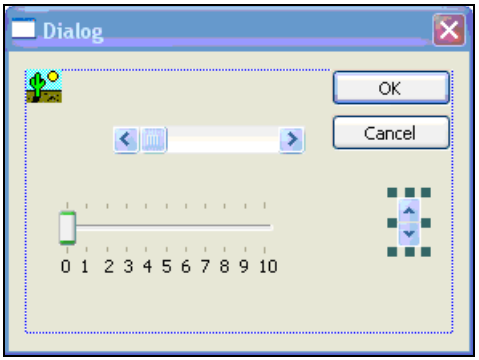
7.1.4. Счетчик (*Spin Control*)

Добавим счетчик **Spin Control** (рис. 7.17). Для счетчика *нельзя добавить* контролируемую переменную категории **Value**. Добавим на окно диалога рядом со счетчиком текстовое поле **Edit Control** (рис. 7.18) и переменную (категории **Value**) `edit1` для него (рис. 7.19).

Изменения в файлах приведены в листингах 7.21—7.23.

Листинг 7.21. Изменения в файле Resource.h для работы со счетчиком и отображением его значения

```
// ...
#define IDC_SPIN1                       1010
#define IDC_EDIT1                       1011
// ...
```



а *б*

Рис. 7.17. Добавление счетчика в окно диалога (*а*) и просмотр его свойств (*б*)

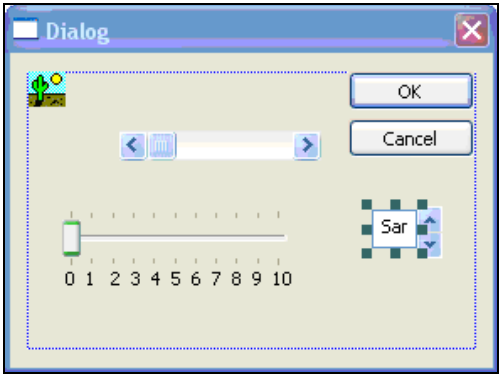


Рис. 7.18. Добавление текстового поля для отображения значения счетчика

Листинг 7.22. Изменения в файле MyDlg.h для отображения значения счетчика

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   int edit1;
};
```

Листинг 7.23. Изменения в файле MyDlg.cpp для отображения значения счетчика

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , scrollbar1(0) , slider1(0)
   , edit1(0)
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   //...
   DDX_Text(pDX, IDC_EDIT1, edit1);
}
```

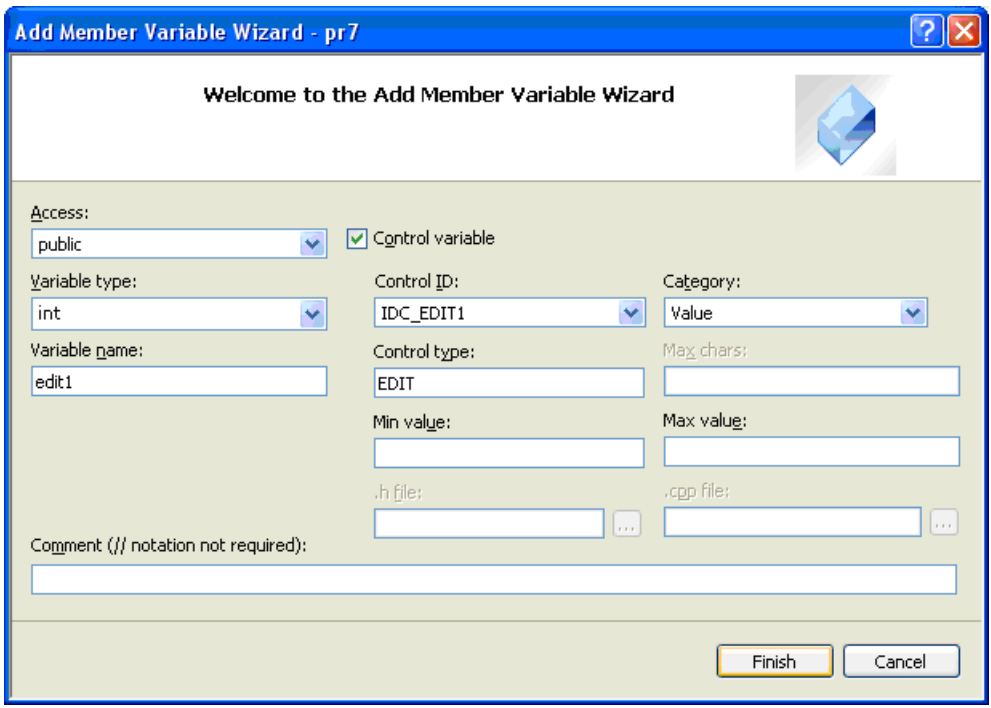


Рис. 7.19. Добавление переменной, связанной с текстовым полем для отображения значения счетчика

Теперь выставим диапазон значений для счетчика и сделаем его привязку к текстовому полю (листинг 7.24).

Листинг 7.24. Изменения в файле MyDlg.cpp для установки диапазона счетчика и его связи с текстовым полем

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Спин
   CSpinButtonCtrl *pspin =
                          (CSpinButtonCtrl *)this->GetDlgItem(IDC_SPIN1);
   // Диапазон значений для счетчика
   pspin->SetRange(0,10);
   // Начальное значение
   pspin->SetPos(0);
   CEdit *pedit1 = (CEdit *)GetDlgItem(IDC_EDIT1);
   // Привязка счетчика к окну редактирования
   pspin->SetBuddy(pedit1);

   return TRUE;
}
```

Установка диапазона значений счетчика выполняется с помощью функции:

```
void CSpinButtonCtrl::SetRange(
   short nLower,                    // Минимальное значение
   short nUpper);                   // Максимальное значение
void CSpinButtonCtrl::SetRange32(
   int nLower,
   int nUpper);
```

Установка текущей позиции счетчика выполняется функцией:

```
int CSpinButtonCtrl::SetPos(        // Предыдущая позиция
   int nPos);                       // Новая (устанавливаемая) позиция
int CSpinButtonCtrl::SetPos32(
   int nPos);
```

Установка связи спина с окном:

```
CWnd* CSpinButtonCtrl::SetBuddy(      // Указатель на предыдущее
                                      // связанное со счетчиком окно
   CWnd* pWndBuddy);                  // Указатель на новое связываемое
                                      // окно
```

Для того чтобы привязка (к текстовому полю с целыми числами) заработала, надо *обязательно* изменить в окне свойств счетчика в поле **Set Buddy Integer** (Установить привязку к целым числам) значение `False` на `True` (рис. 7.20). Результат работы программы показан на рис. 7.21.

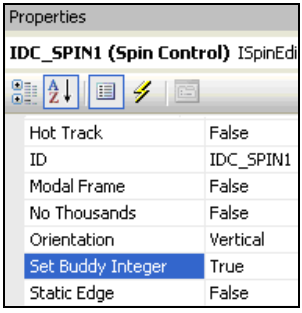


Рис. 7.20. Привязка счетчика к целым значениям

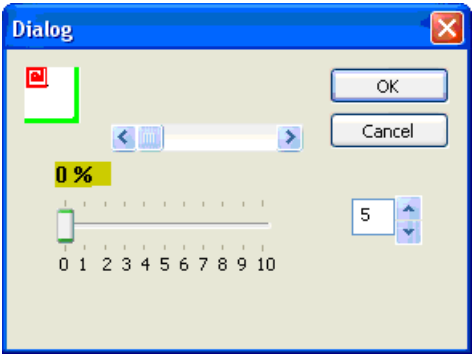


Рис. 7.21. Работа счетчика с целыми числами

Примечание

Если в окне свойств для счетчика в поле **Auto Buddy** (Автоматическая привязка) задать значение `True`, то счетчик будет автоматически привязан к

последнему размещенному в окне диалога элементу редактирования. Но в тексте программы этого видно не будет.

Если нужно контролировать поведение счетчика, то надо добавить в класс диалога обработку сообщения `WM_VSCROLL` и в нем сделать контроль работы счетчика. Пример такой работы показан в листингах 7.25 и 7.26. В программу проекта **pr7** на компакт-диске это не было включено.

Листинг 7.25. Изменения в файле MyDlg.h для контроля значений счетчика

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnVScroll(UINT nSBCode, UINT nPos,
                          CScrollBar* pScrollBar);
};
```

Листинг 7.26. Изменения в файле MyDlg.cpp для контроля значений счетчика

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_WM_VSCROLL()
END_MESSAGE_MAP()
// ...
void MyDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
   // TODO: Add your message handler code here and/or call default

   // Только для счетчика
   if( pScrollBar->m_hWnd == GetDlgItem(IDC_SPIN1)->m_hWnd)
   {
      CEdit *pedit1 = (CEdit *)GetDlgItem(IDC_EDIT1);
      CString s; int z;
```

```
        if(nSBCode == SB_THUMBPOSITION)
        {
            z = (int)nPos;
            s.Format(L"%d", z);
            pedit1->SetWindowTextW(s);
            pedit1->UpdateWindow();
        }
    }
    CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}
```

Изменим значения шага приращения счетчика (по умолчанию он равен 1). Для этого надо добавить сообщение счетчика `UDN_DELTAPOS` (рис. 7.22):

- в окне свойств счетчика выбрать вкладку **Control Events** (Управляющие события);
- в списке событий найти `UDN_DELTAPOS` и выбрать **<Add> OnDeltaposSpin1**.

Изменения в файлах приведены в листингах 7.27 и 7.28.

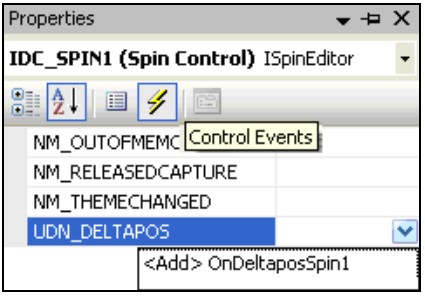


Рис. 7.22. Добавление события для изменения шага счетчика

Листинг 7.27. Изменения в файле MyDlg.h для возможности изменения шага счетчика

```
// ...
class MyDlg : public CDialog
{
```

```
// ...
public:
   afx_msg void OnDeltaposSpin1(NMHDR *pNMHDR, LRESULT *pResult);
};
```

Листинг 7.28. Изменения в файле MyDlg.cpp для возможности изменения шага счетчика

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN1,&MyDlg::OnDeltaposSpin1)
END_MESSAGE_MAP()
// ...
void MyDlg::OnDeltaposSpin1(NMHDR *pNMHDR, LRESULT *pResult)
{
   LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
   // TODO: Add your control notification handler code here
   *pResult = 0;
}
```

Макрос `ON_NOTIFY` применяется для уведомляющих сообщений. Сообщение `UDN_DELTAPOS` посылается родительскому окну при нажатии на одну из кнопок счетчика (кнопки со стрелками) и обрабатывается *до модификации счетчика*, что дает возможность проконтроливать или изменить некоторые значения (например, чтобы запретить изменения в ассоциированном окне, надо в обработке этого сообщения установить `*pResult = 1`). Функция обработки сообщения `UDN_DELTAPOS` имеет два аргумента:

```
void MyDlg::OnDeltapos(
   NMHDR *pNMHDR,       // Указатель на структуру-заголовок
   LRESULT *pResult);    // Флаг изменения состояния счетчика:
                         // 0 - разрешить изменения, 1 - запретить
```

Структура-заголовок описана следующим образом:

```
typedef struct tagNMHDR
{
```

```
   HWND hwndFrom;     // Дескриптор окна управления, пославшего сообщение
   UINT idFrom;       // Идентификатор окна управления, пославшего
                      // сообщение
   UINT code;         // Код самого сообщения
} NMHDR;
```

Структура с информацией о возможных изменениях:

```
typedef struct _NM_UPDOWN
{
   NMHDR hdr;                 // Структура-заголовок
   int iPos;                  // Текущее значение счетчика
   int iDelta;                // Возможное изменение текущего значения
} NMUPDOWN, *LPNMUPDOWN;
```

Теперь добавим изменения шага счетчика так, чтобы при нажатии на кнопку счетчика, его значение изменялось на 2. Изменения в файле приведены в листинге 7.29.

Листинг 7.29. Изменения в файле MyDlg.cpp для задания нового шага счетчика

```
// ...
void MyDlg::OnDeltaposSpin1(NMHDR *pNMHDR, LRESULT *pResult)
{
   LPNMUPDOWN pNMUpDown =  reinterpret_cast<LPNMUPDOWN>(pNMHDR);
   // TODO: Add your control notification handler code here
   // Удваиваем шаг счетчика (был 1, станет 2)
   pNMUpDown->iDelta *= 2;

   *pResult = 0;
}
```

Значение счетчика для дальнейшего использования надо брать из значения переменной `edit1`. Добавим контроль значений в текстовое поле `edit1`. Чтобы можно было вводить только цифры, надо в окне свойств для `IDC_EDIT1` в поле **Number** задать значение `True` (рис. 7.23). Теперь при попытке пользователя ввести символ, не являющийся цифрой, будет появляться сообщение (рис. 7.24). Для работы с текстовым полем надо завести переменную (категории **Control**) `wedit1` (рис. 7.25).

Также надо добавить обработку сообщения `EN_UPDATE` (рис. 7.26):

- в окне свойств для `IDC_EDIT1` выбрать вкладку **Control Events**;
- в списке событий найти `EN_UPDATE` и выбрать **<Add> OnEnUpdateEdit1**.

Сообщение `EN_UPDATE` возникает когда пользователь вводит текст в текстовое поле, *но текстовое поле еще не обновилось*. Изменения в файлах приведены в листингах 7.30 и 7.31.

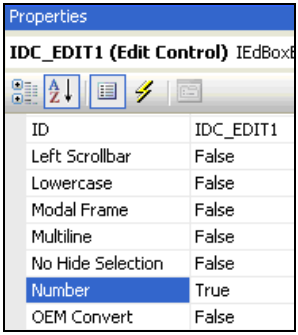


Рис. 7.23. Изменение свойств текстового поля

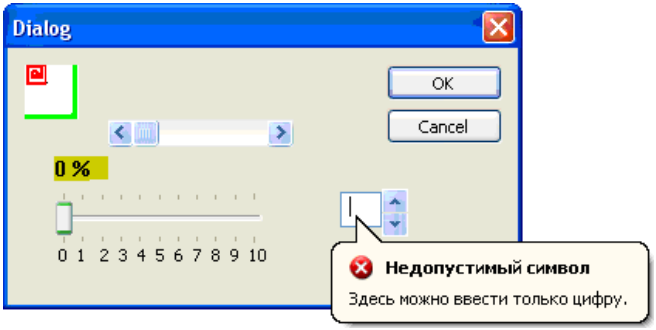


Рис. 7.24. Контроль корректности значений текстового поля

Листинг 7.30. Изменения в файле MyDlg.h для возможности контроля значений, вводимых в текстовое поле

```
// ...
class MyDlg : public CDialog
{
```

```
// ...
public:
   CEdit wedit1;
public:
   afx_msg void OnEnUpdateEdit1();
};
```

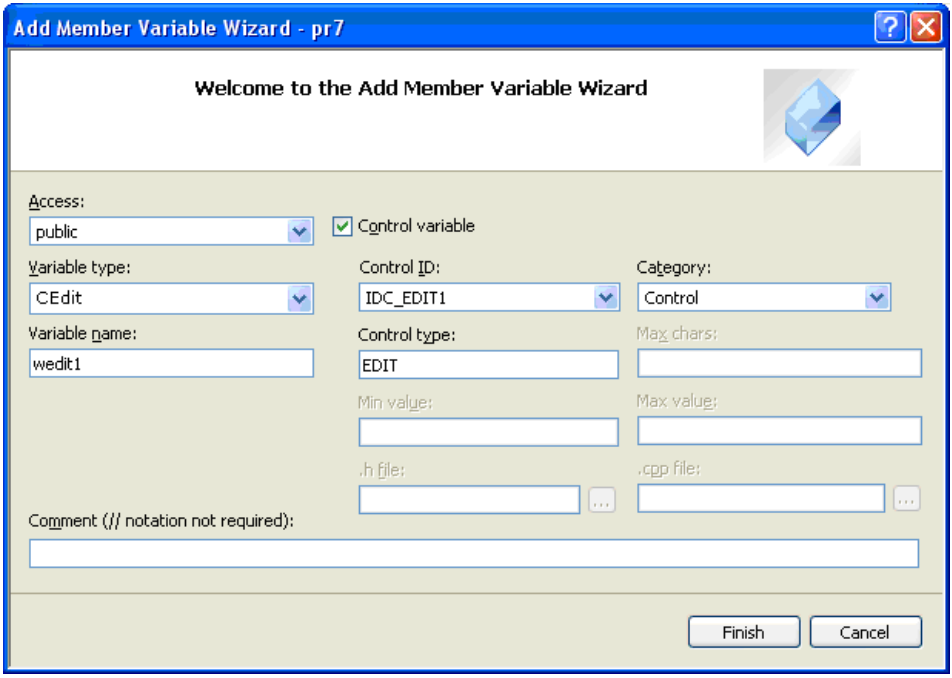


Рис. 7.25. Управляющая переменная, связанная с текстовым полем

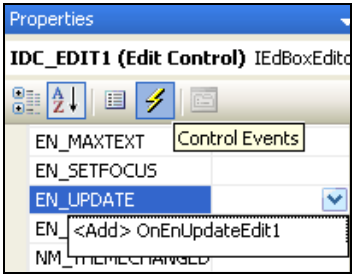


Рис. 7.26. Добавление сообщения для контроля значений текстового поля

Листинг 7.31. Изменения в файле MyDlg.cpp для возможности контроля значений, вводимых в текстовое поле

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   DDX_Text(pDX, IDC_EDIT1, edit1);
   DDX_Control(pDX, IDC_EDIT1, wedit1);
}

BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN1, &MyDlg::OnDeltaposSpin1)
   ON_EN_UPDATE(IDC_EDIT1, &MyDlg::OnEnUpdateEdit1)
END_MESSAGE_MAP()
// ...
void MyDlg::OnEnUpdateEdit1()
{
   // TODO:  If this is a RICHEDIT control, the control will not
   // send this notification unless you override the
   // CDialog::OnInitDialog()
   // function to send the EM_SETEVENTMASK message to the control
   // with the ENM_UPDATE flag ORed into the lParam mask.

   // TODO:  Add your control notification handler code here
}
```

Теперь добавим обработку так, чтобы при вводе неверных значений в текстовое поле его содержимое становилось равным нижней границе значений счетчика (нулю). Изменения в файле приведены в листинге 7.32.

Листинг 7.32. Изменения в файле MyDlg.cpp для контроля значений, вводимых в текстовое поле

```
// ...
void MyDlg::OnEnUpdateEdit1()
{
   // ...
```

```
   // TODO:  Add your control notification handler code here
   CString s;                    // Содержимое текстового поля
   const wchar_t *wch;           // Си-строка с Unicode
   char *ch;                     // Си-строка без Unicode
   int z;                        // Содержимое текстового поля в целом
                                 // формате
   size_t sizeb,                 // Размер ch в байтах
          cnv;                   // Количество преобразованный символов

   wedit1.GetWindowTextW(s);     // Получить содержимое текстового поля
   wch = s.GetBuffer();          // Получить Си-строку с Unicode
   sizeb = (s.GetLength()+1)*2;  // Вычислить размер для многобайтовой
                                 // строки
   ch = new char[sizeb];
   wcstombs_s(&cnv, ch, sizeb, wch, sizeb);   // Преобразовать wch в ch
   z = atoi(ch);                              // Преобразовать ch в целое
                                              // число
   if(z < 0 || z > 10)           // Если это число недопустимое
   {
      wedit1.SetSel(0, -1);      // Выделить весь текст в текстовом поле
      wedit1.ReplaceSel(L"0")    // Заменить его на "0"
   }
}
```

Копирование содержимого текстового поля (или названия окна) в строку выполняется с помощью функции:

```
int CWnd::GetWindowText(         // Длина скопированной строки
   LPTSTR lpszStringBuf,         // Строка для копирования
   int nMaxCount) const;         // Максимальное количество символов для
                                 // копирования (включая '\0')
```

или

```
void CWnd::GetWindowText(
   CString& rString) const;      // Строка для копирования
```

Получить содержимое буфера строки можно функцией:

```
PXSTR CSimpleStringT::GetBuffer(   // Указатель на символьный буфер
   int nMinBufferLength);          // Максимальный размер буфера
```

или

```
PXSTR CSimpleStringT::GetBuffer();
```

Класс для работы со строкой:

```
class CStringT : public CSimpleStringT
typedef CStringT CString
```

Получить длину строки можно с помощью функции:

```
int CSimpleStringT::GetLength() const throw();     // Длина строки
```

Многие функции языка C++ перешли из языка C и могут работать только со строкой многобайтовых символов (`char *`). Преобразование строки расширенных символов (в кодировке Unicode) в соответствующую строку многобайтовых символов (без Unicode) выполняется функцией:

```
errno_t wcstombs_s(           // 0 - хорошо или код ошибки
   size_t *pConvertedChars,   // Заполняется количеством преобразованных
                              // символов
   char *mbstr,               // Указатель на многобайтовую Си-строку без
                              // Unicode (результат)
   size_t sizeInBytes,        // Размер буфера для mbstr
   const wchar_t *wcstr,      // Указатель на расширенную Си-строку с
                              // Unicode (для преобразования)
   size_t count);             // Максимальное количество байт, которые
                              // могут быть сохранены в выходной строке
```

Преобразование строки в целое число выполняется с помощью функции:

```
int atoi(                     // Целое число
   const char *str);          // Строка
```

Для преобразования расширенной строки в целое число существует функция:

```
int _wtoi(                    // Целое число
   const wchar_t *str);       // Расширенная строка
```

Здесь проще было бы использовать функцию `_wtoi()` (но была использована `atoi()`, чтобы показать работу с `wcstombs_s()`, т. к. многие функции требуют именно тип `const char*`).

Функцию `_wtoi()` можно использовать, например, так:

```
CString s;
const wchar_t *wch;
int z;
wedit1.GetWindowTextW(s);
wch = s.GetBuffer();
z = _wtoi(wch);
```

Изменение выделения текста выполняется функцией:

```
void CEdit::SetSel(
   int nStartChar,           // Начальная позиция выделения
   int nEndChar,             // Конечная позиция
   BOOL bNoScroll = FALSE);  // Флаг установки текстового курсора в зону
                             // видимости: FALSE - установить, TRUE - нет
```

Если `nStartChar = 0` и `nEndChar = -1`, то будет выделен весь текст. Если `nStartChar = -1`, то выделение текста будет снято.

Определение положения текущего выделения текста выполняется с помощью функции:

```
void CEdit::GetSel(
   int& nStartChar,          // Начальная позиция выделения
   int& nEndChar) const;     // Конечная позиция
```

Замена *выделенного* текста на заданный:

```
void CEdit::ReplaceSel(
   LPCTSTR lpszNewText,      // Новый (заданный) текст
   BOOL bCanUndo = FALSE );  // Возможность последующей отмены действий
                             // данной функции TRUE - может быть
                             // отменена, FALSE -нет
```

Удаление *выделенного* текста выполняется функцией:

```
void CEdit::Clear();
```

Дальнейшая работа со значением счетчика аналогична работе с текстовым полем (до закрытия окна диалога — через переменную `wedit1`, после закрытия окна диалога — через переменную `edit1`).

7.1.5. Использование кодировки Unicode

Unicode — это расширенный 2-байтовый многоязычный символьный код, предназначенный для того, чтобы поддерживать все наборы символов (включая наборы символов, которые не могут быть представлены в одном байте). Использование кодировки Unicode задается при создании проекта (рис. 7.27).

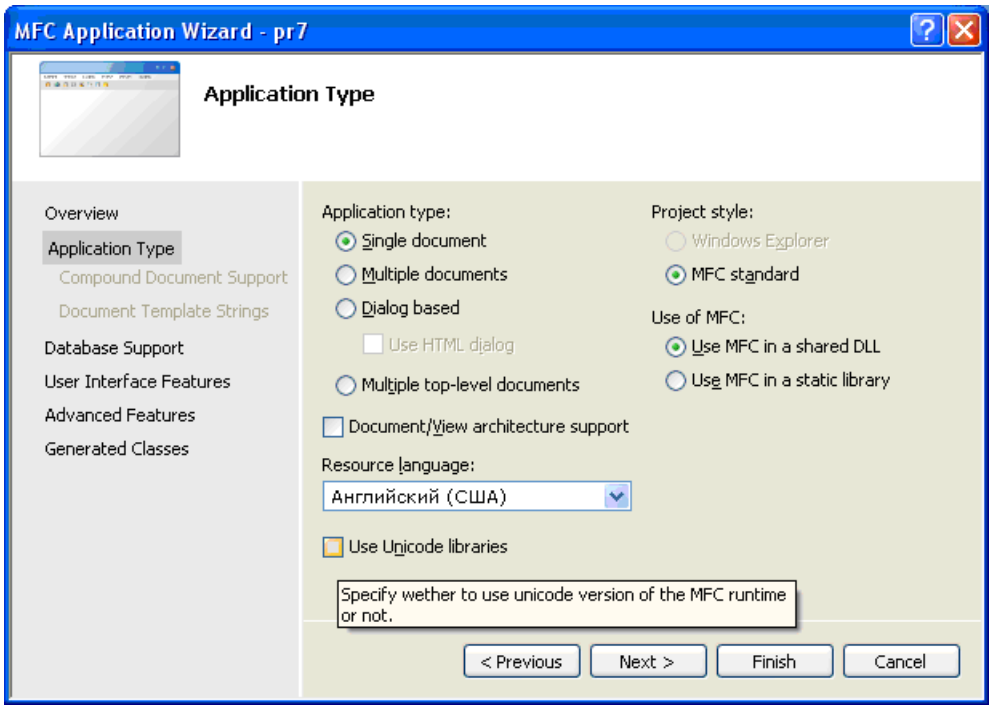
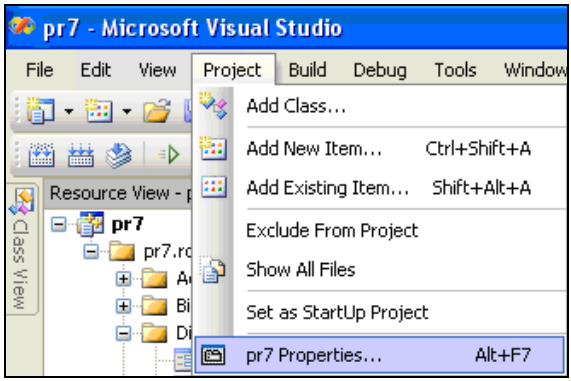


Рис. 7.27. Выбор использования Unicode

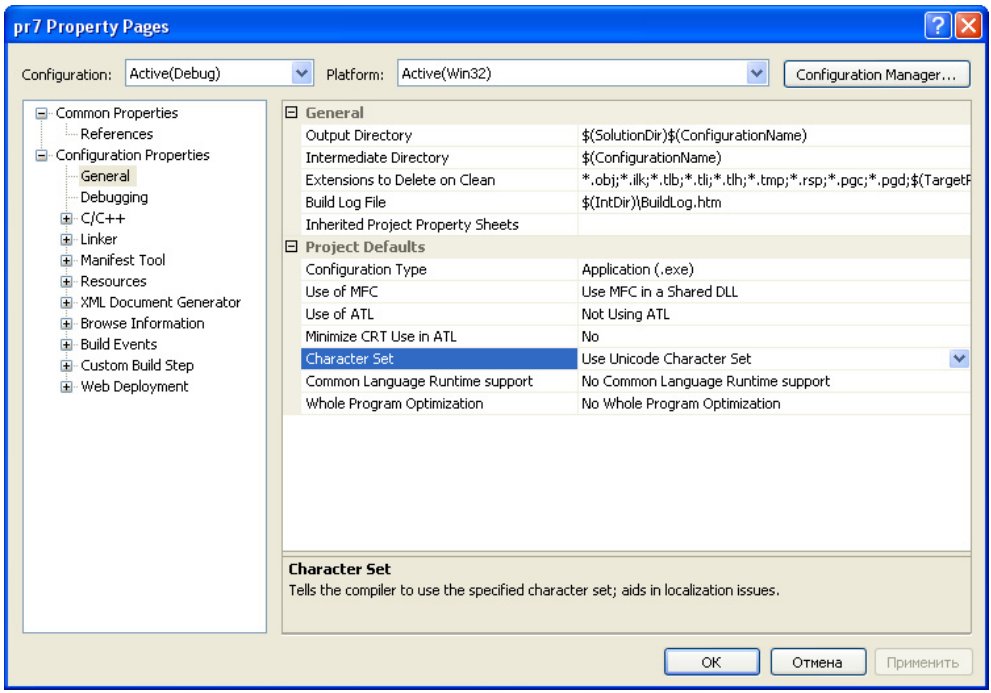
При *использовании Unicode* класс `CString` заменяется на `CStringW` (так же, как и многие функции имеют на конце `'W'`, например `funcW()`). Все строки должны быть представлены в виде: `L"строка"` или `_T("строка")`. Преобразование строки типа `CString` в Си-строку типа `char*` (или `const char*`) автоматически не происходит, и для этого надо использовать функцию `wcstombs_s()`.

Если *отказаться от использования Unicode*, то класс `CString` заменяется на `CStringA` (как и многие функции имеют на конце `'A'`, например `funcA()`).

Все строки представляются в привычном виде: `"строка"`. Преобразование строки типа `CString` в Си-строку типа `char*` (или `const char*`) происходит автоматически.



а



б

Рис. 7.28. Открытие окна свойств проекта с помощью меню (*а*) и изменение свойств проекта (*б*)

В проектах **pr1**—**pr7** было включено использование **Unicode** (чтобы показать работу с ним). В дальнейшем, начиная с *гл. 8*, для удобства работы использование **Unicode** будет отключено.

Посмотреть и изменить использование кодировки Unicode можно через свойства проекта. Для этого необходимо:

- в меню выполнить команду **Project | pr7 Properties** (рис. 7.28, *а*);
- в окне **pr7 Property** в категории **Configuration Properties** (Свойства проекта) выбрать вкладку **General** (Основные) и в поле **Character Set** (Набор символов) выбрать одно из двух значений (рис. 7.28, *б*):
  - **Use Unicode Character Set** (Использовать Unicode);
  - **Use Multi-Byte Character Set** (Не использовать Unicode);
- нажать кнопку **OK**.

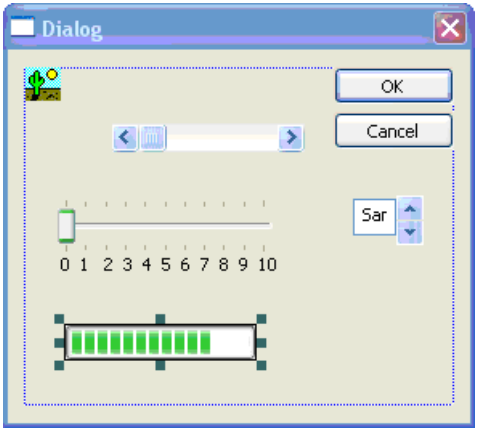
При отключении в свойствах проекта использования Unicode необходимо в тексте программы изменить все написания строк с `L"строка"` на `"строка"` и все функции с `funcW()` на `funcA()`.

Если проект предусматривает работу с текстовыми файлами (например, открытие их в окне редактирования), то использование Unicode *надо обязательно отключить*, иначе весь текст будет отображаться в виде `' '`.

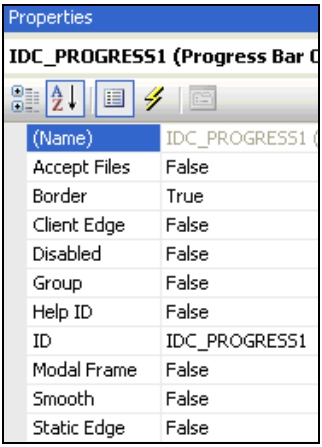
> ***Примечание***
>
> Здесь же (рис. 7.28, *б*) можно изменить способ подключения библиотек. В поле **Use of MFC** можно выбрать **Use MFC in a Shared DLL** (Динамически подключаемые библиотеки) или **Use MFC in a Static Library** (Статическое использование библиотек). О разнице в использовании библиотек говорилось в *разд. 1.1*.

7.1.6. Индикатор (*Progress Control*)

Добавим индикатор **Progress Control** (рис. 7.29) и две кнопки **Запустить** (`IDC_BUTTON1`) и **Стоп** (`IDC_BUTTON2`) для запуска и остановки индикатора (рис. 7.30). Сделаем обработку сообщений от этих кнопок (для быстрого добавления обработки от кнопки надо дважды щелкнуть по ней левой кнопкой мыши). Изменения в файлах приведены в листингах 7.33—7.35.



а



б

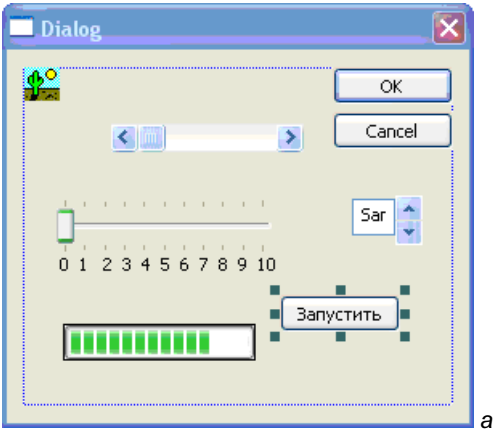
Рис. 7.29. Добавление индикатора в окно диалога (*а*) и просмотр его свойств (*б*)

Листинг 7.33. Изменения в файле Resource.h для работы с индикатором

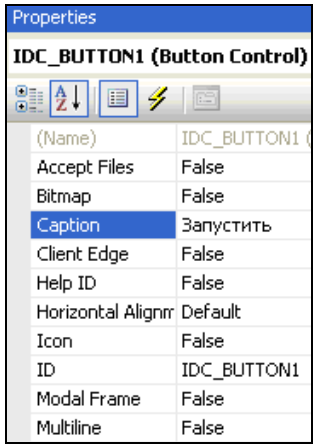
```
// ...
#define IDC_PROGRESS1                   1012
#define IDC_BUTTON1                     1013
#define IDC_BUTTON2                     1014
// ...
```

Листинг 7.34. Изменения в файле MyDlg.h для работы с индикатором

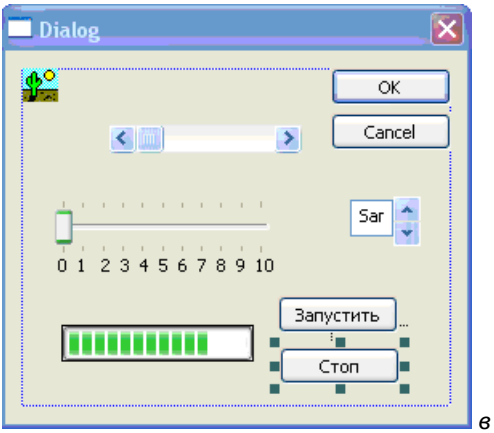
```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnBnClickedButton1();
public:
   afx_msg void OnBnClickedButton2();
};
```



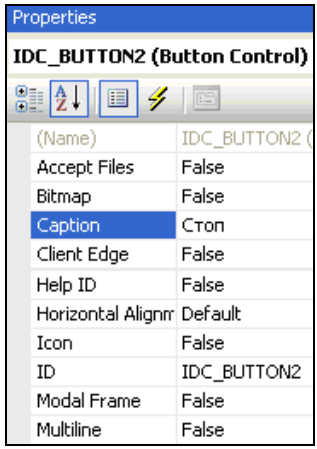
а



б



в



г

Рис. 7.30. Добавление кнопки для запуска индикатора (*а*) и просмотр ее свойств (*б*), добавление кнопки для остановки индикатора (*в*) и просмотр ее свойств (*г*)

Листинг 7.35. Изменения в файле MyDlg.cpp для работы с индикатором

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
   ON_BN_CLICKED(IDC_BUTTON2, &MyDlg::OnBnClickedButton2)
END_MESSAGE_MAP()
```

```
// ...
void MyDlg::OnBnClickedButton1()
{
    // TODO: Add your control notification handler code here
}

void MyDlg::OnBnClickedButton2()
{
    // TODO: Add your control notification handler code here
}
```

Добавим переменную `ftimer` для работы с таймером и сделаем так, чтобы при нажатии на кнопки таймер запускался и останавливался. Изменения в файлах приведены в листингах 7.36 и 7.37.

Листинг 7.36. Изменения в файле MyDlg.h для работы с таймером

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   bool ftimer;          // true - таймер запущен, false - нет
};
```

Листинг 7.37. Изменения в файле MyDlg.cpp для работы с таймером

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , scrollbar1(0), slider1(0)
   , edit1(0)
{
   ftimer = false;
}
// ...
void MyDlg::OnBnClickedButton1()
```

```
{
   // TODO: Add your control notification handler code here
   // Запустить таймер 1 с интервалом 1 сек (1000 миллисек)
   if(!ftimer)
   {
      progr.SetPos(1);
      SetTimer(1, 1000, NULL);
      ftimer = true;
   }
}

void MyDlg::OnBnClickedButton2()
{
   // TODO: Add your control notification handler code here
   // Остановить таймер 1
   if(ftimer)
   {
      KillTimer(1);
      ftimer = false;
   }
}
```

Установка таймера выполняется с помощью функции:

```
UINT_PTR CWnd::SetTimer(  // Идентификатор нового таймера или 0 – при
ошибке
   UINT_PTR nIDEvent,     // Идентификатор нового таймера (1, 2, и т.д.
                          // (таймеров может быть несколько)
   UINT nElapse,          // Временной интервал таймера в миллисекундах.
                          // Указатель на функцию, которая будет
                          // обрабатывать сообщения таймера WM_TIMER
   void (CALLBACK* lpfnTimer)(HWND, UINT, UINT_PTR, DWORD));
```

Удаление таймера осуществляется функцией:

```
BOOL CWnd::KillTimer(     // 0 – ошибка
   UINT_PTR nIDEvent);    // Идентификатор удаляемого таймера
```

Для работы с индикатором заведем переменную (категории **Control**) `progr` (рис. 7.31) и добавим установку начальных значений индикатора. Изменения в файлах приведены в листингах 7.38 и 7.39.

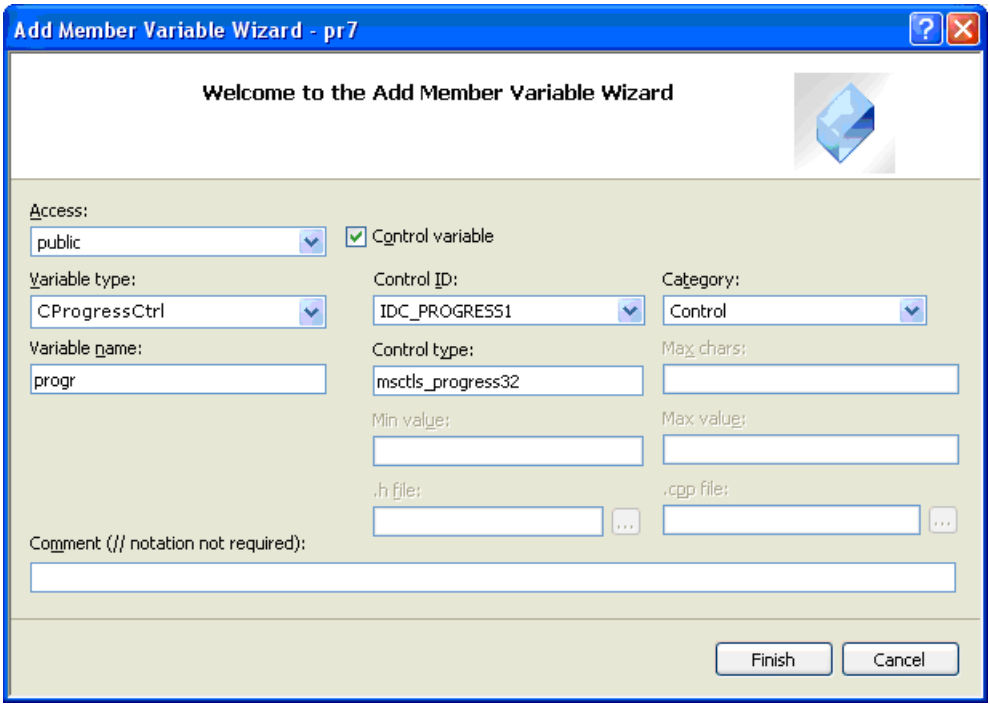


Рис. 7.31. Добавление переменной, связанной с индикатором

Листинг 7.38. Изменения в файле MyDlg.h для установки начальных значений индикатора

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CProgressCtrl progr;
};
```

Листинг 7.39. Изменения в файле MyDlg.cpp для установки начальных значений индикатора

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_PROGRESS1, progr);
}
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Индикатор CProgressCtrl
   progr.SetRange(1,10+1);
   progr.SetStep(1);
   progr.SetPos(1);
}
```

Установка диапазона значений индикатора выполняется функцией:

```
void CProgressCtrl::SetRange(
   short nLower,            // Нижняя граница (умолчание = 0)
   short nUpper);           // Верхняя граница (умолчание = 100)
```

или

```
void CProgressCtrl::SetRange32(
   int nLower,
   int nUpper);
```

Установка шага приращения индикатора:

```
int CProgressCtrl::SetStep(  // Старое значение шага
   int nStep);               // Новое (устанавливаемое) значение шага
```

При работа индикатора происхоит его постепенное заполнение прямоугольными сегментами. Значение шага равное 1 — это не добавление одного такого сегмента. Автоматически берутся длина окна индикатора, диапазон его значений и заданный шаг, и, исходя из этих данных, рассчитывается количество сегментов в одном шаге. Таким образом, при одинаковом диа-

пазоне чем длиннее размер окна индикатора, тем больше сегментов будет в одном шаге.

Установка текущей позиции индикатора выполняется с помощью функции:

```
int CProgressCtrl::SetPos(      // Старая позиция
   int nPos);                   // Новая (устанавливаемая) позиция
```

Добавим в класс диалога обработку сообщения таймера `WM_TIMER` (рис. 7.32):

- в окне свойств диалогового окна выбрать вкладку **Messages**;
- в списке сообщений найти `WM_TIMER` и выбрать **<Add> OnTimer**.

Изменения в файлах приведены в листингах 7.40 и 7.41.

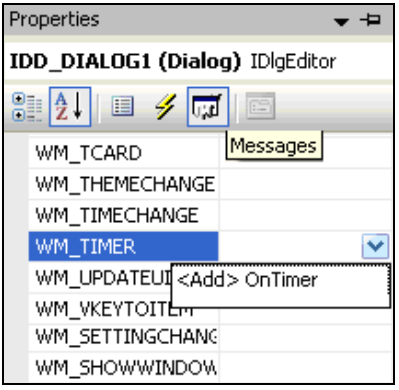


Рис. 7.32. Добавление обработки сообщения таймера

Листинг 7.40. Изменения в файле MyDlg.h для обработки сообщения таймера

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnTimer(UINT_PTR nIDEvent);
};
```

Листинг 7.41. Изменения в файле MyDlg.cpp для обработки сообщения таймера

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_WM_TIMER()
END_MESSAGE_MAP()
// ...
void MyDlg::OnTimer(UINT_PTR nIDEvent)
{
   // TODO: Add your message handler code here and/or call default

   CDialog::OnTimer(nIDEvent);
}
```

Функция обработки сообщения таймера описана следующим образом:

```
afx_msg void CWnd::OnTimer(
   UINT_PTR nIDEvent);         // Идентификатор таймера, пославшего
                               // сообщение(1, 2, и т. д.)
```

Сделаем так, чтобы при получении сообщения от таймера (оно будет генерироваться каждую секунду) значение индикатора увеличивалось. Также выведем текущее значение индикатора в окно диалога. Правильнее это делать в обработке сообщения `WM_PAINT` (аналогично обработке регулятора), т. к. при изменении размеров диалогового окна или при перекрытии его другим окном выведенный текст исчезнет. Здесь это делается прямо в обработке `WM_TIMER`, т. к. размеры окна диалога не меняются и выводимый текст обновляется каждую секунду. Изменения в файле приведены в листинге 7.42.

Листинг 7.42. Изменения в файле MyDlg.cpp для связи индикатора с таймером

```
// ...
void MyDlg::OnTimer(UINT_PTR nIDEvent)
{
   // TODO: Add your message handler code here and/or call default

   CString s;
   int z;
```

```
   if(nIDEvent == 1)              // Если это таймер 1
   {
      z = progr.GetPos();         // Получить текущую позицию индикатора
      CString s;
      s.Format(L"%d", z);
      if(z > 10)                  // Остановить таймер
      {
         OnBnClickedButton2();
         return CDialog::OnTimer(nIDEvent);
      }
      CClientDC dc(this);         // Получить контекст устройства
                                  // для вывода текста
      // "Стереть" старый текст (вывести пробелы)
     dc.TextOutW(20, 150, L"    ");
      // Вывести новый текст (текущее значение индикатора)
      dc.TextOutW(20, 150, s);
      progr.StepIt();             // Увеличить индикатор на размер шага
   }
   CDialog::OnTimer(nIDEvent);
}
```

Получить текущую позицию индикатора можно с помощью функции:

```
int CProgressCtrl::GetPos();
```

Увеличение значения индикатора на заданный ранее шаг выполняется функцией:

```
int CProgressCtrl::StepIt();      // Предыдущее значение индикатора
```

Раз уже у нас есть обработка таймера, то рядом со значением индикатора будем показывать и текущее время (это лучше делать в обработке сообщения `WM_PAINT`; но мы здесь выполним в `OnTimer()` — для упрощения кода). Если значение секунд текущего времени будет равно 0, 5, 10, 15 и т. д., таймер индикатора будет останавливаться. Изменения в файле приведены в листинге 7.43.

Листинг 7.43. Изменения в файле MyDlg.cpp для вывода текущего времени в окно диалога

```
// ...
void MyDlg::OnTimer(UINT_PTR nIDEvent)
{
   // TODO: Add your message handler code here and/or call default

   CString s;
   int z;
   CTime t;
   SYSTEMTIME st;
   if(nIDEvent == 1)                // Если это таймер 1
   {
      // ...
      progr.StepIt();               // Увеличить индикатор на размер шага

      t = CTime::GetCurrentTime();  // Получить текущее время
      s = t.Format("%H:%M:%S");     // Преобразовать его в строку вида
                                    // "чч:мм:сс"
      t.GetAsSystemTime(st);        // Заполнить структуру с данными
                                    // даты и времени
      dc.TextOutW(50, 150, L"   "); // "Стереть" старый текст
      // Вывести текущее значение времени
      dc.TextOutW(50, 150, s);
      // Если текущее значение секунд кратно 5
      if(st.wSecond % 5 == 0)
      {
         OnBnClickedButton2();      // Остановить таймер 1
      }
   }   // end if(nIDEvent == 1)
   CDialog::OnTimer(nIDEvent);
}
```

Для работы с датой и временем существует класс:

```
class CTime
```

Конструктор класса выглядит следующим образом:

```
CTime::CTime() throw();
```

или

```
CTime::CTime(
   int nYear,         // Год
   int nMonth,        // Месяц
   int nDay,          // День
   int nHour,         // Час
   int nMin,          // Минута
   int nSec,          // Секунда
   int nDST = -1);    // Флаг установки летнего времени
```

Флаг установки летнего времени может принимать значения:

- `nDST = 0` — стандартная установка времени;
- `nDST > 0` — используется летнее время;
- `nDST < 0` — время вычисляется автоматически, независимо от того, какое время используется (стандартное или летнее).

Получить текущее время можно с помощью функции:

```
static CTime WINAPI CTime ::GetCurrentTime() throw();
```

Создание строки с отформатированными значениями даты/времени выполняется функцией:

```
CString CTime::Format(             // Отформатированная строка
   LPCTSTR pszFormat) const;        // Строка формата
```

или

```
CString CTime::Format(
   UINT nFormatID) const;           // Идентификатор строки формата
```

Значения форматов времени могут быть такими:

- `%A` — день недели (*Monday*);
- `%a` — сокращенный день недели (*Mon*);
- `%B` — месяц (*March*);
- `%b` — сокращенный месяц (*Mar*);
- `%c` — дата и время в формате мм/чч/гг чч:мм:сс;
- `%#c` — дата и время в формате *Tuesday, March 14, 2007, 12:41:29*;

- `%d` — день месяца (01—31);
- `%H` — часы в формате 24 (00—23);
- `%I` — часы в формате 12 (01—12);
- `%j` — текущий день по счету в году (001—366);
- `%m` — месяц (01—12);
- `%M` — минуты (00—59);
- `%p` — значок AM/PM для значения часа (AM — время до 12 дня, PM — после);
- `%S` — секунды (00—59);
- `%U` — текущая неделя по счету в году (00—53) (воскресенье — первый день недели);
- `%w` — текущий день по счету в неделе (0—6) (воскресенье — 0);
- `%W` — какой по счету день недели в году (00—53). Например, если текущий день — понедельник и значение по `%W = 10`, то это десятый понедельник в году, от первого понедельника января;
- `%x` — дата в формате мм/чч/гг;
- `%#x` — дата в формате *Tuesday, March 14, 2007*;
- `%y` — год в формате ГГ (00—99);
- `%Y` — год в формате ГГГГ;
- `%z`, `%Z` — данные стандарта часового пояся (например, Russian Standard Time);
- `%%` — знак процента.

Преобразование формата даты и времени, хранящихся в объекте `CTime`, в формат Win32 выполняется с помощью функции:

```
bool CTime::GetAsSystemTime(      // 0 - ошибка
   SYSTEMTIME& st) const throw(); // Структура с преобразованным временем
```

Структура с данными о дате и времени определена как:

```
typedef struct _SYSTEMTIME
{
   WORD wYear;              // Год (1601 - 30827)
   WORD wMonth;             // Месяц (1-12, 1 — январь)
   WORD wDayOfWeek;         // День недели (0-6, 0 — Воскресенье)
```

```
    WORD wDay;                // Число (1-31)
    WORD wHour;               // Часы (0-23)
    WORD wMinute;             // Минуты (0-59)
    WORD wSecond;             // Секунды (0-59)
    WORD wMilliseconds;       // Миллисекунды (0-999)
} SYSTEMTIME, *PSYSTEMTIME;
```

Тип `WORD` определен как:

```
typedef unsigned short WORD;
```

Результат работы программы показан на рис. 7.33.

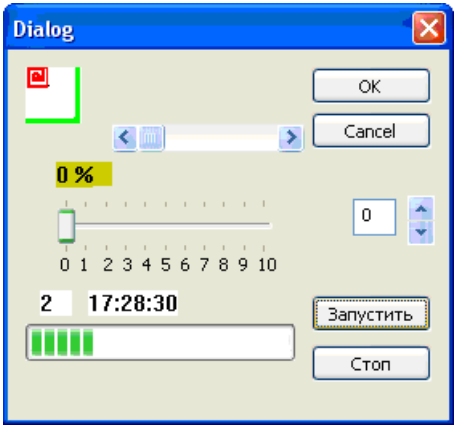


Рис. 7.33. Результат работы с индикатором и таймером

7.1.7. Быстрая клавиша (*Hot Key*)

Добавим быструю (горячую) клавишу **Hot Key** (рис. 7.34) и переменную (категории **Control**) `hotkey1` (рис. 7.35). Изменения в файлах приведены в листингах 7.44—7.46.

Листинг 7.44. Изменения в файле Resource.h для работы с быстрой клавишей

```
// ...
#define IDC_HOTKEY1                     1015
// ...
```

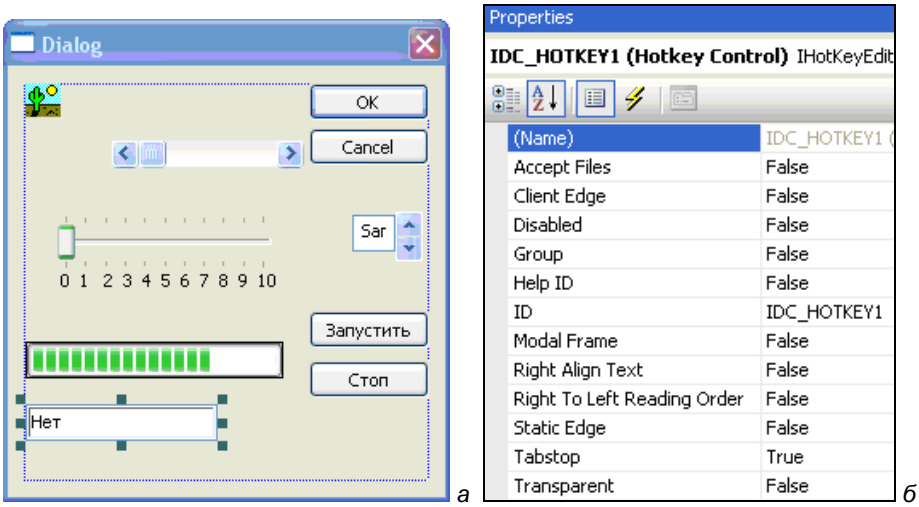


Рис. 7.34. Добавление быстрой клавиши в окно диалога (*а*) и просмотр ее свойств (*б*)

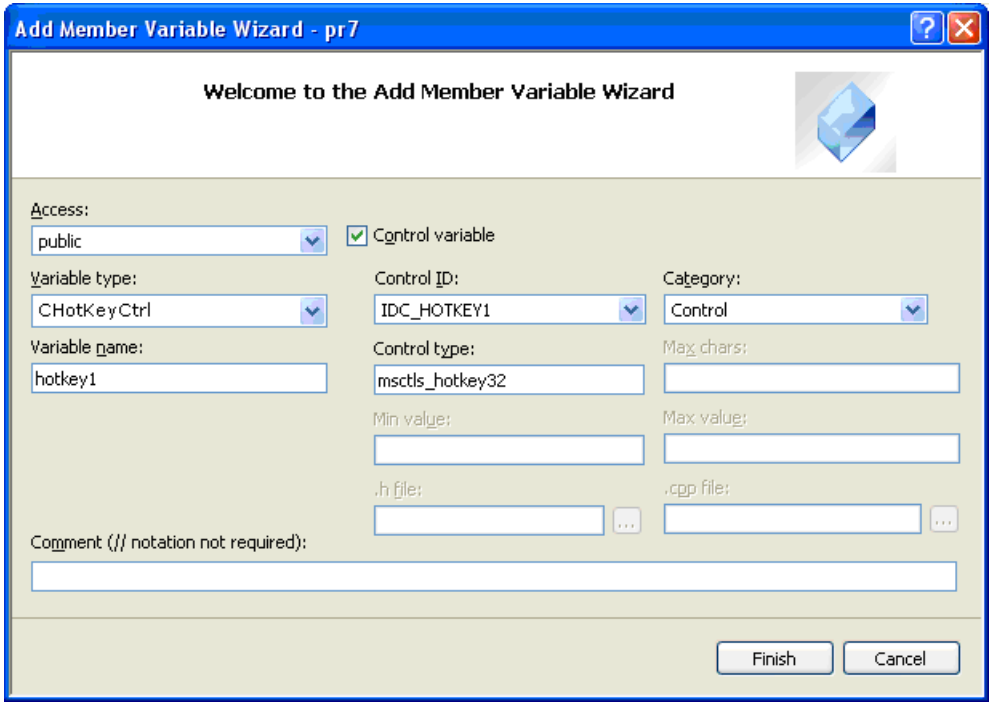


Рис. 7.35. Добавление переменной для работы с быстрой клавишей

Листинг 7.45. Изменения в файле MyDlg.h для работы с быстрой клавишей

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CHotKeyCtrl hotkey1;
};
```

Листинг 7.46. Изменения в файле MyDlg.cpp для работы с быстрой клавишей

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_HOTKEY1, hotkey1);
}
```

Заведем две статические переменные для хранения *текущих* значений горячей клавиши: `vk` (для виртуального кода клавиши) и `mod` (для кода клавиш-модификаторов). В процессе работы программы эти значения могут меняться, поэтому переменные объявлены статическими, на внешнем уровне и только один (первый раз) инициализируются значениями (комбинация горячей клавиши изначально будет <Ctrl>+<Shift>+<F1>). Так же добавим начальную инициализацию горячей клавиши. Изменения в файле приведены в листинге 7.47.

Листинг 7.47. Изменения в файле MyDlg.cpp для задания начального значения комбинации горячей клавиши

```
// ...
static WORD  vk = VK_F1,              // Виртуальный код клавиши
            // Клавиши-модификаторы
            mod = HOTKEYF_SHIFT | HOTKEYF_CONTROL;
// ...
BOOL MyDlg::OnInitDialog()
```

```
{
   // ...
   // Горячая клавиша
   hotkey1.SetHotKey(vk, mod);
}
```

Коды виртуальных клавиш были приведены в *разд. 5.1.4*.

Клавиши-модификаторы могут быть такими:

- `HOTKEYF_ALT` — клавиша <Alt>;
- `HOTKEYF_CONTROL` — клавиша <Ctrl>;
- `HOTKEYF_EXT` — была нажата клавиша перемещения курсора (стрелки вверх, вниз и т. д.);
- `HOTKEYF_SHIFT` — клавиша <Shift>.

Инициализация горячей клавиши выполняется функцией:

```
void CHotKeyCtrl::SetHotKey(
   WORD wVirtualKeyCode,        // Виртуальный код клавиши
   WORD wModifiers);            // Клавиши-модификаторы
```

Теперь при выходе из окна диалога по кнопке **OK** надо сохранить текущую выбранную комбинацию горячей клавиши и получить единый код горячей клавиши. Для регистрации корячей клавиши надо послать *фреймовому окну* сообщение `WM_SETHOTKEY`. Активизация горячей клавиши происходит следующим образом: фреймовому окну посылается сообщение `ON_WM_SYSCOMMAND` с идентификатором сообщения `SC_HOTKEY`**.** Добавим для диалогового окна обработку сообщения от кнопки **OK** (это можно сделать, дважды щелкнув по ней левой кнопкой мыши в окне ресурсов диалога). Изменения в файлах приведены в листингах 7.48 и 7.49.

Листинг 7.48. Изменения в файле MyDlg.h для сохранения значения горячей клавиши

```
// ...
class MyDlg : public CDialog
{
// ...
public:
```

```
    afx_msg void OnBnClickedOk();
};
```

Листинг 7.49. Изменения в файле MyDlg.cpp для сохранения значения горячей клавиши

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
    ON_BN_CLICKED(IDOK, &MyDlg::OnBnClickedOk)
END_MESSAGE_MAP()
// ...
void MyDlg::OnBnClickedOk()
{
    // TODO: Add your control notification handler code here

    // Получить и сохранить (в vk и mod)текущие значения горячей клавиши
    hotkey1.GetHotKey(vk, mod);
    // Получить полный единый код горячей клавиши
    DWORD allkey = hotkey1.GetHotKey();
    // Послать сообщение для регистрации горячей клавиши и ее код
    AfxGetMainWnd()->SendMessage(WM_SETHOTKEY, allkey);

    OnOK();
}
```

Получение информации о горячей клавише выполняется с помощью функции:

```
void CHotKeyCtrl::GetHotKey(
    WORD &wVirtualKeyCode,          // Виртуальный код клавиши
    WORD &wModifiers) const;        // Клавиши-модификаторы
```

Получение единого кода горячей клавиши выполняется функцией:

```
DWORD CHotKeyCtrl::GetHotKey() const;
```

Эта функция возвращает виртуальный код клавиши и модификатор. Виртуальный код находится в младшем байте младшего слова, а модификатор — в старшем байте младшего слова.

Пример использования:

```
CHotKeyCtrl hot;
DWORD all = hot.GetHotKey();
hot.SetHotKey(LOBYTE(LOWORD(all)), HIBYTE(LOWORD(all)));
```

Теперь в окно фрейма надо добавить обработку сообщения `WM_SYSCOMMAND` (рис. 7.36):

- в окне **Class View** вызвать контекстное меню для класса `CMainFrame` и выполнить команду **Properties**;
- в окне свойств **Properties** выбрать вкладку **Messages**, найти в списке сообщений `WM_SYSCOMMAND` и выбрать **<Add> OnSysCommand**.

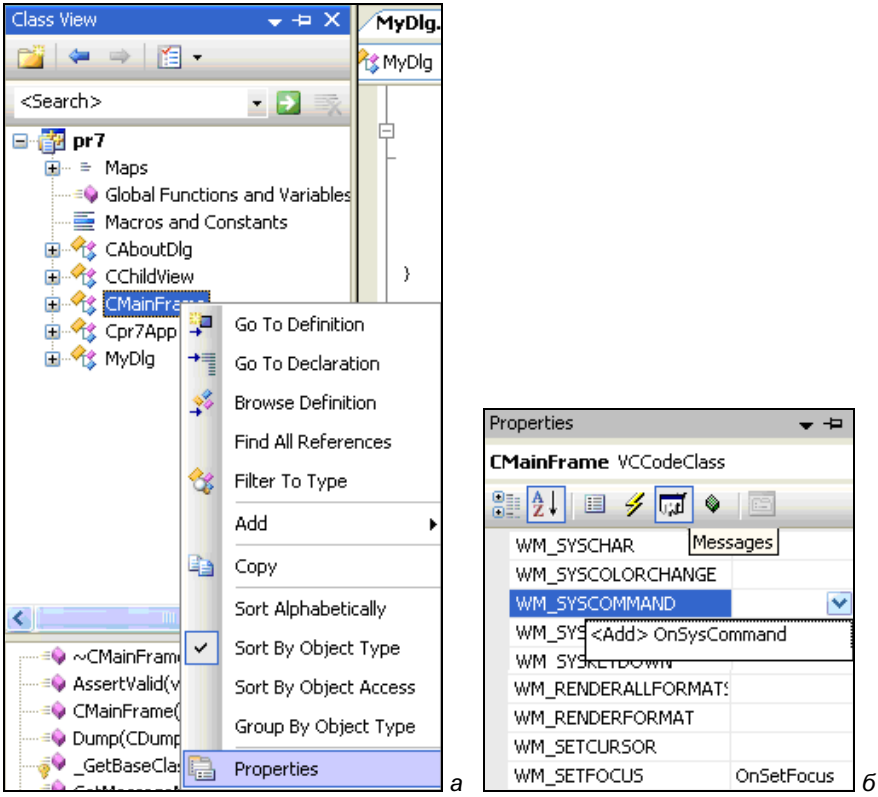


Рис. 7.36. Открытие свойств класса окна фрейма с помощью контекстного меню (*а*) и добавление обработки системных команд (*б*)

Сделаем так, чтобы при нажатии горячих клавиш выдавалось сообщение. Изменения в файлах приведены в листингах 7.50 и 7.51.

Листинг 7.50. Изменения в файле MainFrm.h для обработки сообщения горячей клавиши

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
};
```

Листинг 7.51. Изменения в файле MainFrm.cpp для обработки сообщения горячей клавиши

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_SYSCOMMAND()
END_MESSAGE_MAP()
// ...
void CMainFrame::OnSysCommand(UINT nID, LPARAM lParam)
{
   // TODO: Add your message handler code here and/or call default

   if (nID == SC_HOTKEY)          // Если была нажата горячая клавиша
   {
      AfxMessageBox(L"Сработала горячая клавиша");
      return;
   }

   CFrameWnd::OnSysCommand(nID, lParam);
}
```

Результат работы программы показан на рис. 7.37. Изначальная комбинация горячей клавиши была <Ctrl>+<Shift>+<F1>.

Чтобы изменить эту комбинацию надо:

- щелкнуть левой кнопкой мышки по полю горячей клавиши (чтобы передать фокус этому элементу управления);
- нажать на клавиатуре комбинацию клавиш <Ctrl>+<W> (в окне автоматически появится новая комбинация);
- выйти из диалога по кнопке **OK**;
- закрыть два окна сообщения (от линейки прокрутки и регулятора);
- нажать комбинацию <Ctrl>+<W> — появится окно сообщения "Сработала горячая клавиша".

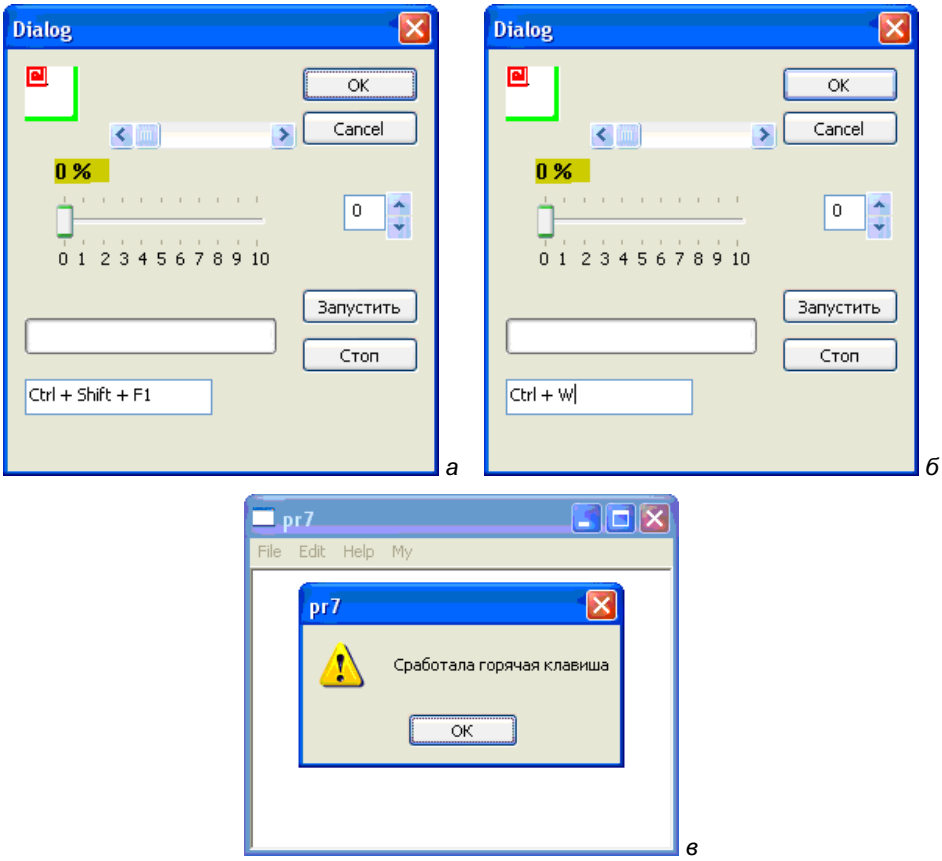
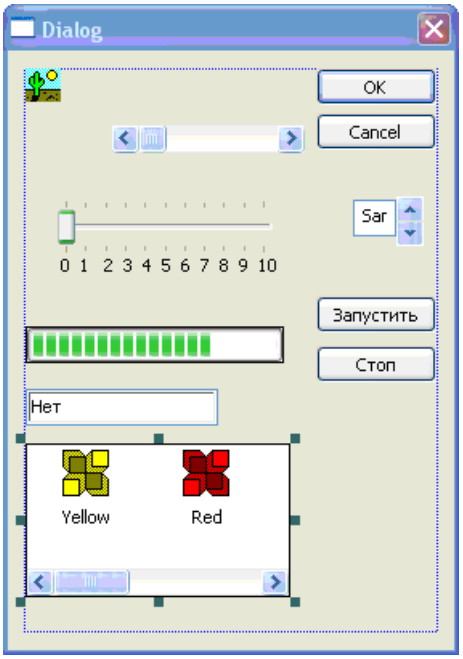


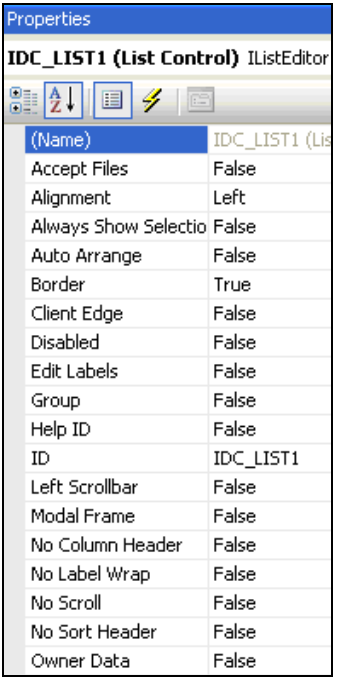
Рис. 7.37. Начальное (*а*) и измененное (*б*) значение комбинации горячей клавиши и обработка ее нажатия (*в*)

7.1.8. Список (*List Control*)

Добавим список **List Control** (рис. 7.38). Для работы со списком заведем переменную (категории **Control**) `list1` (рис. 7.39). Изменения в файлах приведены в листингах 7.52—7.54.



а



б

Рис. 7.38. Добавление списка в окно диалога (*а*) и просмотр его свойств (*б*)

Листинг 7.52. Изменения в файле Resource.h для работы со списком

```
// ...
#define IDC_LIST1               1016
// ...
```

Листинг 7.53. Изменения в файле MyDlg.h для работы со списком

```
// ...
class MyDlg : public CDialog
```

```
{
// ...
public:
   CListCtrl list1;
};
```

Листинг 7.54. Изменения в файле MyDlg.cpp для работы со списком

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   DDX_Control(pDX, IDC_LIST1, list1);
}
```

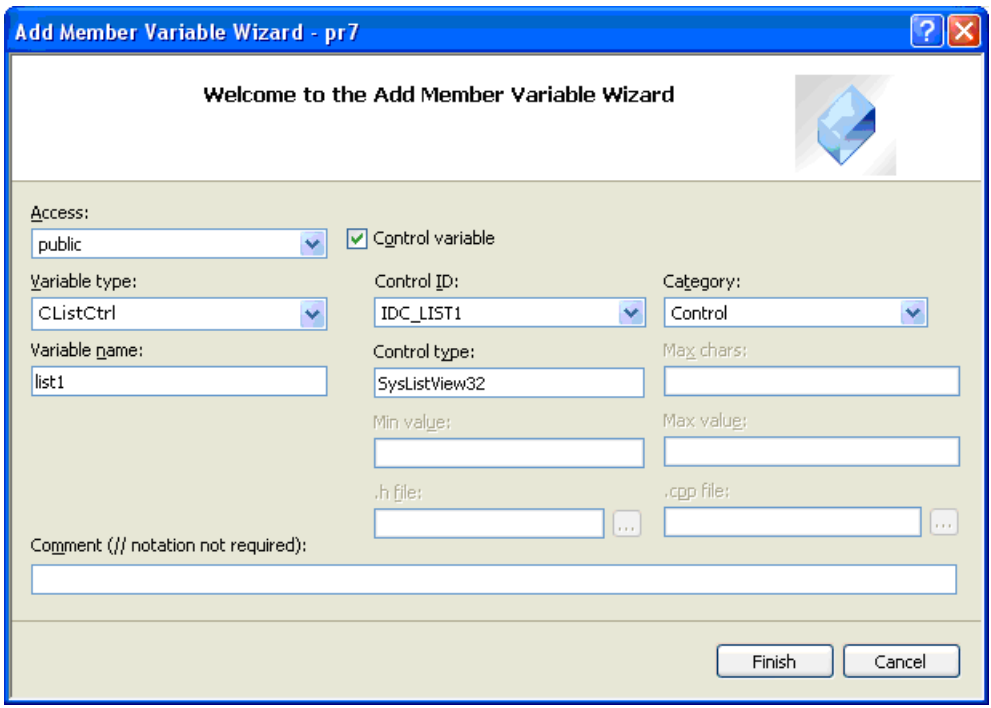


Рис. 7.39. Добавление переменной для работы со списком

Список может иметь четыре состояния (начальное состояние списка задается в окне свойств списка, в поле **View** (рис. 7.40)):

- **Icon** — большие значки (рис. 7.41, *а*);
- **Small Icon** — маленькие значки (рис. 7.41, *б*);
- **List** — список (рис. 7.41, *в*);
- **Report** — таблица (рис 7.41, *г*).

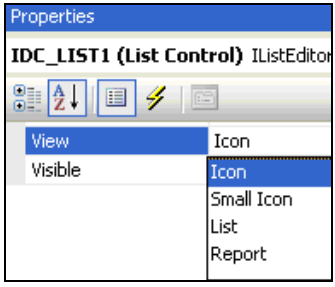


Рис. 7.40. Возможные виды списка

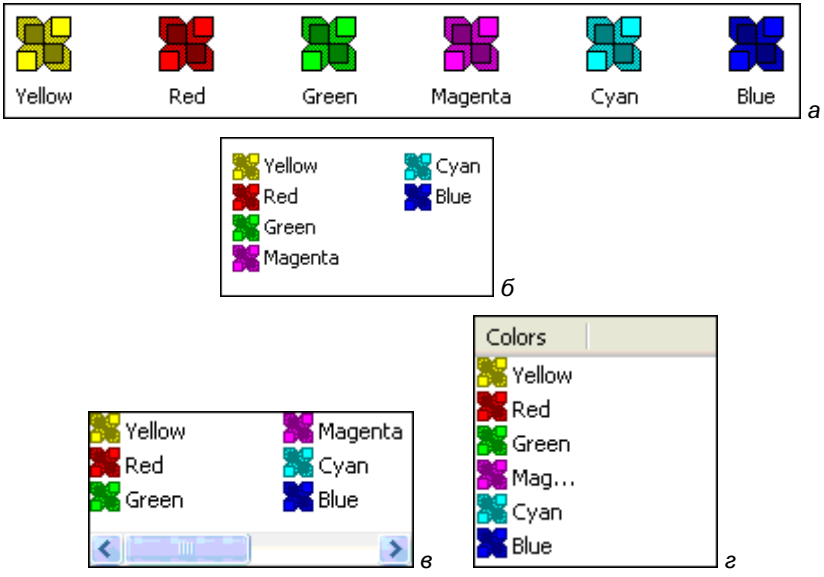


Рис. 7.41. Список в виде больших значков (**Icon**) (*а*), в виде маленьких значков (**Small Icon**) (*б*), в виде **List** (*в*) и в виде таблицы (**Report**) (*г*)

Для работы с большими и малыми значками (иконками) нам понадобятся два списка иконок. Изменения в файле приведены в листинге 7.55.

Листинг 7.55. Изменения в файле MyDlg.h для работы с иконками

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CListCtrl list1;
   CImageList i_large;              // Список больших икон
   CImageList i_small;              // Список маленьких икон
};
```

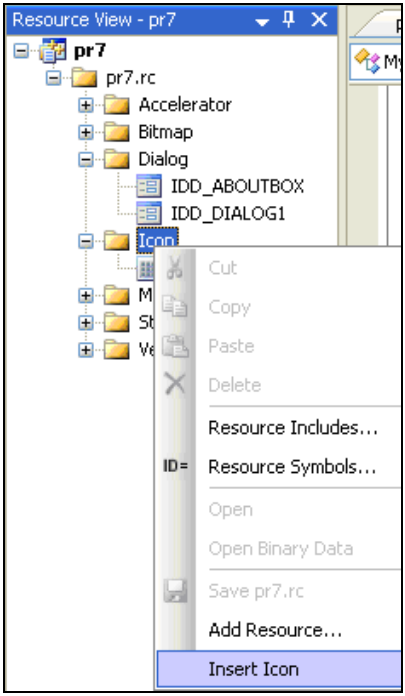


Рис. 7.42. Добавление иконки в ресурсы с помощью контекстного меню

Для работы с изображениями существует класс:

```
class CImageList : public CObject
```

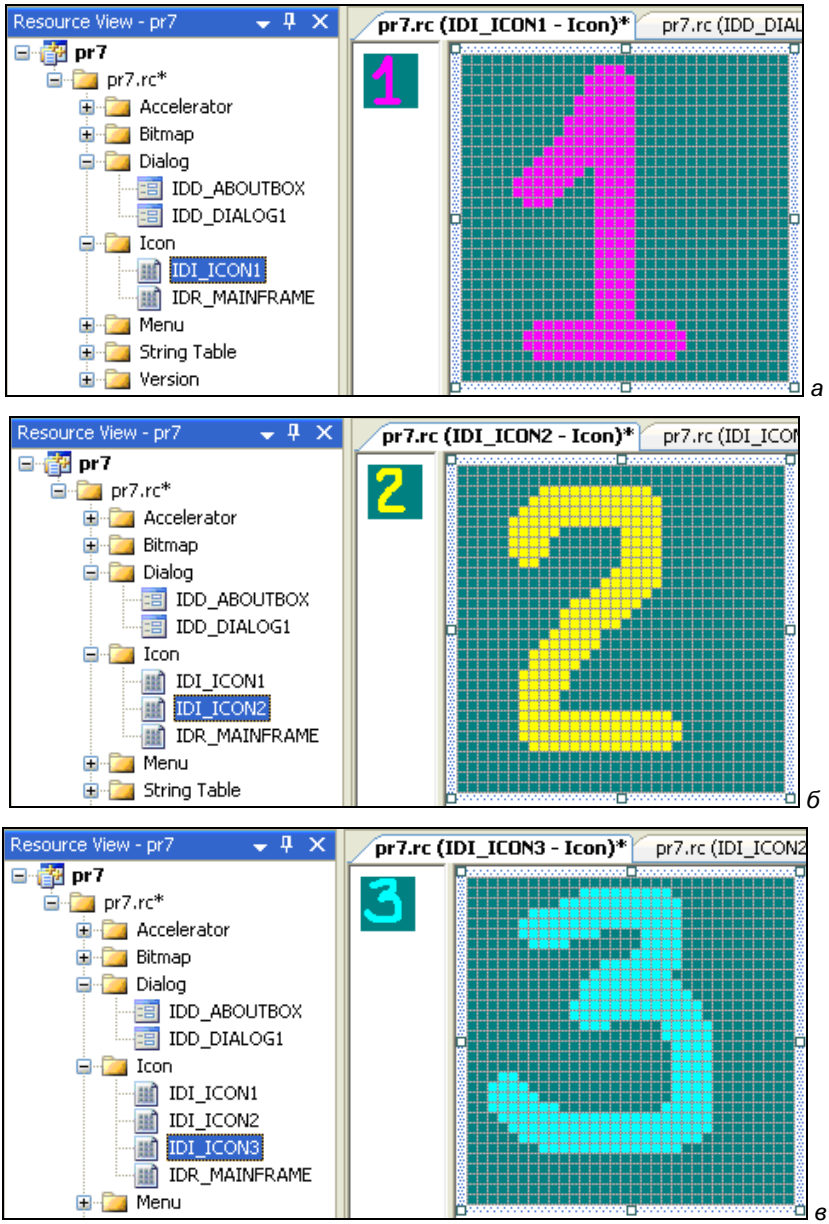


Рис. 7.43. Три иконки для списка

Пусть в списке будут три элемента. В ресурсах надо создать три иконки (32×32 или 16×16). По умолчанию предлагается 32×32. Для создания иконки надо в окне **Resource View** вызвать контекстное меню для ресурса **Icon** и выполнить команду **Insert Icon** (Вставить иконку) (рис. 7.42).

Создадим три иконки размером 32×32 (например, как показано на рис. 7.43).

Изменения в файле приведены в листинге 7.56.

Листинг 7.56. Изменения в файле Resource.h при добавлении новых иконок

```
// ...
#define IDI_ICON1                   132
#define IDI_ICON2                   133
#define IDI_ICON3                   134
// ...
```

Теперь заполним списки иконок, подключим их к списку и заполним названия элементов списка. Изменения в файле приведены в листинге 7.57.

Листинг 7.57. Изменения в файле MyDlg.cpp для начальной инициализации списка

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Список с иконками
   // Создание списков маленьких и больших иконок
   i_small.Create(16,16, NULL, 0, 1);
   i_large.Create(32, 32, NULL, 0, 1);
   // Добавление иконок в список
   i_small.Add(AfxGetApp()->LoadIconW(IDI_ICON1));
   i_small.Add(AfxGetApp()->LoadIconW(IDI_ICON2));
   i_small.Add(AfxGetApp()->LoadIconW(IDI_ICON3));
   i_large.Add(AfxGetApp()->LoadIconW(IDI_ICON1));
   i_large.Add(AfxGetApp()->LoadIconW(IDI_ICON2));
   i_large.Add(AfxGetApp()->LoadIconW(IDI_ICON3));
```

```
   // Подключение списков иконок к реальному рабочему списку
   list1.SetImageList(&i_small, LVSIL_SMALL);
   list1.SetImageList(&i_large, LVSIL_NORMAL);
   // Заполнение самого списка (с привязкой к иконкам)
   list1.InsertItem(0, L"elem1",0);
   list1.InsertItem(1, L"elem2",1);
   list1.InsertItem(2, L"elem3",2);
}
```

Создание списка изображений выполняется функцией:

```
BOOL CImageList::Create(    // 0 - ошибка
   int cx,                  // Ширина изображения
   int cy,                  // Высота изображения
   UINT nFlags,             // Тип создаваемого списка
                            // (использование маски)
   int nInitial,            // Начальное количество изображений
   int nGrow);              // Размер выделения памяти
```

Типы масок списка изображений (`nFlags`) могут быть такими:

- `ILC_COLOR` — встроенная палитра цветов (можно `NULL`);
- `ILC_COLOR4` — 4 бита (16 цветов);
- `ILC_COLOR8` — 8 бит;
- `ILC_COLOR16` — 16 бит;
- `ILC_COLOR24` — 24 бита;
- `ILC_COLOR32` — 32 бита;
- `ILC_COLORDDB` — изображение зависит от типа устройства;
- `ILC_MASK` — изображение содержит два битовых изображения, одно из которых является монохромным (используется как маска).

При задании ненулевого значения параметра `nInitial` сразу при создании списка создается битовый массив, который в дальнейшем будет содержать изображения. Параметр `nGrow` позволяет более эффективно выделять память для хранения изображений (он задает, на сколько изображений надо увеличивать битовый массив при необходимости).

Добавление изображения в список выполняется с помощью функции:

```
int CImageList::Add(          // Индекс в списке занесенного элемента
                              // или -1 при ошибке
   HICON hIcon);              // Дескриптор иконки с битовым изображением
```

Получить указатель на объект приложения можно функцией:

```
CWinApp* AFXAPI AfxGetApp();
```

Загрузить иконку из ресурсов приложения:

```
HICON CWinApp::LoadIcon(       // Дескриптор иконы или 0 при ошибке
   UINT nIDResource) const;    // Идентификатор загружаемой иконки
```

Установить текущий список изображений:

```
CImageList* CListCtrl::SetImageList(  // Указатель на прежний список
   CImageList* pImageList,            // Указатель на подключаемый список
   int nImageListType);               // Тип списка
```

Типы списков изображений (`nImageListType`) могут быть следующими:

- `LVSIL_NORMAL` — список с большими иконками;
- `LVSIL_SMALL` — список с маленькими иконками;
- `LVSIL_STATE` — список с изображениями состояний.

Добавить элемент в список можно функцией:

```
int CListCtrl::InsertItem(    // Индекс в списке занесенного элемента
                              // или -1 при ошибке
   int nItem,                 // Индекс заносимого (нового элемента)
   LPCTSTR lpszItem,          // Строка с текстом для элемента
   int nImage);               // Индекс значка в списке изображений
```

или

```
int CListCtrl::InsertItem(
   const LVITEM* pItem);      // Указатель на структуру с атрибутами
                              // списка LVITEM (будет описана дальше)
```

или

```
int CListCtrl::InsertItem(
   int nItem,
   LPCTSTR lpszItem);
```

Чтобы можно было изменять вид списка, добавим на окно диалога четыре кнопки с заголовками (в поле **Caption**) **Large** (`IDC_BUTTON3`), **Small** (`IDC_BUTTON4`), **List** (`IDC_BUTTON5`), **Report** (`IDC_BUTTON6`) (рис. 7.44).

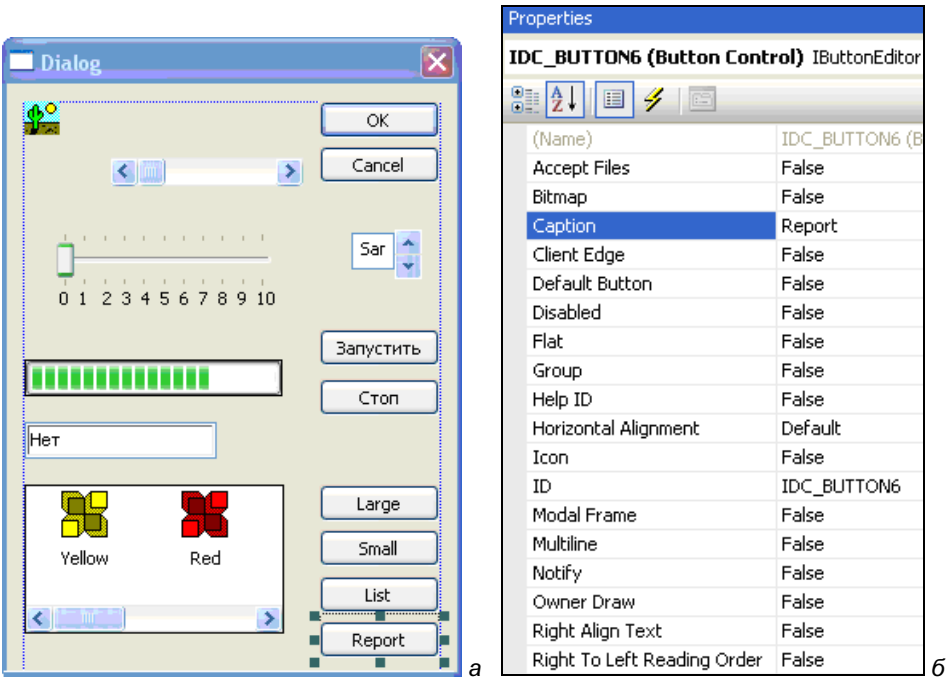


Рис. 7.44. Кнопки для выбора вида списка (*а*) и окно свойств кнопки (*б*)

Теперь (когда в окне диалога много кнопок), чтобы не путаться в функциях для их обработок, будем добавлять функцию обработки не с помощью двойного щелчка левой кнопки мыши (как раньше), а с помощью меню:

- ❒ для нужной кнопки вызвать контекстное меню и выполнить команду **Add Event Handler** (рис. 7.45);
- ❒ в окне **Event Handler Wizard** задать следующие значения (рис. 7.46):
  - в списке **Message type** выбрать значение **BN_CLICKED**;
  - в списке **Class list** выбрать значение **MyDlg**;
  - в поле **Function handler name** ввести **OnBnClickedButton3_Large**;
- ❒ нажать кнопку **Add and Edit**.

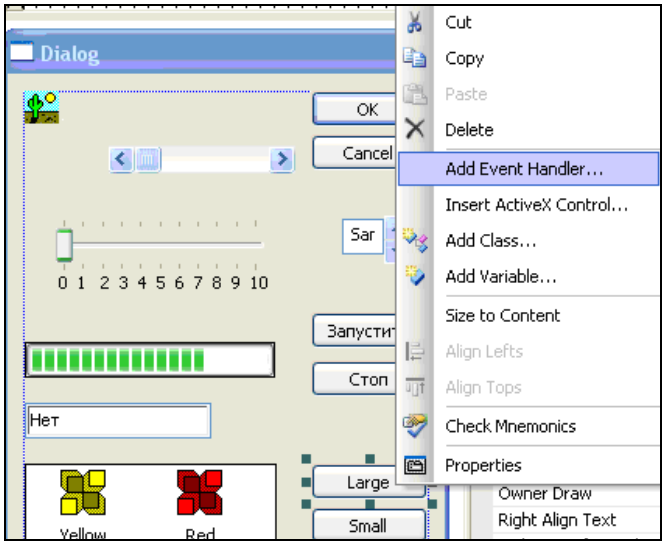


Рис. 7.45. Вызов контекстного меню
для добавления обработки сообщения для кнопки **Large**

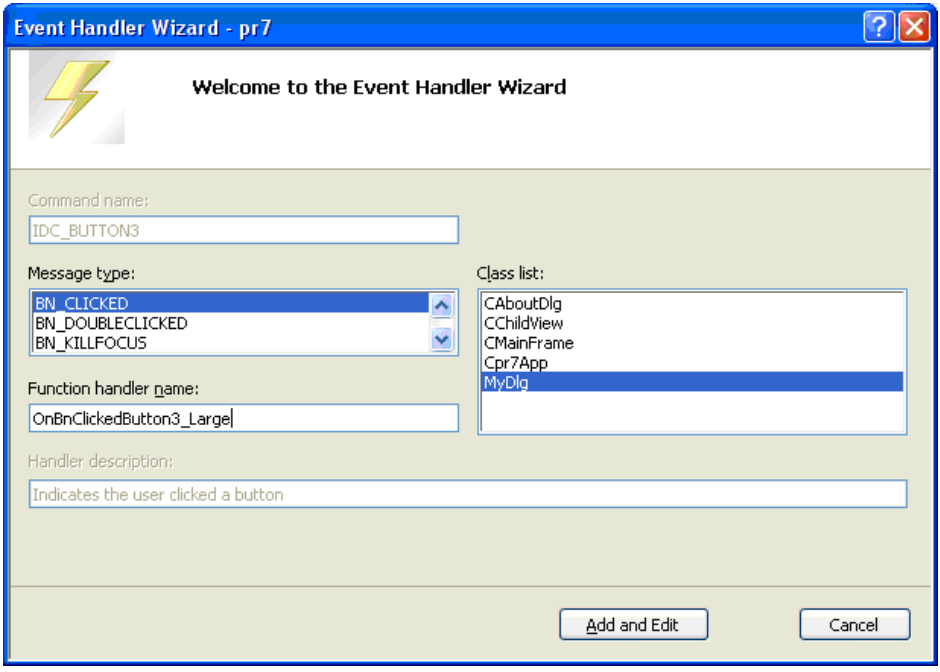


Рис. 7.46. Добавление обработки сообщения о нажатии кнопки **Large** в окне диалога

Аналогичным образом добавим функции `OnBnClickedButton4_Small()`, `OnBnClickedButton5_List()`, `OnBnClickedButton6_Report()`. Изменения в файлах приведены в листингах 7.58—7.60.

Листинг 7.58. Изменения в файле Resource.h для обработки выбора кнопок вида списка

```
// ...
#define IDC_BUTTON3                     1017
#define IDC_BUTTON4                     1018
#define IDC_BUTTON5                     1019
#define IDC_BUTTON6                     1020
// ...
```

Листинг 7.59. Изменения в файле MyDlg.h для обработки выбора кнопок вида списка

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnBnClickedButton3_Large();
public:
   afx_msg void OnBnClickedButton4_Small();
public:
   afx_msg void OnBnClickedButton5_List();
public:
   afx_msg void OnBnClickedButton6_Report();
};
```

Листинг 7.60. Изменения в файле MyDlg.cpp для обработки выбора кнопок вида списка

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
```

```
   ON_BN_CLICKED(IDC_BUTTON3, &MyDlg::OnBnClickedButton3_Large)
   ON_BN_CLICKED(IDC_BUTTON4, &MyDlg::OnBnClickedButton4_Small)
   ON_BN_CLICKED(IDC_BUTTON5, &MyDlg::OnBnClickedButton5_List)
   ON_BN_CLICKED(IDC_BUTTON6, &MyDlg::OnBnClickedButton6_Report)
END_MESSAGE_MAP()
// ...
void MyDlg::OnBnClickedButton3_Large()
{
   // TODO: Add your control notification handler code here
   // Получить текущий стиль
   DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
   // Задать новый стиль на основе старого, изменив только вид списка
   SetWindowLong(list1.m_hWnd, GWL_STYLE,
                 (style&~LVS_TYPEMASK) | LVS_ICON);
}
void MyDlg::OnBnClickedButton4_Small()
{
   // TODO: Add your control notification handler code here
   // Получить текущий стиль
   DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
   // Задать новый стиль на основе старого, изменив только вид списка
   SetWindowLong(list1.m_hWnd, GWL_STYLE,
                 (style&~LVS_TYPEMASK) | LVS_SMALLICON);
}
void MyDlg::OnBnClickedButton5_List()
{
   // TODO: Add your control notification handler code here
   // Получить текущий стиль
   DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
   // Задать новый стиль на основе старого, изменив только вид списка
   SetWindowLong(list1.m_hWnd, GWL_STYLE,
                 (style&~LVS_TYPEMASK) | LVS_LIST);
}
void MyDlg::OnBnClickedButton6_Report()
{
```

```
    // TODO: Add your control notification handler code here
    // Получить текущий стиль
    DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
    // Задать новый стиль на основе старого, изменив только вид списка
    SetWindowLong(list1.m_hWnd, GWL_STYLE,
                  (style&~LVS_TYPEMASK) | LVS_REPORT);
}
```

Получение атрибутов определенного окна (здесь — списка) выполняется функцией:

```
LONG GetWindowLong(  // Значение в виде 32-битного целого или 0 - при
                     // ошибке
   HWND hWnd,        // Дескриптор окна
   int nIndex);      // Определение типа атрибута (см. SetWindowLong())
```

Изменения атрибутов определенного окна (здесь окна списка):

```
LONG SetWindowLong(  // Предыдущее значение в виде 32-битного целого или
                     // 0 - при ошибкке
   HWND hWnd,        // Дескриптор окна для изменения
   int nIndex,       // Определение типа изменяемого атрибута
   LONG dwNewLong);  // Устанавливаемое (новое) значение атрибута
```

Типы изменяемых атрибутов окна (`nIndex`) могут быть следующие:

- `GWL_EXSTYLE` — устанавливает новый расширенный стиль окна;
- `GWL_STYLE` — устанавливает новый стиль окна;
- `GWL_WNDPROC` — устанавливает новый адрес для процедуры окна;
- `GWL_HINSTANCE` — устанавливает новый дескриптор приложения;
- `GWL_USERDATA` — устанавливает данные пользователя, связанные окном. Эти данные предназначены для использования приложением, которое создавало окно. Это значение инициализировано нулем.

Следующие типы доступны, когда параметр `hWnd` идентифицирует диалоговое окно:

- `DWL_DLGPROC` — устанавливает новый адрес диалоговой процедуры;
- `DWL_MSGRESULT` — устанавливает возвращаемую величину сообщения обработанного в диалоговой процедуре;

- `DWL_USER` — устанавливает новую дополнительную информацию, которая используется в приложении, как, например, дескрипторы или указатели.

Стили для списка (`dwNewLong`) могут быть такими:

- `LVS_ALIGNLEFT` — записи выравниваются по левому краю (в режиме больших и малых иконок);
- `LVS_ALIGNMASK` — определяет наиболее подходящее выравнивание для каждого режима;
- `LVS_ALIGNTOP` — записи выравниваются по верхнему краю (в режиме больших и малых иконок);
- `LVS_AUTOARRANGE` — записи автоматически выравниваются при изменении списка (в режиме больших и малых иконок);
- `LVS_EDITLABELS` — позволяет редактировать текстовые записи прямо в списке просмотра. Для работы этого режима надо обрабатывать сообщение `LVN_ENDLABELEDIT`;
- `LVS_ICON` — режим больших иконок;
- `LVS_LIST` — режим списка;
- `LVS_NOCOLUMNHEADER` — заголовки столбцов не выводятся (в режиме таблицы);
- `LVS_NOLABELWRAP` — текст в надписях не может переноситься на другую строчку (в режиме больших и малых иконок);
- `LVS_NOSCROLL` — отключает прокрутку списка в окне (все записи должны помещаться в клиентскую область окна);
- `LVS_NOSORTHEADER` — отключает стиль имитации кнопок для заголовков таблицы;
- `LVS_OWNERDATA` — определяет виртуальный список;
- `LVS_REPORT` — режим таблицы;
- `LVS_SHAREIMAGELISTS` — список использует список изображений совместно с другими элементами управления;
- `LVS_SHOWSELALWAYS` — позволяет всегда показывать выбранный элемент списка (даже когда список потерял фокус);
- `LVS_SINGLESEL` — позволяет выбрать только один элемент в списке;
- `LVS_SMALLICON` — режим маленьких иконок;

Рис. 7.47. Вид списка в режимах **Large** (*а*), **Small** (*б*), **List** (*в*) и **Report** (*г*)

- `LVS_SORTASCENDING` — сортирует записи по возрастанию;
- `LVS_SORTDESCENDING` — сортирует записи по убыванию;
- `LVS_TYPEMASK` — определяет управляющий текущий стиль окна.

Результаты работы программы показаны на рис. 7.47. Все виды изображений получились, кроме таблицы (режим **Report**). Чтобы таблица заработала, ее надо подготовить. Подготовим таблицу из трех колонок (в первой колонке будет отображаться сам элемент (как в режиме **List**), а во второй и третьей будет вспомогательная информация). Изменения в файле приведены в листинге 7.61.

Листинг 7.61. Изменения в файле MyDlg.cpp для подготовки списка в виде таблицы

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Добавление трех колонок для таблицы
   list1.InsertColumn(0, L"Название", LVCFMT_LEFT, 70);
   list1.InsertColumn(1, L"Инф1", LVCFMT_LEFT, 70);
   list1.InsertColumn(2, L"Инф2", LVCFMT_LEFT, 70);
   // Добавление в список дополнительной информации
   // по элементам для таблицы
   list1.SetItemText(0, 1, L"elem1-inf1");
   list1.SetItemText(0, 2, L"elem1-inf2");
   list1.SetItemText(1, 1, L"elem2-inf1");
   list1.SetItemText(1, 2, L"elem2-inf2");
   list1.SetItemText(2, 1, L"elem3-inf1");
   list1.SetItemText(2, 2, L"elem3-inf2");

   return TRUE;
}
```

Вставка новой колонки (для таблицы) выполняется функцией:

```
int CListCtrl::InsertColumn(    // Индекс нового столбца
   int nCol,                    // Индекс добавляемого (нового) столбца
```

```
    LPCTSTR lpszColumnHeading,    // Заголовок нового столбца
    int nFormat = LVCFMT_LEFT,    // Режим выравнивания столбцов
    int nWidth = -1,              // Ширина столбца в пикселах
    int nSubItem = -1);           // Индекс элемента, связанного со столбцом
```

Вставка значений дополнительных полей для таблицы выполняется с помощью функции:

```
BOOL CListCtrl::SetItemText(    // 0 - ошибка
    int nItem,                  // Индекс элемента в списке
    int nSubItem,               // Индекс элемента с дополнительной
                                // информацией для nItem
    LPCTSTR lpszText);          // Текст
```

Результат работы программы показан на рис. 7.48. Добавим обработку списка так, чтобы при выходе из окна диалога по кнопке **OK** анализировался выбранный элемент списка. Изменения в файле приведены в листинге 7.62.

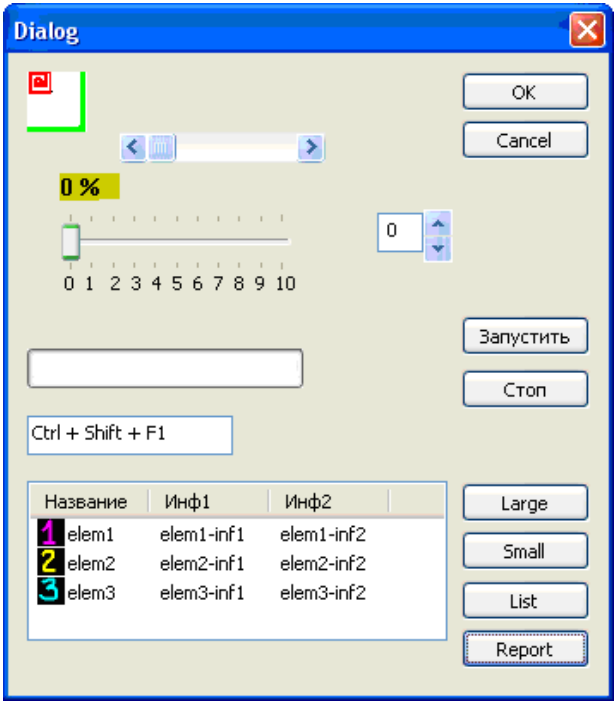


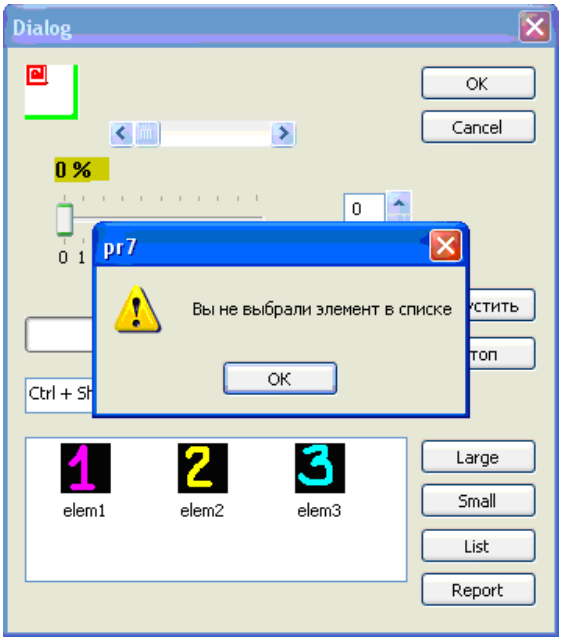
Рис. 7.48. Список в виде заполненной таблицы

Листинг 7.62. Изменения в файле MyDlg.cpp для обработки выбранного элемента списка

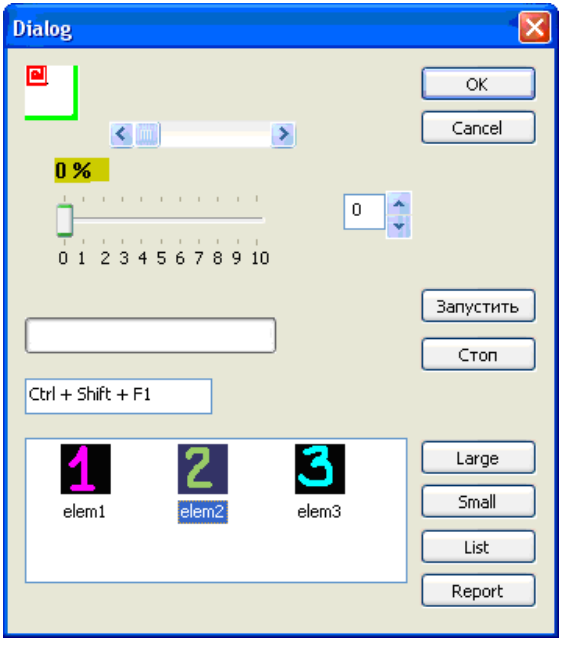
```
//...
void MyDlg::OnBnClickedOk()
{
   // ...
   // Работа со списком
   // Получить индекс выбранного элемента списка
   int item = list1.GetSelectionMark();
   if(item != -1)          // Если был выбран элемент
   {
      // Получить всю текстовую информацию выбранного элемента
      CString s;
      s += L"Название элемента: ";
      s += list1.GetItemText(item, 0);
      s += L"\nИнф1: ";
      s += list1.GetItemText(item, 1);
      s += L"\nИнф2: ";
      s += list1.GetItemText(item, 2);
      AfxMessageBox(s);
   }
   else                        // Если элемент не выбран
   {
      AfxMessageBox(L"Вы не выбрали элемент в списке");
      return;             // Не выходить из окна диалога
   }

   OnOK();
}
```

Получить индекс первого выделенного элемента (и при множественном выборе) можно с помощью функции:

```
int CListCtrl::GetSelectionMark();          // Индекс элемента или -1,
                                            // если элемент не выбран
```

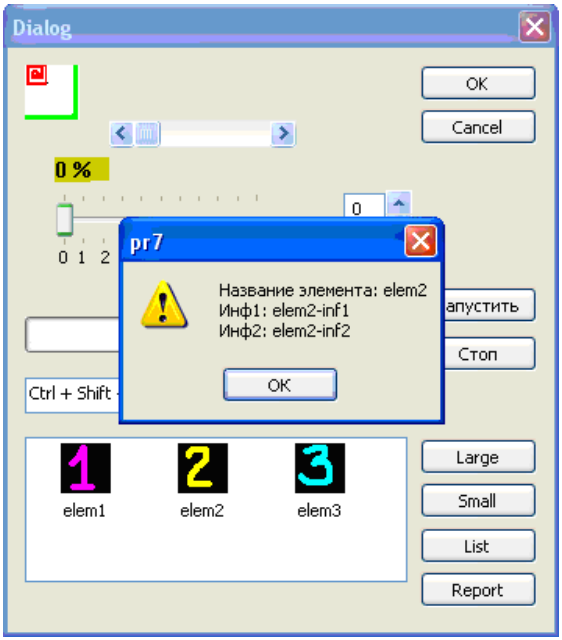


а



б

Рис. 7.49. Контроль выбора элемента списка (*а*), обработка выбранного элемента списка (*б*)



в

Рис. 7.49. Обработка выбранного элемента при закрытии диалога по кнопке **OK** (*в*)

Результат работы программы показан на рис. 7.49. Добавим код, чтобы при переходе к другим элементам управления или к другому виду списка (переключение между большими и малыми иконками и т. д.) выделенный элемент списка не терял фокус. Сделаем также, чтобы изначально всегда был выбран первый элемент списка. Изменения в файле приведены в листинге 7.63.

Листинг 7.63. Изменения в файле MyDlg.cpp для сохранения фокуса выбранного элемента

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Выделенный элемент всегда остается
   SetWindowLong (list1.m_hWnd, GWL_STYLE,
   GetWindowLong(list1.m_hWnd,GWL_STYLE) | LVS_SHOWSELALWAYS);
   // Выделить первый элемент списка
```

```
   list1.SetItem(0, 0, LVIF_STATE, NULL, 0, LVIS_SELECTED,
                 LVIS_SELECTED, 0);
   list1.SetSelectionMark(0);

   return TRUE;
}
```

Установка атрибутов вида списка выполняется с помощью функции:

```
BOOL CListCtrl::SetItem(     // 0 - ошибка
   const LVITEM* pItem);     // Указатель на структуру LVITEM
```

или

```
BOOL CListCtrl::SetItem(     // 0 - ошибка
   int nItem,                // Индекс элемента, чьи атрибуты должны быть
                             // установлены
   int nSubItem,             // Индекс дополнительного поля (столбца)
                             // элемента, чьи атрибуты должны быть
                             // установлены
   UINT nMask,               // Какие атрибуты должны быть установлены
   LPCTSTR lpszItem,         // Указатель на строку элемента
   int nImage,               // Индекс значка в списке изображений
   UINT nState,              // Значения атрибутов, которые надо
                             // установить
   UINT nStateMask,          // Устанавливаемое значение
   LPARAM lParam);           // 32-битное значение, ассоциируемое с записью
```

или

```
BOOL CListCtrl::SetItem(     // 0 - ошибка
   int nItem,
   int nSubItem,
   UINT nMask,
   LPCTSTR lpszItem,
   int nImage,
   UINT nState,
   UINT nStateMask,
   LPARAM lParam,
   int nIndent);             // Ширина (в пикселях) отступа элемента
```

Структура с атрибутами списка определена следующим образом:

```
typedef struct _LVITEM
{
   UINT mask;         // Какие из оставшихся полей должны быть заполнены
                      // (можно комбинировать)
   int iItem;         // Индекс элемента списка
   int iSubItem;      // Индекс дополнительного поля элемента списка
   UINT state;        // Текущее состояние атрибутов
   UINT stateMask;    // Новое (задаваемое) состояние атрибутов
   LPTSTR pszText;    // Текст (указатель на строку) элемента
   int cchTextMax;    // Размер буфера, на который указывает pszText
   int iImage;        // Индекс значка в списке изображений
   LPARAM lParam;    // 32-битное значение, ассоциируемое с записью
#if (_WIN32_IE >= 0x0300)
   int iIndent;      // Ширина (в пикселах) отступа элемента
#endif
#if (_WIN32_IE >= 0x560)     // Только для версии Visual C++ 6.0
   int iGroupId;             // Номер группы
   UINT cColumns;            // Номер колонки
   PUINT puColumns;          // Указатель на массив колонок длиной
                             // cColumns
#endif
} LVITEM, *LPLVITEM;
```

Значения полей структуры списка (`mask`) могут быть такими:

- `LVIF_COLUMNS` — элемент `cColumns` должен быть заполнен;
- `LVIF_DI_SETITEM` — список затребованной информации об элементах хранится в операционной системе;
- `LVIF_GROUPID` — элемент `iGroupId` должен быть заполнен;
- `LVIF_IMAGE` — элемент `iImage` должен быть заполнен;
- `LVIF_INDENT` — элемент `iIndent` должен быть заполнен;
- `LVIF_NORECOMPUTE` — для получения текстовой информации элемент управления не генерирует сообщение `LVN_GETDISPINFO`;
- `LVIF_PARAM` — поле `lParam` содержит данные;

- LVIF_STATE — поле state содержит данные;
- LVIF_TEXT — поле pszText содержит данные.

Значения атрибутов структуры списка state и stateMask возможны такие (могут комбинироваться):

- LVIS_CUT — запись выделена для операции вырезания и вставки (cut-and-paste);
- LVIS_DROPHILITED — запись выделена для переноса (drag-and-drop);
- LVIS_FOCUSED — запись имеет фокус ввода. Хотя может быть выделено несколько записей одновременно, только одна может иметь фокус ввода;
- LVIS_SELECTED — запись выбрана.

Установить индекс выделенного элемента можно с помощью функции:

```
int CListCtrl::SetSelectionMark(    // Индекс предыдущего выделенного
                                    // элемента или -1 — если его нет
   int iIndex);                     // Индекс нового выделенного элемента
```

В процессе работы *основные* элементы списка можно редактировать. Для добавления этой возможности надо изменить у списка в окне свойств в поле **Edit Labels** (Редактирование названий) значение с False на True (рис. 7.50) и добавить обработку сообщения об окончании редактирования названия элемента списка LVN_ENDLABELEDIT (рис. 7.51):

- в окне свойств списка выбрать вкладку **Control Events**;
- в списке событий найти LVN_ENDLABELEDIT и выбрать **<Add> OnLvnEndlabeleditList1**.

Изменения в файлах приведены в листингах 7.64 и 7.65.

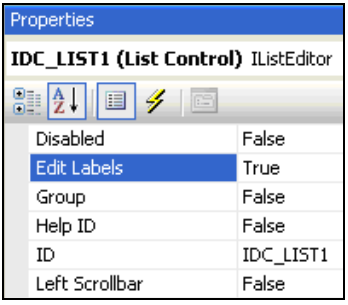


Рис. 7.50. Свойство, позволяющее редактировать название элемента списка

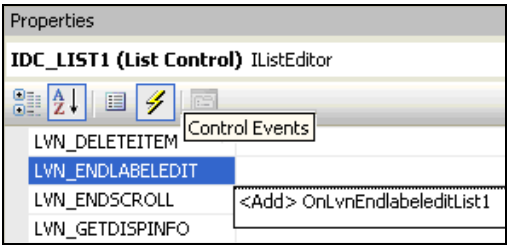


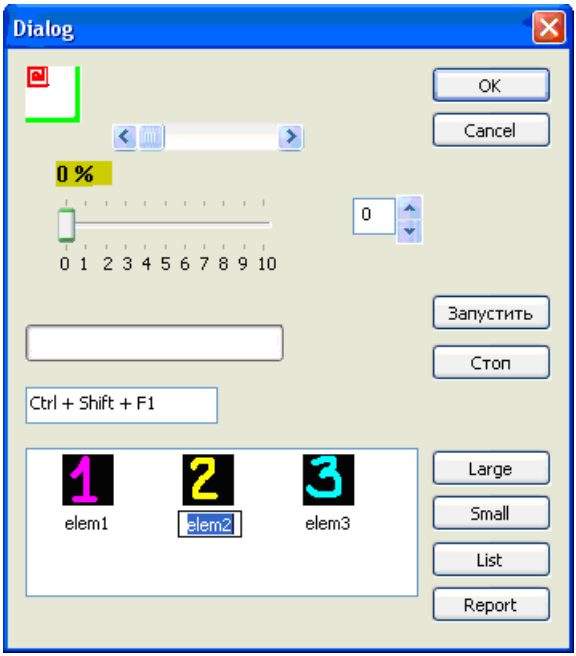
Рис. 7.51. Добавление сообщения
об изменении названия элемента списка

Листинг 7.64. Изменения в файле MyDlg.h для возможности редактирования элементов списка в процессе работы программы

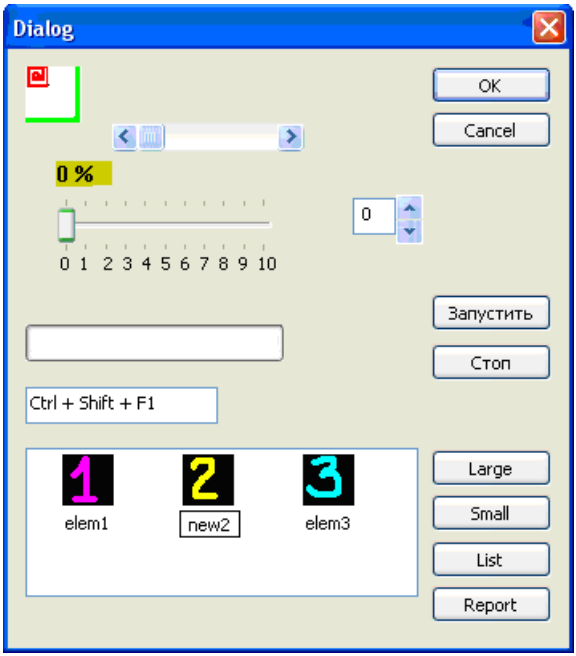
```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnLvnEndlabeleditList1(NMHDR *pNMHDR, LRESULT *pResult);
};
```

Листинг 7.65. Изменения в файле MyDlg.cpp для возможности редактирования элементов списка в процессе работы программы

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_NOTIFY(LVN_ENDLABELEDIT, IDC_LIST1, &MyDlg::OnLvnEndlabeleditList1)
END_MESSAGE_MAP()
// ...
void MyDlg::OnLvnEndlabeleditList1(NMHDR *pNMHDR, LRESULT *pResult)
{
   NMLVDISPINFO *pDispInfo =    reinterpret_cast<NMLVDISPINFO*>(pNMHDR);
   // TODO: Add your control notification handler code here

   *pResult = 0;
}
```

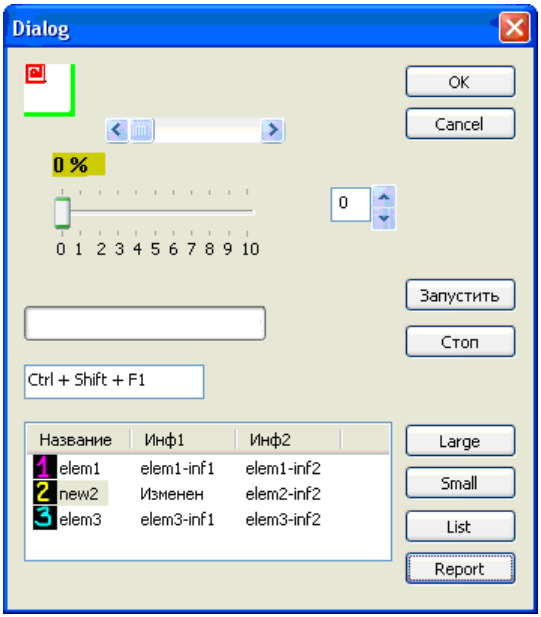


а



б

Рис. 7.52. Изменение названия элемента списка **elem2** (*а*) на **new2** (*б*)



в

Рис. 7.52. Отображение в таблице информации об измененном элементе списка (*в*)

Структура с данными о изменении состояния списка определена как:

```
typedef struct tagNMLVDISPINFO
{
   NMHDR hdr;                    // см. разд. 7.1.4
   LVITEM item;                  // см. SetItem()
} NMLVDISPINFO, *LPNMLVDISPINFO;
```

Добавим обработку кода при изменении пользователем текста элемента списка. Изменения в файле приведены в листинге 7.66.

Листинг 7.66. Изменения в файле MyDlg.cpp для отображения информации об изменении элемента списка

```
// ...
void MyDlg::OnLvnEndlabeleditList1(NMHDR *pNMHDR, LRESULT *pResult)
{
   NMLVDISPINFO *pDispInfo = reinterpret_cast<NMLVDISPINFO*>(pNMHDR);
   // TODO: Add your control notification handler code here
   if(pDispInfo->item.pszText != NULL)
```

```
    {
        list1.SetItemText(pDispInfo->item.iItem, 0,
                          pDispInfo->item.pszText);
        list1.SetItemText(pDispInfo->item.iItem, 1, L"Изменен");
    }
    *pResult = 0;
}
```

Результат работы программы показан на рис. 7.52. У элемента 2 списка изменили название с **elem2** на **new2** и нажали клавишу <Enter>. Далее нажали в окне диалога кнопку **Report** для отображения списка в виде таблицы. В столбце **Инф1** у второго элемента появилась надпись "Изменен".

7.1.9. Дерево (*Tree Control*)

Добавим дерево **Tree Control** (рис. 7.53) и переменную `tree1` (категории **Control**) для работы с ним (рис. 7.54). Изменения в файлах приведены в листингах 7.67—7.69.

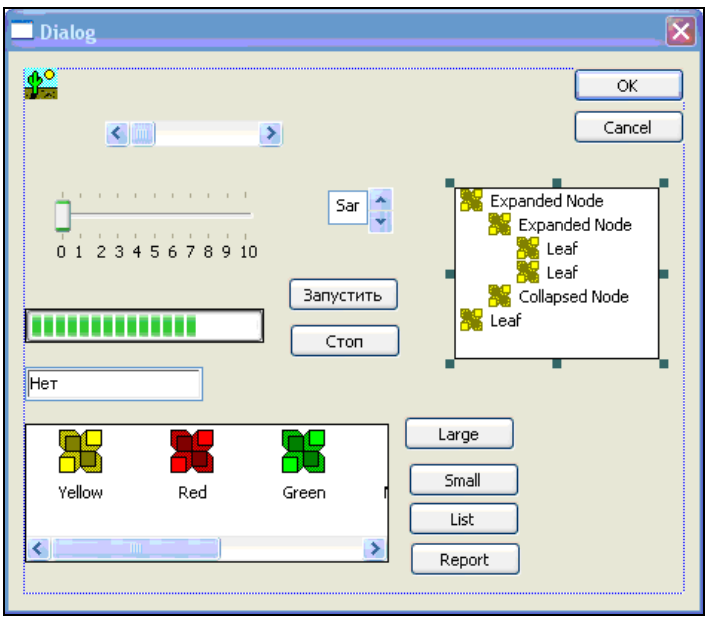


Рис. 7.53. Добавление дерева в окно диалога

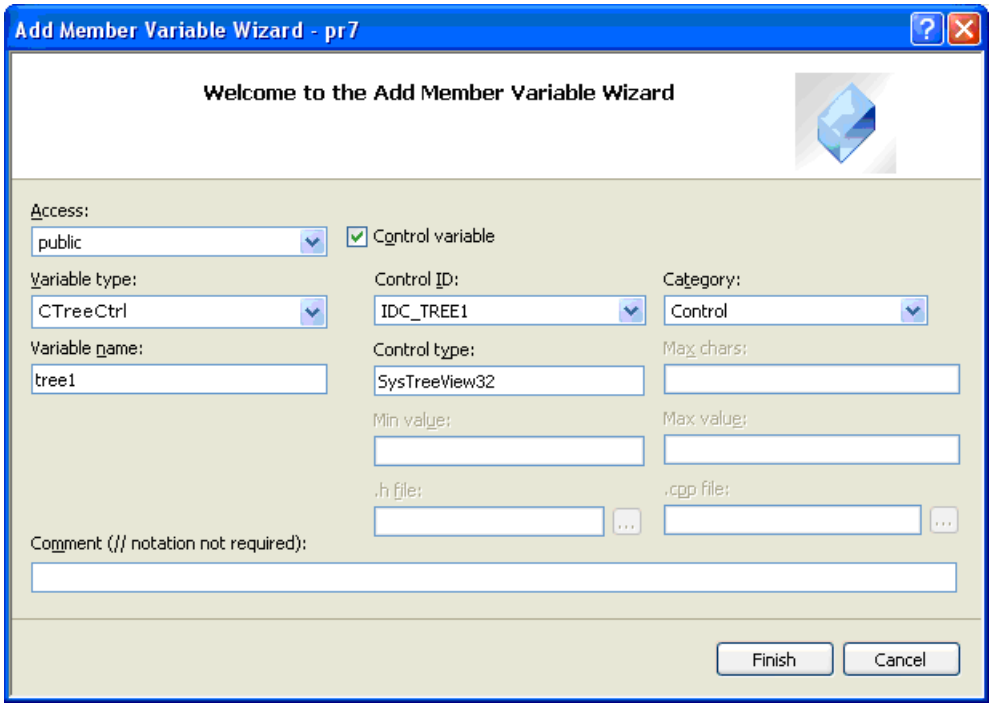


Рис. 7.54. Добавление переменной для работы с деревом

Листинг 7.67. Изменения в файле Resource.h для работы с деревом

```
// ...
#define IDC_TREE1                       1021
// ...
```

Листинг 7.68. Изменения в файле MyDlg.h для работы с деревом

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CTreeCtrl tree1;
};
```

Листинг 7.69. Изменения в файле MyDlg.cpp для работы с деревом

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_TREE1, tree1);
}
```

Работа с деревом почти аналогична работе со списком **List Control**. Только вместо индекса элемента (как в списке) используется дескриптор. Изменения в файлах приведены в листингах 7.70 и 7.71.

Листинг 7.70. Изменения в файле MyDlg.h для начальной инициализации дерева

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CTreeCtrl tree1;
   CImageList imtree;        // Список изображений
   HTREEITEM tr[4];          // Массив дескрипторов элементов дерева
};
```

Листинг 7.71. Изменения в файле MyDlg.cpp для начальной инициализации дерева

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Дерево
   // Создание, заполнение и подключение списка изображений
   imtree.Create(13,13, NULL, 0, 1);
   imtree.Add(AfxGetApp()->LoadIconW(IDI_ICON1));
```

```
imtree.Add(AfxGetApp()->LoadIconW(IDI_ICON2));
imtree.Add(AfxGetApp()->LoadIconW(IDI_ICON3));
tree1.SetImageList(&imtree, TVSIL_NORMAL);
// Заполнение дерева
TVITEM tv;                // Атрибуты элемента дерева
TVINSERTSTRUCT tvins;     // Информация для добавления нового элемента
                          // к дереву
// Добавление корня дерева
tv.mask = TVIF_TEXT | TVIF_IMAGE | TVIF_SELECTEDIMAGE;
tv.pszText = L"ROOT";
tv.cchTextMax = 5;            // Размер pszText
tv.iImage = 0;                // Индекс иконки (если элемент не выбран)
tv.iSelectedImage = 0;        // Индекс иконки (если элемент выбран)
tvins.hParent = TVI_ROOT;     // Корень
tvins.hInsertAfter = TVI_FIRST;   // Первый из корней
tvins.item = tv;
tr[0] = tree1.InsertItem(&tvins);
// Добавление элемента1, выпадающего из корня
tv.pszText = L"elem1";
tv.cchTextMax = 6;
tv.iImage = 1;
tv.iSelectedImage = 1;
tvins.hParent = tr[0];        // Родитель (ROOT)
tvins.hInsertAfter = TVI_FIRST;   // Первый из дочерних 1
tvins.item = tv;
tr[1] = tree1.InsertItem(&tvins);
// Добавление элемента 1_1, выпадающего из элемента 1
tv.pszText = L"elem1_1";
tv.cchTextMax = 8;
tv.iImage = 2;
tv.iSelectedImage = 2;
tvins.hParent = tr[1];         // Родитель (elem1)
tvins.hInsertAfter = TVI_FIRST;   // Первый из дочерних 1_1
tvins.item = tv;
tr[2] = tree1.InsertItem(&tvins);
```

```
   // Добавление элемента 2, выпадающего из корня
   tv.pszText = L"elem2";
   tv.cchTextMax = 6;
   tv.iImage = 1;
   tv.iSelectedImage = 1;
   tvins.hParent = tr[0];           // Родитель (ROOT)
   tvins.hInsertAfter = TVI_LAST;   // Последний из дочерних 1
   tvins.item = tv;
   tr[3] = tree1.InsertItem(&tvins);

   return TRUE;
}
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // Работа с выбранным элементом дерева
   // Получить дескриптор выбранного элемента
   HTREEITEM ret = tree1.GetSelectedItem();
   int i;
   // Узнать, какой элемент из списка дескрипторов выбран
   for(i = 0; i < 4; i++)
   {
      if(ret == tr[i])
         break;
   }
   if(i < 4)            // Если был выбран элемент из списка
   {
      CString st;
      st = tree1.GetItemText(ret);
      AfxMessageBox(st);
   }

   OnOK();
}
```

Установить новый список изображений для дерева можно с помощью функции:

```
CImageList* CTreeCtrl::SetImageList( // Указатель на предыдущий список
                                     // или 0 - при ошибке
   CImageList * pImageList,          // Указатель на устанавливаемый список
   int nImageListType);              // Тип списка изображений
```

Типы списков изображений для дерева (`nImageListType`) могут быть такие:

- `TVSIL_NORMAL` — список обычных изображений для выбранных и невыбранных элементов;
- `TVSIL_STATE` — список с изображениями состояний.

Атрибуты элемента дерева задаются как структура:

```
typedef struct tagTVITEM
{
   UINT mask;            // Какие из оставшихся полей содержат информацию
   HTREEITEM hItem;      // Дескриптор элемента
   UINT state;           // Текущее состояние элемента
   UINT stateMask;       // Возможное состояние элемента
   LPTSTR pszText;       // Текст элемента
   int  cchTextMax;      // Размер буфера для pszText
   int iImage;           // Индекс иконки не выбранного элемента
   int iSelectedImage;   // Индекс иконки выбранного элемента
   int cChildren;        // Количество дочерних записей у данного элемента
   LPARAM lParam;        // 32-битное значение, ассоциируемое с записью
} TVITEM, *LPTVITEM;
```

Значение полей (`mask`) можно комбинировать:

- `TVIF_CHILDREN` — поле `cChildren` содержит данные;
- `TVIF_DI_SETITEM` — затребованная информация об элементах хранится в операционной системе;
- `TVIF_HANDLE` — поле `hItem` содержит данные;
- `TVIF_IMAGE` — поле `iImage` содержит данные;
- `TVIF_PARAM` — поле `lParam` содержит данные;
- `TVIF_SELECTEDIMAGE` — поле `iSelectedImage` содержит данные;
- `TVIF_STATE` — поля `state` и `stateMask` содержат данные;
- `TVIF_TEXT` — поля `pszText` и `cchTextMax` содержат данные.

Значения состояния (state) и (stateMask) могут быть следующими:

- TVIS_BOLD — запись отображается жирным шрифтом;
- TVIS_CUT — запись выделена для операции вырезания и вставки (cut-and-paste);
- TVIS_DROPHILITED — запись выделена для переноса (drag-and-drop);
- TVIS_EXPANDED — список дочерних записей должен быть развернут;
- TVIS_EXPANDEDONCE — список дочерних записей был развернут хотя бы один раз;
- TVIS_EXPANDPARTIAL — дерево частично развернуто (версия Visual C++ 4.70);
- TVIS_SELECTED — запись выбрана.

Информация для добавления нового элемента к дереву хранится в структуре, описанной как:

```
typedef struct tagTVINSERTSTRUCT
{
   HTREEITEM hParent;          // Дескриптор родительской записи
   HTREEITEM hInsertAfter;     // Дескриптор записи, после которой должна
                               // быть вставлена новая запись
#if (_WIN32_IE >= 0x0400)
   union
   {
      TVITEMEX itemex;         // Для версии Visual C++ 4.71 структура
                               // с атрибутами элемента
      TVITEM item;             // Структура с атрибутами записи
   } DUMMYUNIONNAME;
#else
   TVITEM item;                // Структура с атрибутами записи
#endif
} TVINSERTSTRUCT, *LPTVINSERTSTRUCT;
```

Если hParent = TVI_ROOT или NULL, то происходит вставка корневой записи.

Значения дескриптора записи (hInsertAfter) могут быть такими:

- TVI_FIRST — запись вставляется в начало списка;
- TVI_LAST — запись вставляется в конец списка;

- `TVI_ROOT` — добавляемый элемент является корнем;
- `TVI_SORT` — запись вставляется в список в алфавитном порядке.

Вставка новой записи в дерево выполняется с помощью функции:

```
HTREEITEM CTreeCtrl::InsertItem(       // Дескриптор вставленной записи
   LPTVINSERTSTRUCT lpInsertStruct);   // Указатель на структуру
                                       // TVINSERTSTRUCT
```

или

```
HTREEITEM CTreeCtrl::InsertItem(
   UINT nMask,
   LPCTSTR lpszItem,
   int nImage,
   int nSelectedImage,
   UINT nState,
   UINT nStateMask,
   LPARAM lParam,
   HTREEITEM hParent,
   HTREEITEM hInsertAfter);
```

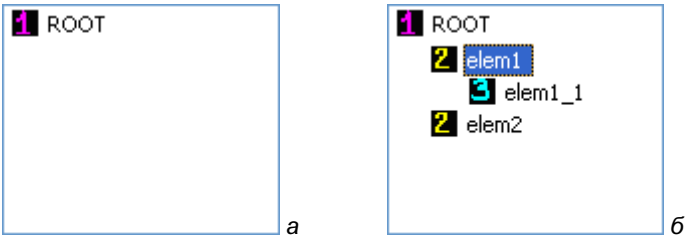
или

```
HTREEITEM CTreeCtrl::InsertItem(
   LPCTSTR lpszItem,
   HTREEITEM hParent = TVI_ROOT,
   HTREEITEM hInsertAfter = TVI_LAST);
```

или

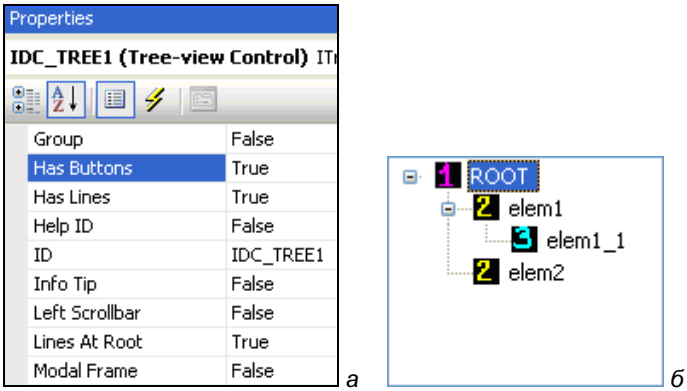
```
HTREEITEM CTreeCtrl::InsertItem(
   LPCTSTR lpszItem,
   int nImage,
   int nSelectedImage,
   HTREEITEM hParent = TVI_ROOT,
   HTREEITEM hInsertAfter = TVI_LAST);
```

Вид списка при работе программы показан на рис. 7.55. Чтобы дерево выглядело обычным образом, надо в окне свойств дерева в полях **Has Buttons** (Имеет кнопки), **Has Lines** (Имеет линии) и **Lines At Root** (Линии в корне) задать значение `True` (рис. 7.56).



а *б*

Рис. 7.55. Вид дерева по умолчанию (*а*) и в раскрытом виде (*б*)



а *б*

Рис. 7.56. Изменения свойств дерева (*а*)
и соответствующее изменение его отображения (*б*)

Добавим в обработку нажатия кнопки **OK** окна диалога печать сообщения о выбранном элементе дерева. Изменения в файле приведены в листинге 7.72.

Листинг 7.72. Изменения в файле MyDlg.cpp для обработки выбранного элемента дерева при закрытии окна диалога

```
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // Работа с выбранным элементом дерева
   // Получить дескриптор выбранного элемента
   HTREEITEM ret = tree1.GetSelectedItem();
   int i;
```

```
   // Узнать, какой элемент из списка дескрипторов выбран
   for(i = 0; i < 4; i++)
   {
      if(ret == tr[i])
         break;
   }
   if(i < 4)                  // Если был выбран элемент из списка
   {
      CString st;
      st = tree1.GetItemText(ret);
      AfxMessageBox(st);
   }

   OnOK();
}
```

Получить дескриптор выбранного элемента можно с помощью функции:

```
HTREEITEM CTreeCtrl::GetSelectedItem() const;
```

Получить текст записи элемента:

```
CString CTreeCtrl::GetItemText(     // Текст записи
   HTREEITEM hItem) const;          // Дескриптор элемента
```

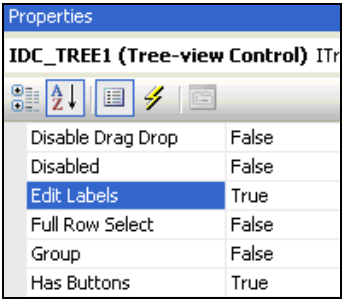


Рис. 7.57. Изменение свойств дерева для возможности редактирования элементов

Аналогично действиям со списком добавим возможность изменения названий элементов дерева. Для этого надо в окне свойств дерева в поле

Edit Labels задать значение `True` (рис. 7.57) и добавить для дерева обработку сообщения `TVN_ENDLABELEDIT` (рис. 7.58), для этого:

- в окне свойств списка выбрать вкладку **Control Events**;
- в списке событий найти `TVN_ENDLABELEDIT` и выбрать **<Add> OnTvnEndlabeleditTree1**.

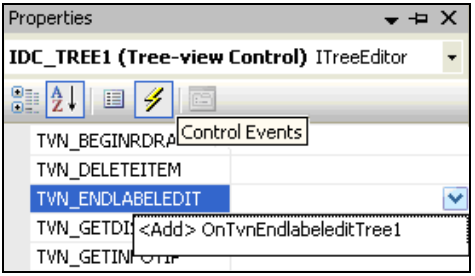


Рис. 7.58. Добавления обработки события редактирования элемента

Изменения в файлах приведены в листингах 7.73 и 7.74.

Листинг 7.73. Изменения в файле MyDlg.h для возможности редактирования элемента дерева

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnTvnEndlabeleditTree1(NMHDR *pNMHDR, LRESULT *pResult);
};
```

Листинг 7.74. Изменения в файле MyDlg.cpp для возможности редактирования элемента дерева

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
// ...
```

```
   ON_NOTIFY(TVN_ENDLABELEDIT, IDC_TREE1, &MyDlg::OnTvnEndlabeleditTree1)
END_MESSAGE_MAP()
// ...
void MyDlg::OnTvnEndlabeleditTree1(NMHDR *pNMHDR, LRESULT *pResult)
{
   LPNMTVDISPINFO pTVDispInfo =
   reinterpret_cast<LPNMTVDISPINFO>(pNMHDR);
   // TODO: Add your control notification handler code here
   *pResult = 0;
}
```

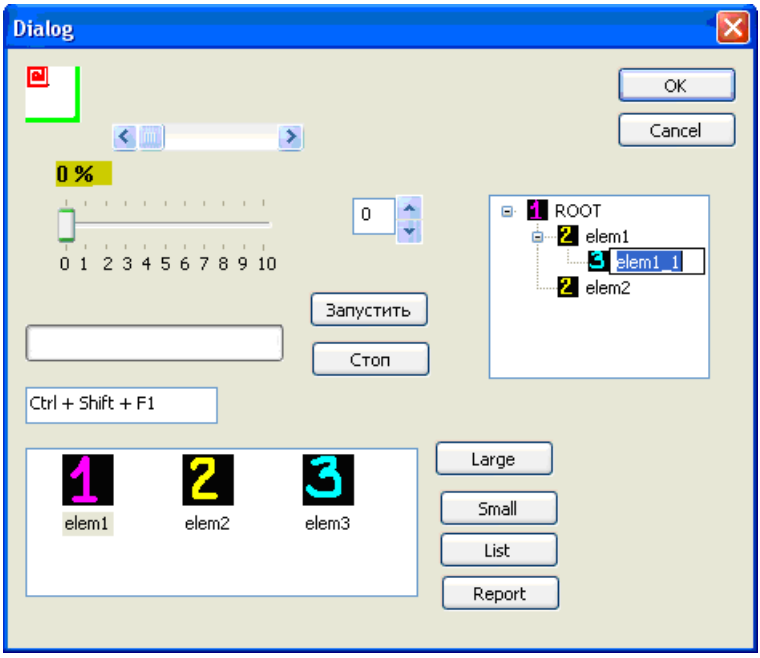
Структура с данными об изменении состояния дерева определена как:

```
typedef struct tagNMTVDISPINFO
{
   NMHDR hdr;                              // см. разд. 7.1.4.
   TVITEM item;                            // Структура TVITEM
} NMTVDISPINFO, *LPNMTVDISPINFO;
```

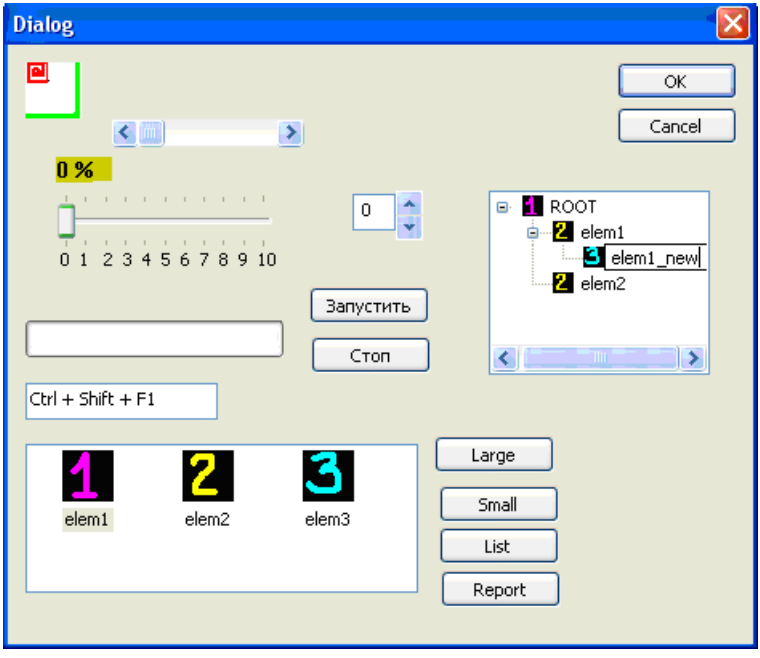
Добавим обработку этого сообщения. Изменения в файле приведены в листинге 7.75.

Листинг 7.75. Изменения в файле MyDlg.cpp для отображения информации об изменении элемента дерева

```
// ...
void MyDlg::OnTvnEndlabeleditTree1(NMHDR *pNMHDR, LRESULT *pResult)
{
   LPNMTVDISPINFO pTVDispInfo = reinterpret_cast<LPNMTVDISPINFO>(pNMHDR);
   // TODO: Add your control notification handler code here
   if(pTVDispInfo->item.pszText != NULL)
   {
      tree1.SetItemText(pTVDispInfo->item.hItem,
                        pTVDispInfo->item.pszText);
   }

   *pResult = 0;
}
```



а



б

Рис. 7.59. Редактирование элемента дерева (*а*), измененный элемент дерева (*б*)

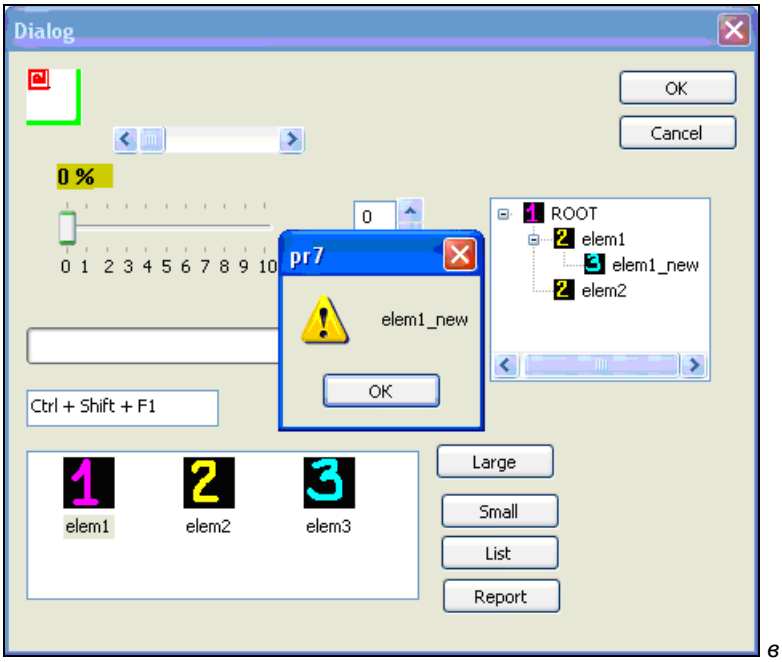


Рис. 7.59. Обработка выбранного элемента (*в*)

Результат работы программы показан на рис. 7.59. У дерева изменили название элемента с **elem1_1** на **elem1_new** и нажали в окне диалога кнопку **OK**.

7.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 7.76. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDR_MAINFRAME                   128
#define IDR_pr7TYPE                     129
#define IDD_DIALOG1                     130
```

```
#define IDB_BITMAP1                     131
#define IDI_ICON1                       132
#define IDI_ICON2                       133
#define IDI_ICON3                       134
#define IDC_SCROLLBAR1                  1007
#define IDC_SLIDER1                     1009
#define IDC_SPIN1                       1010
#define IDC_EDIT1                       1011
#define IDC_PROGRESS1                   1012
#define IDC_BUTTON1                     1013
#define IDC_BUTTON2                     1014
#define IDC_HOTKEY1                     1015
#define IDC_LIST1                       1016
#define IDC_BUTTON3                     1017
#define IDC_BUTTON4                     1018
#define IDC_BUTTON5                     1019
#define IDC_BUTTON6                     1020
#define IDC_TREE1                       1021
#define ID_MY_DLG                       32771
// ...
```

Листинг 7.77. Файл MyDlg.h (объявление класса диалога, добавлен мастером)

```
// MyDlg.h
#pragma once
#include "afxwin.h"
#include "afxcmn.h"

// MyDlg dialog
class MyDlg : public CDialog
{
   DECLARE_DYNAMIC(MyDlg)

public:
   MyDlg(CWnd* pParent = NULL);   // standard constructor
   virtual ~MyDlg();
```

```
// Dialog Data
   enum { IDD = IDD_DIALOG1 };


protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support


   DECLARE_MESSAGE_MAP()
public:
   virtual BOOL OnInitDialog();
   CBitmap bmp;
public:
   int scrollbar1;
public:
   afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar);
public:
   int slider1;
public:
   afx_msg void OnPaint();
public:
   int edit1;
public:
   afx_msg void OnDeltaposSpin1( NMHDR *pNMHDR, LRESULT *pResult);
public:
   CEdit wedit1;
public:
   afx_msg void OnEnUpdateEdit1();
public:
   afx_msg void OnBnClickedButton1();
public:
   afx_msg void OnBnClickedButton2();
   bool ftimer;            // true - таймер запущен, false - нет
public:
   CProgressCtrl progr;
public:
   afx_msg void OnTimer(UINT_PTR nIDEvent);
```

```
public:
   CHotKeyCtrl hotkey1;
public:
   afx_msg void OnBnClickedOk();
public:
   CListCtrl list1;
   CImageList i_large;              // Список больших икон
   CImageList i_small;              // Список маленьких икон
public:
   afx_msg void OnBnClickedButton3_Large();
public:
   afx_msg void OnBnClickedButton4_Small();
public:
   afx_msg void OnBnClickedButton5_List();
public:
   afx_msg void OnBnClickedButton6_Report();
public:
   afx_msg void OnLvnEndlabeleditList1(NMHDR *pNMHDR, LRESULT *pResult);
public:
   CTreeCtrl tree1;
   CImageList imtree;        // Список изображений
   HTREEITEM tr[4];          // Массив дескрипторов элементов дерева
public:
   afx_msg void OnTvnEndlabeleditTree1(NMHDR *pNMHDR, LRESULT *pResult);
};
```

Листинг 7.78. Файл MyDlg.cpp (определение класса диалога, добавлен мастером)

```
// MyDlg.cpp : implementation file
//
#include "stdafx.h"
#include "pr7.h"
#include "MyDlg.h"

// MyDlg dialog
IMPLEMENT_DYNAMIC(MyDlg, CDialog)
```

```
MyDlg::MyDlg(CWnd* pParent /*=NULL*/)     : CDialog(MyDlg::IDD, pParent)
   , scrollbar1(0)
   , slider1(0)
   , edit1(0)
{
   ftimer = false;
}

static WORD vk = VK_F1,                          // Виртуальный код клавиши
mod = HOTKEYF_SHIFT | HOTKEYF_CONTROL;      // Клавиши-модификаторы

MyDlg::~MyDlg()
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   DDX_Scroll(pDX, IDC_SCROLLBAR1, scrollbar1);
   DDX_Slider(pDX, IDC_SLIDER1, slider1);
   DDX_Text(pDX, IDC_EDIT1, edit1);
   DDX_Control(pDX, IDC_EDIT1, wedit1);
   DDX_Control(pDX, IDC_PROGRESS1, progr);
   DDX_Control(pDX, IDC_HOTKEY1, hotkey1);
   DDX_Control(pDX, IDC_LIST1, list1);
   DDX_Control(pDX, IDC_TREE1, tree1);
}

BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_WM_HSCROLL()
   ON_WM_PAINT()
   ON_NOTIFY(UDN_DELTAPOS, IDC_SPIN1, &MyDlg::OnDeltaposSpin1)
   ON_EN_UPDATE(IDC_EDIT1, &MyDlg::OnEnUpdateEdit1)
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
   ON_BN_CLICKED(IDC_BUTTON2, &MyDlg::OnBnClickedButton2)
   ON_WM_TIMER()
```

```
    ON_BN_CLICKED(IDOK, &MyDlg::OnBnClickedOk)
    ON_BN_CLICKED(IDC_BUTTON3, &MyDlg::OnBnClickedButton3_Large)
    ON_BN_CLICKED(IDC_BUTTON4, &MyDlg::OnBnClickedButton4_Small)
    ON_BN_CLICKED(IDC_BUTTON5, &MyDlg::OnBnClickedButton5_List)
    ON_BN_CLICKED(IDC_BUTTON6, &MyDlg::OnBnClickedButton6_Report)
    ON_NOTIFY(LVN_ENDLABELEDIT, IDC_LIST1, &MyDlg::OnLvnEndlabeleditList1)
    ON_NOTIFY(TVN_ENDLABELEDIT, IDC_TREE1, &MyDlg::OnTvnEndlabeleditTree1)
END_MESSAGE_MAP()

// MyDlg message handlers

BOOL MyDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO:  Add extra initialization here

    // Рисунок
    CStatic *pbmp = (CStatic *)this->GetDlgItem(IDC_STATIC);
    bmp.LoadBitmapW(IDB_BITMAP1);
    pbmp->SetBitmap(bmp.operator HBITMAP());

    // Регулятор
    CSliderCtrl *psl = (CSliderCtrl *)this->GetDlgItem(IDC_SLIDER1);
    psl->SetRangeMin(0);
    psl->SetRangeMax(10);

    // Счетчик
    CSpinButtonCtrl *pspin =
                          (CSpinButtonCtrl *)this->GetDlgItem(IDC_SPIN1);
    // Диапазон значений для счетчика
    pspin->SetRange(0,10);
    // Начальное значение
    pspin->SetPos(0);
    CEdit *pedit1 = (CEdit *)GetDlgItem(IDC_EDIT1);
```

```
// Привязка счетчика к окну редактирования
pspin->SetBuddy(pedit1);

// Индикатор CProgressCtrl
progr.SetRange(1,8);
progr.SetStep(1);
progr.SetPos(0);

// Горячая клавиша
hotkey1.SetHotKey(vk, mod);

// Список с иконками
// Создание списков маленьких и больших иконок
i_small.Create(16,16, NULL, 0, 1);
i_large.Create(32, 32, NULL, 0, 1);
// Добавление иконок в список
i_small.Add(AfxGetApp()->LoadIconW( IDI_ICON1));
i_small.Add(AfxGetApp()->LoadIconW( IDI_ICON2));
i_small.Add(AfxGetApp()->LoadIconW( IDI_ICON3));
i_large.Add(AfxGetApp()->LoadIconW( IDI_ICON1));
i_large.Add(AfxGetApp()->LoadIconW( IDI_ICON2));
i_large.Add(AfxGetApp()->LoadIconW( IDI_ICON3));
// Подключение списков иконок к реальному рабочему списку
list1.SetImageList(&i_small, LVSIL_SMALL);
list1.SetImageList(&i_large, LVSIL_NORMAL);
// Заполнение самого списка (с привязкой к иконкам)
list1.InsertItem(0, L"elem1",0);
list1.InsertItem(1, L"elem2",1);
list1.InsertItem(2, L"elem3",2);
// Добавление трех колонок для таблицы
list1.InsertColumn(0, L"Название", LVCFMT_LEFT, 70);
list1.InsertColumn(1, L"Инф1", LVCFMT_LEFT, 70);
list1.InsertColumn(2, L"Инф2", LVCFMT_LEFT, 70);
// Добавление в список дополнительной информации по элементам для
// таблицы
```

```
list1.SetItemText(0, 1, L"elem1-inf1");
list1.SetItemText(0, 2, L"elem1-inf2");
list1.SetItemText(1, 1, L"elem2-inf1");
list1.SetItemText(1, 2, L"elem2-inf2");
list1.SetItemText(2, 1, L"elem3-inf1");
list1.SetItemText(2, 2, L"elem3-inf2");
// Выделенный элемент всегда остается
SetWindowLong (list1.m_hWnd, GWL_STYLE, GetWindowLong(list1.m_hWnd,
               GWL_STYLE) | LVS_SHOWSELALWAYS);

// Выделить первый элемент списка
list1.SetItem(0, 0, LVIF_STATE, NULL, 0, LVIS_SELECTED, LVIS_SELECTED,
              0);
list1.SetSelectionMark(0);

// Дерево
// Создание, заполнение и подключение списка изображений
imtree.Create(13,13, NULL, 0, 1);
imtree.Add(AfxGetApp()->LoadIconW( IDI_ICON1));
imtree.Add(AfxGetApp()->LoadIconW( IDI_ICON2));
imtree.Add(AfxGetApp()->LoadIconW( IDI_ICON3));
tree1.SetImageList(&imtree, TVSIL_NORMAL);
// Заполнение дерева
TVITEM tv;                  // Атрибуты элемента дерева
TVINSERTSTRUCT tvins;     // Информация для добавления нового элемента
                          // к дереву
// Добавление корня дерева
tv.mask = TVIF_TEXT | TVIF_IMAGE | TVIF_SELECTEDIMAGE;
tv.pszText = L"ROOT";
tv.cchTextMax = 5;          // Размер pszText
tv.iImage = 0;              // Индекс иконки (если элемент не выбран)
tv.iSelectedImage = 0;      // Индекс иконки (если элемент выбран)
tvins.hParent = TVI_ROOT;   // Корень
tvins.hInsertAfter = TVI_FIRST;       // Первый из корней
tvins.item = tv;
tr[0] = tree1.InsertItem(&tvins);
```

```
    // Добавление элемента1, выпадающего из корня
    tv.pszText = L"elem1";
    tv.cchTextMax = 6;
    tv.iImage = 1;
    tv.iSelectedImage = 1;
    tvins.hParent = tr[0];                    // Родитель (ROOT)
    tvins.hInsertAfter = TVI_FIRST;           // Первый из дочерних1
    tvins.item = tv;
    tr[1] = tree1.InsertItem(&tvins);
    // Добавление элемента1_1, выпадающего из элемента1
    tv.pszText = L"elem1_1";
    tv.cchTextMax = 8;
    tv.iImage = 2;
    tv.iSelectedImage = 2;
    tvins.hParent = tr[1];                    // Родитель (elem1)
    tvins.hInsertAfter = TVI_FIRST;           // Первый из дочерних1_1
    tvins.item = tv;
    tr[2] = tree1.InsertItem(&tvins);
    // Добавление элемента2, выпадающего из корня
    tv.pszText = L"elem2";
    tv.cchTextMax = 6;
    tv.iImage = 1;
    tv.iSelectedImage = 1;
    tvins.hParent = tr[0];              // Родитель (ROOT)
    tvins.hInsertAfter = TVI_LAST;      // Последний из дочерних1
    tvins.item = tv;
    tr[3] = tree1.InsertItem(&tvins);

    return TRUE;  // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}

void MyDlg::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call default
```

```
// Только для линейки прокрутки
if( pScrollBar->m_hWnd == GetDlgItem(IDC_SCROLLBAR1)->m_hWnd)
{
   // Установка значений линейки (от 0 до 10)
   pScrollBar->SetScrollRange(0, 10);
   switch(nSBCode)          // Что было нажато на линейке
   {
      // Зацепили и потащили ползунок
      case SB_THUMBPOSITION:
                        // Выставить ползунок в текущую позицию
                        pScrollBar->SetScrollPos(nPos);
                        break;
      // Нажали на стрелку влево
      case SB_LINELEFT:
                        nPos = pScrollBar->GetScrollPos();
                        nPos--;
                        pScrollBar->SetScrollPos(nPos);
                        break;
      // Нажали на стрелку вправо
      case SB_LINERIGHT:
                        nPos = pScrollBar->GetScrollPos();
                        nPos++;
                        pScrollBar->SetScrollPos(nPos);
                        break;
   }
}
else if(pScrollBar->m_hWnd == GetDlgItem(IDC_SLIDER1)->m_hWnd)
{
   // Для вывода значения регулятора посылаем сообщение о перерисовке
   // окна без перерисовки всего окна
   this->Invalidate(FALSE);
}

CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

```
void MyDlg::OnPaint()
{
   CPaintDC dc(this); // device context for painting
   // TODO: Add your message handler code here

   // Вывод значений регулятора
   CSliderCtrl *psl = (CSliderCtrl *)this->GetDlgItem(IDC_SLIDER1);
   CString str1;
   str1.Format(L"%d", psl->GetPos());
   str1 += " %";
   // Закрашиваем ("стираем") поле вывода текста
   dc.FillSolidRect(30, 70, 35, 15, RGB(230, 230, 0));
   dc.TextOutW(30, 70, str1);
   // Do not call CDialog::OnPaint() for painting messages
}

void MyDlg::OnDeltaposSpin1(NMHDR *pNMHDR, LRESULT *pResult)
{
   LPNMUPDOWN pNMUpDown = reinterpret_cast<LPNMUPDOWN>(pNMHDR);
   // TODO: Add your control notification handler code here

   // Удваиваем шаг счетчика (был 1, станет 2)
   pNMUpDown->iDelta *= 2;

   *pResult = 0;
}

void MyDlg::OnEnUpdateEdit1()
{
   // TODO:  If this is a RICHEDIT control, the control will not
   // send this notification unless you override the
   //CDialog::OnInitDialog()
   // function to send the EM_SETEVENTMASK message to the control
   // with the ENM_UPDATE flag ORed into the lParam mask.

   // TODO:  Add your control notification handler code here
   CString s;                // Содержимое текстового поля
```

```
    const wchar_t *wch;       // Си-строка с Unicode
    char *ch;                 // Си-строка без Unicode
    int z;                    // Содержимое текстового поля в целом формате
    size_t sizeb,             // Размер ch в байтах
           cnv;               // Количество преобразованных символов
    wedit1.GetWindowTextW(s); // Получить содержимое текстового поля
    wch = s.GetBuffer();      // Получить Си-строку с Unicode
    // Вычислить размер для многобайтовой строки
    sizeb = (s.GetLength()+1)*2;
    ch = new char[sizeb];
    wcstombs_s(&cnv, ch, sizeb, wch, sizeb);   // Преобразовать wch в ch
    z = atoi( ch );                 // Преобразовать ch в целое число
    if(z < 0 || z > 10)             // Если это число недопустимое
    {
       wedit1.SetSel(0, -1);       // Выделить весь текст в текстовом поле
       wedit1.ReplaceSel(L"0");    // Заменить его на "0"
    }
}
//CProgressCtrl
void MyDlg::OnBnClickedButton1()
{
    // TODO: Add your control notification handler code here
    // Запустить таймер1 с интервалом 1 сек
    if(!ftimer)
    {
       progr.SetPos(0);
       SetTimer(1, 1000, NULL);
       ftimer = true;
    }
}

void MyDlg::OnBnClickedButton2()
{
    // TODO: Add your control notification handler code here
    // Остановить таймер1
```

```
    if(ftimer)
    {
       KillTimer(1);
       ftimer = false;
    }
}

void MyDlg::OnTimer(UINT_PTR nIDEvent)
{
    // TODO: Add your message handler code here and/or call default

    CString s;
    int z;
    CTime t;
    SYSTEMTIME st;
    if(nIDEvent == 1)                // Если это таймер1
    {
       z = progr.GetPos();           // Получить текущую позицию индикатора
       CString s; s.Format(L"%d", z);
       if(z > 10)                    // Остановить таймер
       {
          OnBnClickedButton2();
          return CDialog::OnTimer(nIDEvent);
       }
       CClientDC dc(this);           // Получить контекст устройства для
                                     // вывода текста
       // "Стереть" старый текст (вывести пробелы)
       dc.TextOutW(20, 150, L"   ");
       // Вывести новй текст (текущее значение индикатора)
       dc.TextOutW( 20, 150, s );
       progr.StepIt();               // Увеличить индикатор на размер шага

       t = CTime::GetCurrentTime(); // Получить текущее время
       // Преобразовать его в строку вида "чч:мм:сс"
       s = t.Format("%H:%M:%S");
```

```
      // Заполнить структуру с данными о дате и времени
      t.GetAsSystemTime(st);
      dc.TextOutW(50, 150, L"        ");
      // Вывести текущее значение времени
      dc.TextOutW( 50, 150, s );
      if(st.wSecond % 5 == 0)     // Если текущее значение секунд кратно 5
      {
         OnBnClickedButton2();    // Остановить таймер1
      }
   }  // end if(nIDEvent == 1)
   CDialog::OnTimer(nIDEvent);
}

void MyDlg::OnBnClickedOk()
{
   // TODO: Add your control notification handler code here

   // Получить и сохранить (в vk и mod)текущие значения горячей клавиши
   hotkey1.GetHotKey(vk, mod);
   // Получить полный единый код горячей клавиши
   DWORD allkey = hotkey1.GetHotKey();
   // Послать сообщение для регистрации горячей клавиши и ее код
   AfxGetMainWnd()->SendMessage(WM_SETHOTKEY, allkey);

   // Работа со списком
   // Получить индекс выбранного элемента списка
   int item = list1.GetSelectionMark();
   if(item != -1)                    // Если был выбран элемент
   {
      // Получить всю текстовую информацию выбранного элемента
      CString s;
      s += L"Название элемента: ";
      s += list1.GetItemText(item, 0);
      s += L"\nИнф1: ";
      s += list1.GetItemText(item, 1);
```

```
      s += L"\nИнф2: ";
      s += list1.GetItemText(item, 2);
      AfxMessageBox(s);
   }
   else                          // Если элемент не выбран
   {
      AfxMessageBox(L"Вы не выбрали элемент в списке");
      return;                    // Не выходить из окна диалога
   }

   // Работа с выбранным элементом дерева
   // Получить дескриптор выбранного элемента
   HTREEITEM ret = tree1.GetSelectedItem();
   int i;
   // Узнать, какой элемент из списка дескрипторов выбран
   for(i = 0; i < 4; i++)
   {
      if(ret == tr[i])
         break;
   }
   if(i < 4)                     // Если был выбран элемент из списка
   {
      CString st;
      st = tree1.GetItemText(ret);
      AfxMessageBox(st);
   }

   OnOK();
}

void MyDlg::OnBnClickedButton3_Large()
{
   // TODO: Add your control notification handler code here
   // Получить текущий стиль
```

```
    DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
    // Задать новый стиль на основе старого, изменив только вид списка
    SetWindowLong(list1.m_hWnd, GWL_STYLE,
                  (style&~LVS_TYPEMASK) | LVS_ICON);
}

void MyDlg::OnBnClickedButton4_Small()
{
    // TODO: Add your control notification handler code here
    // Получить текущий стиль
    DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
    // Задать новый стиль на основе старого, изменив только вид списка
    SetWindowLong(list1.m_hWnd, GWL_STYLE,
                  (style&~LVS_TYPEMASK ) | LVS_SMALLICON);
}

void MyDlg::OnBnClickedButton5_List()
{
    // TODO: Add your control notification handler code here
    // Получить текущий стиль
    DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
    // Задать новый стиль на основе старого, изменив только вид списка
    SetWindowLong(list1.m_hWnd, GWL_STYLE,
                  (style&~LVS_TYPEMASK ) | LVS_LIST);
}

void MyDlg::OnBnClickedButton6_Report()
{
    // TODO: Add your control notification handler code here
    // Получить тукущий стиль
    DWORD style = GetWindowLong(list1.m_hWnd,GWL_STYLE);
    // Задать новый стиль на основе старого, изменив только вид списка
    SetWindowLong(list1.m_hWnd, GWL_STYLE,
                  (style&~LVS_TYPEMASK ) | LVS_REPORT);
}
```

```
void MyDlg::OnLvnEndlabeleditList1(NMHDR *pNMHDR, LRESULT *pResult)
{
   NMLVDISPINFO *pDispInfo = reinterpret_cast<NMLVDISPINFO*>(pNMHDR);
   // TODO: Add your control notification handler code here
   if(pDispInfo->item.pszText != NULL)
   {
      list1.SetItemText(pDispInfo->item.iItem, 0,
                        pDispInfo->item.pszText);
      list1.SetItemText(pDispInfo->item.iItem, 1, L"Изменен");
   }
   *pResult = 0;
}

void MyDlg::OnTvnEndlabeleditTree1(NMHDR *pNMHDR, LRESULT *pResult)
{
   LPNMTVDISPINFO pTVDispInfo = reinterpret_cast<LPNMTVDISPINFO>(pNMHDR);
   // TODO: Add your control notification handler code here
   if(pTVDispInfo->item.pszText != NULL)
   {
      tree1.SetItemText(pTVDispInfo->item.hItem,
                        pTVDispInfo->item.pszText);
   }
   *pResult = 0;
}
```

Листинг 7.79. Файл ChildView.h (объявление класса представления)

```
// ...
class CChildView : public CWnd
{
// ...
public:
   CBitmap bmpv;
   CStatic sbmp;
// ...
```

```
public:
   afx_msg void OnMyDlg();
};
```

Листинг 7.80. Файл ChildView.cpp (определение класса представления)

```
// ...
#include "ChildView.h"
#include "MyDlg.h"
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_COMMAND(ID_MY_DLG, &CChildView::OnMyDlg)
END_MESSAGE_MAP()
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here

   MyDlg dlg;
   INT_PTR nRet;
   nRet = dlg.DoModal();
   static int n;        // Флаг показа картинки (0 - создать и показать,
                        // 1 - стереть и удалить

   if(nRet == IDOK)
   {
      // Работа с рисунком
      if(!n)
      {
         bmpv.LoadBitmapW(IDB_BITMAP1);
         sbmp.Create(NULL, WS_CHILD | WS_VISIBLE | SS_BITMAP |
                     SS_CENTERIMAGE, CRect(50,50,85,85), this,
                     IDC_STATIC);
         sbmp.SetBitmap(bmpv.operator HBITMAP());
         n = 1;
```

```
      }
      // Работа с линейкой прокрутки
      CString str1;
      str1.Format(L"%d", dlg.scrollbar1);
      AfxMessageBox(str1);

      // Работа с регулятором
      str1.Format(L"%d", dlg.slider1);
      str1 += " (slider1)";
      AfxMessageBox(str1);
   }
   else                        // if(nRet == IDCANCEL)
   {
      bmpv.DeleteObject();
      sbmp.DestroyWindow();
      n = 0;
   }
}
```

ГЛАВА 8

Вспомогательные элементы управления диалоговых окон

Создайте проект **pr8** аналогично проекту **pr1** *(см. разд. 1.1)*, *отключив Unicode*. Для этого измените следующие опции относительно изначально предложенных мастером:

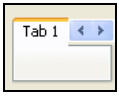
- ❒ на вкладке **Application Type**:
  - SDI-документ;
  - без поддержки архитектуры документ/представление;
  - отключить флажок **Use Unicode libraries**;
- ❒ на вкладке **User Interface Features:**
  - без строки статуса;
  - без панели инструментов.

8.1. Описание программы

Добавьте в проект диалоговое окно `IDD_DIALOG1`, класс диалога `MyDlg`, меню для его вызова **My | Dlg** *(см. разд. 6.1.1)* и функцию `OnInitDialog()` *(см. разд. 6.1.5)*.

В этой главе рассмотрим следующие элементы управления:

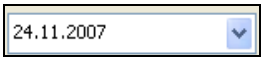
- ❒ **Tab Control** — набор вкладок (`class CTabCtrl : public CWnd`);



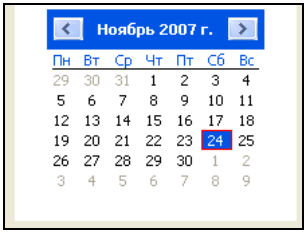
- **Animation Control** — анимация (`class CAnimateCtrl : public CWnd`);

- **Rich Edit 2.0 Control** — расширенное текстовое поле (`class CRichEditCtrl : public CWnd`);

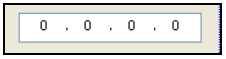
- **Date Time Picker** — дата и время (`class CDateTimeCtrl : public CWnd`);



- **Month Calendar Control** — календарь (`class CMonthCalCtrl : public CWnd`);



- **IP Address Control** — IP-адрес (`class CIPAddressCtrl : public CWnd`);



- **Extended Combo Box** — расширенное поле со списком (`class CComboBoxEx : public CComboBox`).

8.1.1. Набор вкладок (*Tab Control*)

Добавим набор вкладок **Tab Control** (рис. 8.1) и переменную (категории **Control**) `tab1` для работы с ним (рис. 8.2). Изменения в файлах приведены в листингах 8.1—8.3.

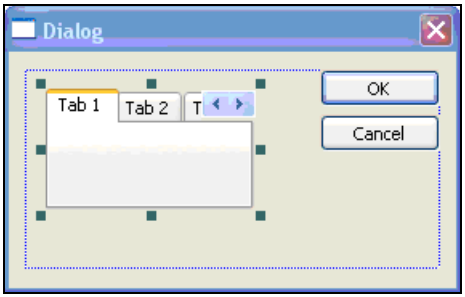


Рис. 8.1. Добавление набора вкладок в окно диалога

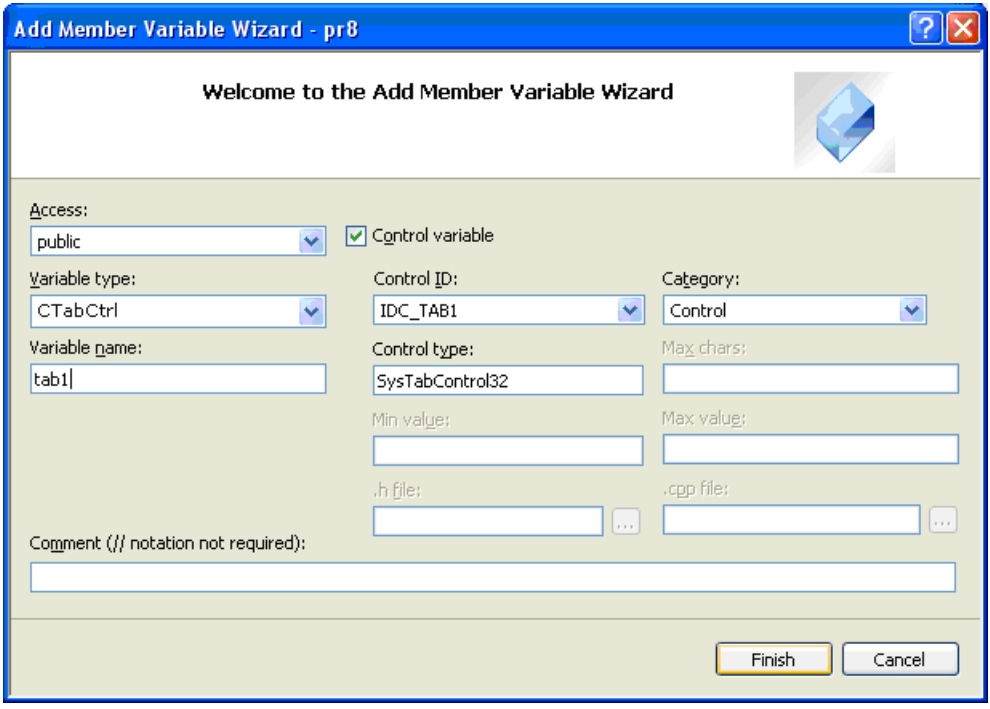


Рис. 8.2. Переменная для работы с набором вкладок

Листинг 8.1. Изменения в файле Resource.h для работы с набором вкладок

```
// ...
#define IDC_TAB1                    1000
// ...
```

Листинг 8.2. Изменения в файле MyDlg.h для работы с набором вкладок

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CTabCtrl tab1;
};
```

Листинг 8.3. Изменения в файле MyDlg.cpp для работы с набором вкладок

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_TAB1, tab1);
}
```

В названии страниц (вкладок) можно использовать иконки. Добавим в ресурсы две иконки с изображением цифр 1 и 2 *(см. разд. 7.1.8)*. Объявим переменную для работы со списком изображений. В функции инициализации диалога создадим список изображений, заполним его и заполним список страниц (вкладок). Изменения в файлах приведены в листингах 8.4—8.6.

Листинг 8.4. Изменения в файле Resource.h для добавления иконок на вкладки

```
// ...
#define IDI_ICON1                       131
#define IDI_ICON2                       132
// ...
```

Листинг 8.5. Изменения в файле MyDlg.h для добавления иконок на вкладки

```
// ...
class MyDlg : public CDialog
```

```
{
// ...
public:
   CTabCtrl tab1;
   CImageList image;              // Список изображений
};
```

Листинг 8.6. Изменения в файле MyDlg.cpp для добавления иконок на вкладки

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Список с иконками
   image.Create(16, 16, NULL, 0, 1);
   image.Add(theApp.LoadIconA(IDI_ICON1));
   image.Add(AfxGetApp()->LoadIconA(IDI_ICON2));     // Можно и так
   tab1.SetImageList(&image);
   // Заполнение страниц (вкладок)
   tab1.InsertItem(0,"Стр1",0);
   tab1.InsertItem(1,"Стр2",1);
}
```

Заполнение списка страниц-вкладок выполняется с помощью функции:

```
LONG CTabCtrl::InsertItem(  // Начальный индекс нового списка
   int nItem,               // Индекс новой (добавляемой) страницы
   TCITEM* pTabCtrlItem);   // Указатель на структуру TCITEM
```

или

```
LONG CTabCtrl::InsertItem(
   int nItem,
   LPCTSTR lpszItem);        // Текст – заголовок страницы
```

или

```
LONG CTabCtrl::InsertItem(
   int nItem,
```

```
   LPCTSTR lpszItem,
   int nImage);              // Индекс иконки в списке изображений
```

или

```
LONG CTabCtrl::InsertItem(  // см. TCITEM
   UINT nMask,
   int nItem,
   LPCTSTR lpszItem,
   int nImage,
   LPARAM lParam);
```

или

```
LONG CTabCtrl::InsertItem(  // см. TCITEM
   UINT nMask,
   int nItem,
   LPCTSTR lpszItem,
   int nImage,
   LPARAM lParam,
   DWORD dwState,
   DWORD dwStateMask);
```

Структура с информацией об элементе-вкладке определена как:

```
typedef struct tagTCITEM
{
   UINT mask;                // Задание заполнения элементов структуры
#if (_WIN32_IE >= 0x0300)
   DWORD dwState;            // Текущее состояние элемента
   DWORD dwStateMask;        // Задействованные биты dwState
#else
   UINT lpReserved1;         // Не используется
   UINT lpReserved2;         // Не используется
#endif
   LPTSTR pszText;           // Заголовок вкладки
   int cchTextMax;           // Размер pszText
   int iImage;               // Индекс иконки в списке изображений
                             // или -1 (без иконки)
   LPARAM lParam;            // 32-битное значение, ассоциируемое с записью
} TCITEM, *LPTCITEM;
```

Значения полей заполнения вкладки (`mask`) могут быть следующими:

- `TCIF_IMAGE` — переменная `iImage` содержит данные;
- `TCIF_PARAM` — переменная `lParam` содержит данные;
- `TCIF_RTLREADING` — текст `pszText` отображается справа налево;
- `TCIF_STATE` — переменная `dwState` содержит данные;
- `TCIF_TEXT` — переменная `pszText` содержит данные.

Текущее состояние вкладки (`dwState`) может принимать значения:

- `TCIS_BUTTONPRESSED` — элемент выбран (версия Visual C++ 4.70);
- `TCIS_HIGHLIGHTED` — элемент выделен и отображается с использованием текущего цвета выделения (версия Visual C++ 4.71).

Добавим обработку выбранной вкладки перед закрытием окна диалога. Для этого добавим обработку нажатия кнопки **OK** в окне диалога (дважды щелкнув по ней левой кнопкой мыши в окне ресурсов диалога). Изменения в файлах приведены в листингах 8.7 и 8.8.

Листинг 8.7. Изменения в файле MyDlg.h для обработки выбранной вкладки при закрытии окна диалога

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   afx_msg void OnBnClickedOk();
};
```

Листинг 8.8. Изменения в файле MyDlg.cpp для обработки выбранной вкладки при закрытии окна диалога

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_BN_CLICKED(IDOK, &MyDlg::OnBnClickedOk)
END_MESSAGE_MAP()
```

```
// ...
void MyDlg::OnBnClickedOk()
{
    // TODO: Add your control notification handler code here

    // TabControl
    // Получить индекс текущего выбранного элемента
    int a = tab1.GetCurSel();
    CString s;
    s.Format("Выбрана страница %d", a+1);
    AfxMessageBox(s);

    OnOK();
}
```

Получить индекс текущего выбранного элемента можно с помощью функции:

```
int CTabCtrl::GetCurSel() const;   // Индекс текущего выбранного элемента
```

Результат работы программы показан на рис. 8.3.

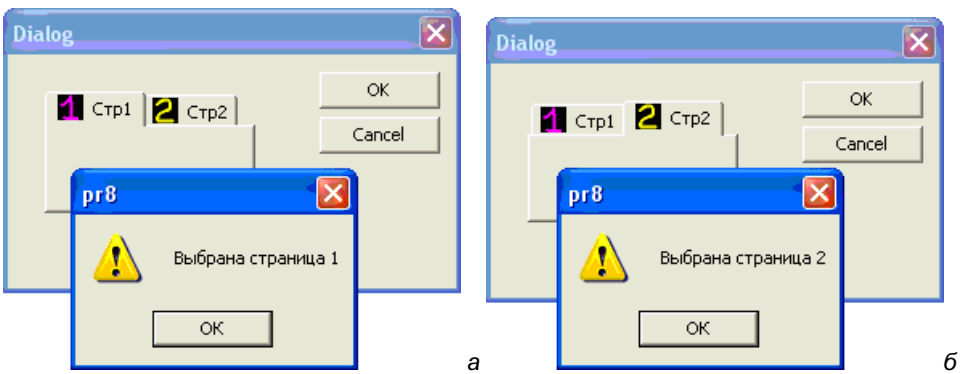


Рис. 8.3. Обработка выбора вкладки 1 (*а*) и вкладки 2 (*б*) при нажатии кнопки **OK** в окне диалога

Еще вкладки удобно использовать как список кнопок с картинками. Для этого надо в окне свойств набора вкладок задать в поле **Buttons** значение `True` (рис. 8.4). Результат работы программы показан на рис. 8.5.

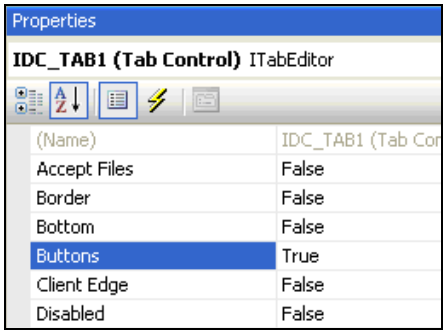


Рис. 8.4. Изменение вида вкладок

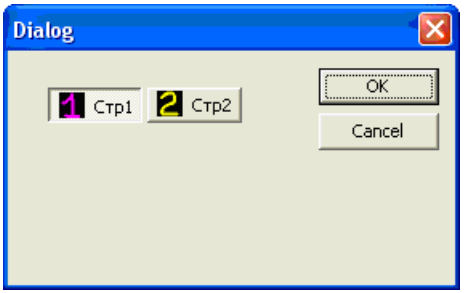


Рис. 8.5. Вкладки в виде кнопок

8.1.2. Работа с заранее подготовленными вкладками

Наиболее удобным способом будет подготовить отдельные страницы вкладок в виде независимых диалоговых окон, а затем объединить их. Подготовим вкладку 1 (новое диалоговое окно), как показано на рис. 8.6 и 8.7:

- в окне **Resource View** вызвать контекстное меню для папки **Dialog** и выполнить команду **Add Resource**;
- в открывшемся окне **Add Resource** выбрать тип ресурса **IDD_PROPPAGE_SMALL [Английский (США)]** и нажать кнопку **New**.

Изменим свойства вкладки (рис. 8.8):

- в окне **Resource View** вызвать контекстное меню для папки `IDD_PROPPAGE_SMALL` и выполнить команду **Properties**;

- в окне свойств задать значения полей:
  - **ID** — `IDD_PROPPAGE_SMALL1`;
  - **Language** — `Русский`.

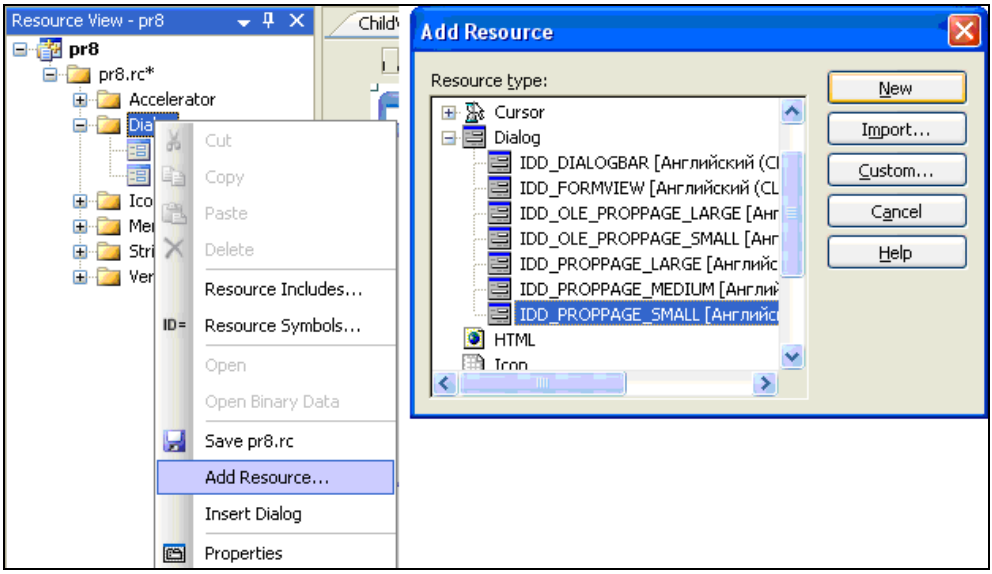


Рис. 8.6. Добавление новой страницы вкладки

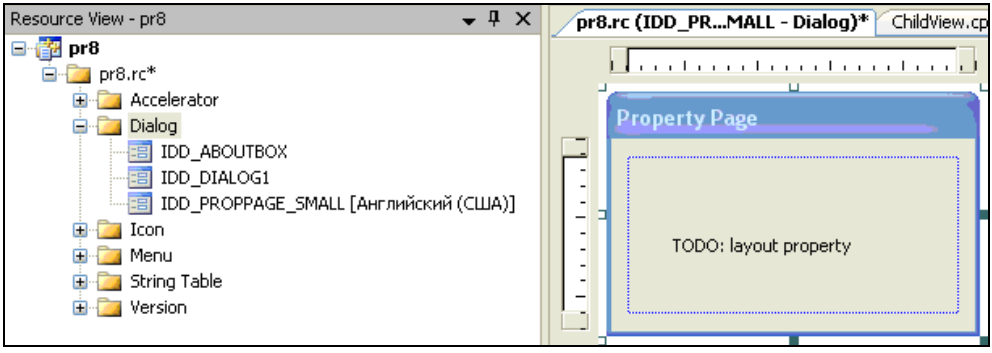


Рис. 8.7. Вид новой страницы вкладки 1

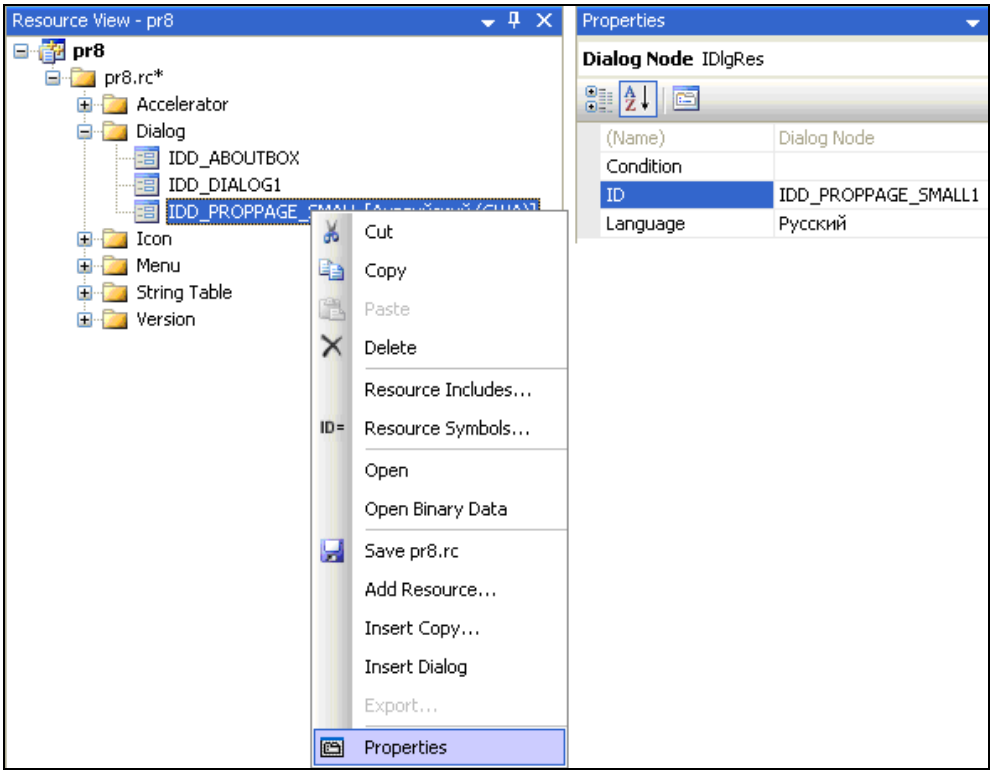


Рис. 8.8. Изменения идентификатора и языка в свойствах вкладки

Изменим вид страницы вкладки.

Для этого нужно:

- вызвать контекстное меню для окна страницы вкладки и выполнить команду **Properties**;
- в окне свойств задать в поле **Caption** текст `Стр1` (рис. 8.9);
- для статического текста в окне вкладки вызвать контекстное меню и выполнить команду **Delete** (рис. 8.10).

Для работы с окном страницы вкладки надо добавить класс для `CMyPage1`, производный от `CPropertyPage`:

- для окна страницы вкладки вызвать контекстное меню и выполнить команду **Add Class** (рис. 8.11);

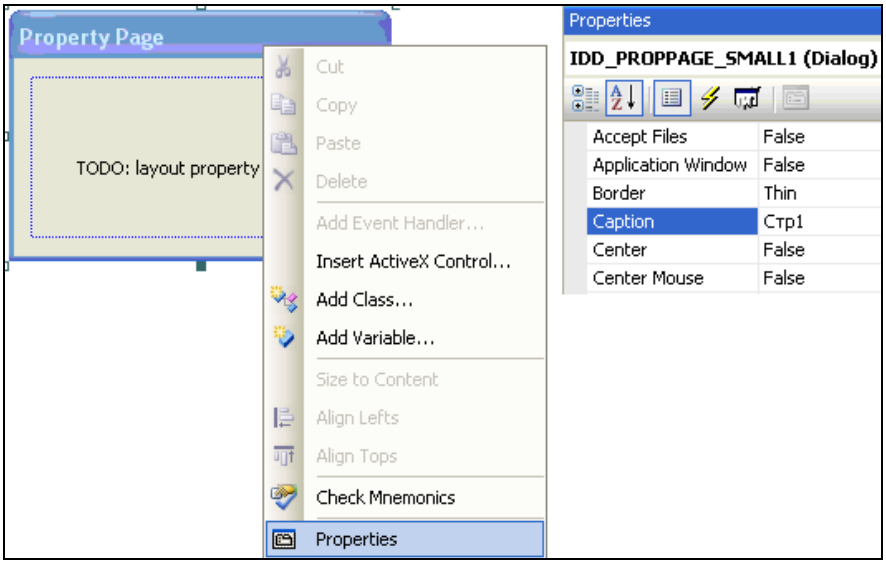


Рис. 8.9. Изменения заголовка страницы вкладки

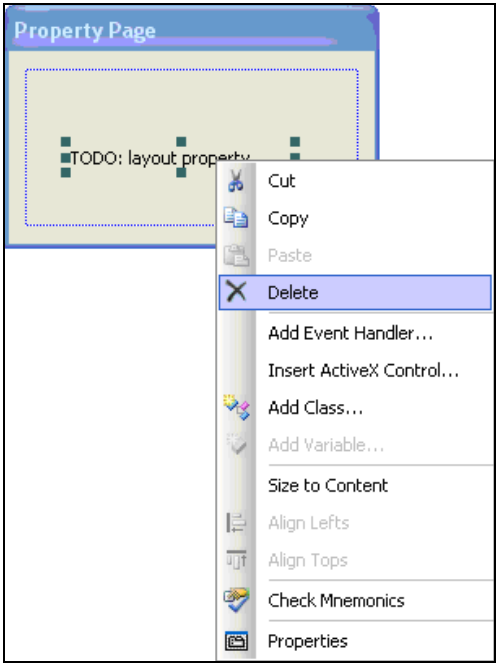


Рис. 8.10. Удаление текста со страницы вкладки

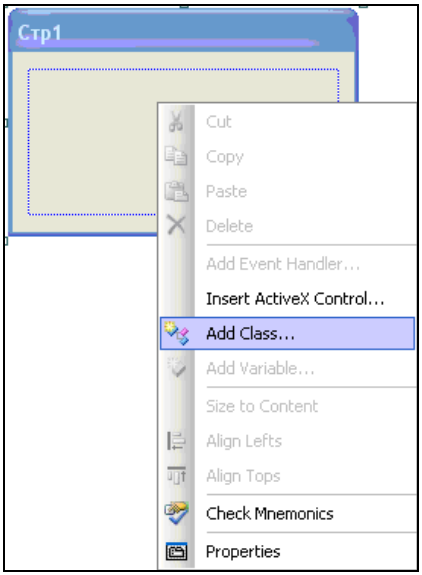


Рис. 8.11. Добавление класса для работы со страницей вкладки

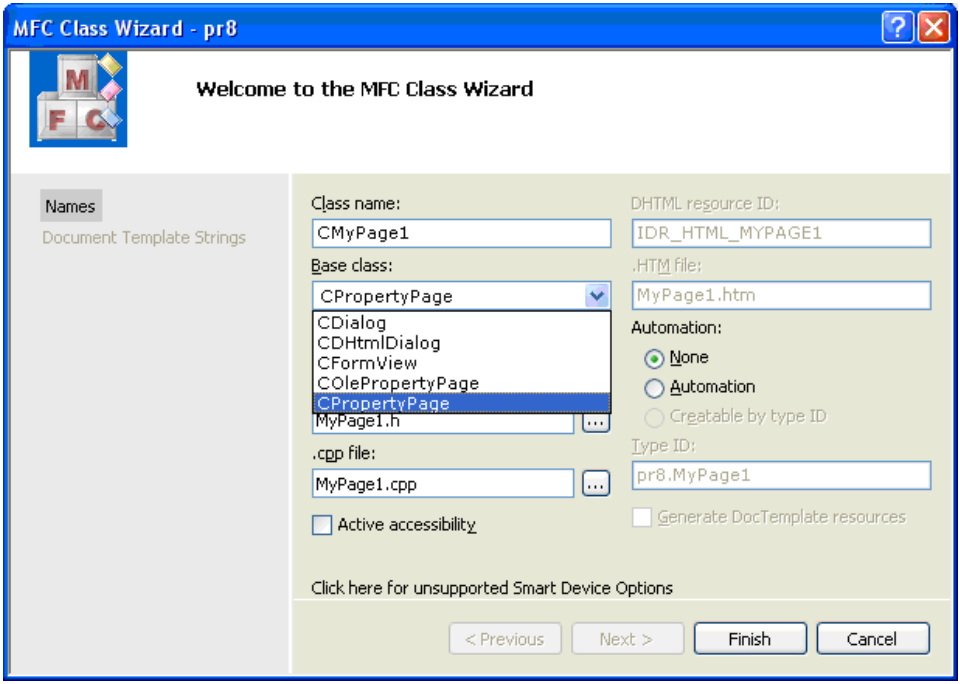


Рис. 8.12. Выбор свойств класса для работы со страницей вкладки

- в открывшемся окне **MFC Class Wizard** заполнить (рис. 8.12):
  - в поле **Class name** — `CMyPage1`;
  - в списке **Base class** — `CPropertyPage`;
  - в поле **.h file** — MyPage1.h (оставить по умолчанию);
  - в поле **.cpp file** — MyPage1.cpp (по умолчанию);
- нажать кнопку **Finish**.

Аналогичным образом нужно подготовить вкладку 2 (рис. 8.13) и класс `CMyPage2`. Изменения в файлах приведены в листингах 8.9—8.13.

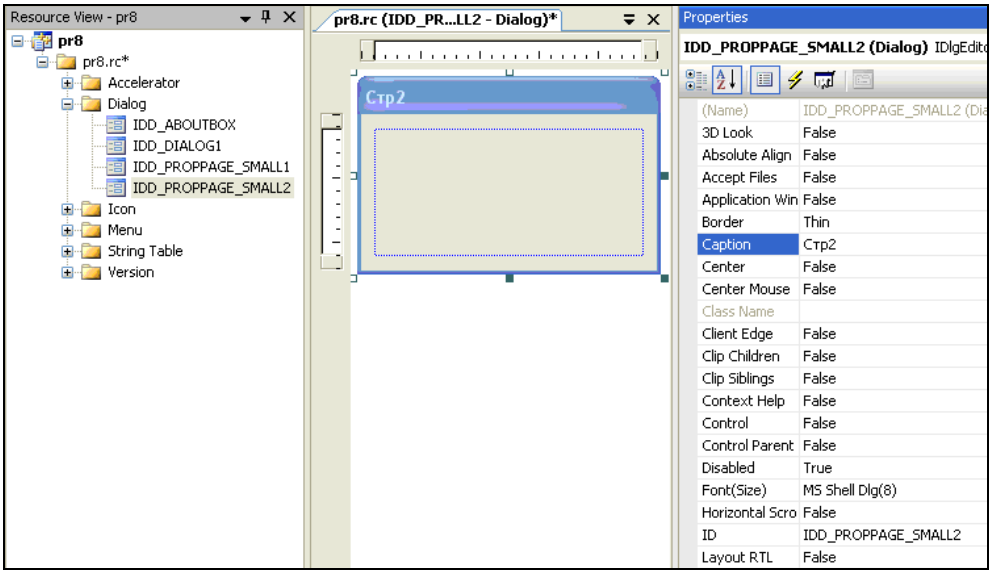


Рис. 8.13. Подготовка второй страницы вкладки

Листинг 8.9. Изменения в файле Resource.h для работы со вкладками

```
// ...
#define IDD_PROPPAGE_SMALL1             133
#define IDD_PROPPAGE_SMALL2             102
// ...
```

Листинг 8.10. Файл CMyPage1.h для работы с вкладкой 1

```
#pragma once

// CMyPage1 dialog
class CMyPage1 : public CPropertyPage
{
   DECLARE_DYNAMIC(CMyPage1)

public:
   CMyPage1();
   virtual ~CMyPage1();

// Dialog Data
   enum { IDD = IDD_PROPPAGE_SMALL1 };

protected:
   virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support

   DECLARE_MESSAGE_MAP()
};
```

Листинг 8.11. Файл MyPage1.cpp для работы с вкладкой 1

```
//
#include "stdafx.h"
#include "pr8.h"
#include "MyPage1.h"

// CMyPage1 dialog
IMPLEMENT_DYNAMIC(CMyPage1, CPropertyPage)

CMyPage1::CMyPage1() : CPropertyPage(CMyPage1::IDD)
{
}
```

```
CMyPage1::~CMyPage1()
{
}

void CMyPage1::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CMyPage1, CPropertyPage)
END_MESSAGE_MAP()

// CMyPage1 message handlers
```

Листинг 8.12. Файл CMyPage2.h для работы с вкладкой 2

```
#pragma once

// CMyPage2 dialog
class CMyPage2 : public CPropertyPage
{
    DECLARE_DYNAMIC(CMyPage2)

public:
    CMyPage2();
    virtual ~CMyPage2();

// Dialog Data
    enum { IDD = IDD_PROPPAGE_SMALL2 };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

    DECLARE_MESSAGE_MAP()
};
```

Листинг 8.13. Файл MyPage2.cpp для работы с вкладкой 2

```
//
#include "stdafx.h"
#include "pr8.h"
#include "MyPage2.h"

// CMyPage2 dialog
IMPLEMENT_DYNAMIC(CMyPage2, CPropertyPage)

CMyPage2::CMyPage2() : CPropertyPage(CMyPage2::IDD)
{
}

CMyPage2::~CMyPage2()
{
}

void CMyPage2::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CMyPage2, CPropertyPage)
END_MESSAGE_MAP()

// CMyPage2 message handlers
```

Класс для индивидуальных страниц свойств определен как:

```
class CPropertyPage : public CDialog
```

Добавим код для работы со страницами вкладок. Изменения в файлах приведены в листингах 8.14 и 8.15.

Листинг 8.14. Изменения в файле MyDlg.h для подключения вкладок

```
// ...
#include "MyPage1.h"
#include "MyPage2.h"

class MyDlg : public CDialog
{
// ...
public:
   CPropertySheet sheet;      // Класс страницы свойств
   CMyPage1 page1;            // Класс первой вкладки-диалога
   CMyPage2 page2;            // Класс второй вкладки-диалога
};
```

Листинг 8.15. Изменения в файле MyDlg.cpp для подключения вкладок

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Добавление страниц-диалогов
   sheet.AddPage(&page1);
   sheet.AddPage(&page2);
   // Создание страницы свойст
   sheet.Create(this, WS_CHILD | WS_VISIBLE, WS_EX_CONTROLPARENT);
   // Положение относительно родительского окна диалога (IDD_DIALOG1)
   sheet.SetWindowPos(NULL, 10,65,0,0, SWP_NOZORDER | SWP_NOSIZE |
                      SWP_NOACTIVATE);
}
```

Класс представления страниц свойств определен как:

```
class CPropertySheet : public CWnd
```

Добавление новой страницы в конец списка вкладок выполняется с помощью функции:

```
void CPropertySheet::AddPage(
   CPropertyPage *pPage);      // Указатель на класс добавляемой страницы
```

Создание окна со списком вкладок выполняется функцией:

```
virtual BOOL CPropertySheet::Create(       // 0 - ошибка
   CWnd* pParentWnd = NULL,          // Указатель на родительское окно
   DWORD dwStyle = (DWORD)-1,        // Стиль(и) окна
   DWORD dwExStyle = 0);             // Расширенный стиль(и) окна
```

Стили окна `dwStyle` и `dwExStyle` были описаны в *разд. 1.3.2*.

Изменение размеров и позиции окна выполняется фцункцией:

```
BOOL CWnd::SetWindowPos(           // 0 - ошибка
   const CWnd* pWndInsertAfter,    // Указатель на окно, после которого
                                   // будет расположено это окно
   int x,                          // Координаты левого верхнего угла
   int y,
   int cx,                         // Ширина окна
   int cy,                         // Высота окна
   UINT nFlags);                   // Параметры размера и позиции окна
```

Вместо окна в `pWndInsertAfter` можно указать одно из предопределенных значений, задающих расположение окна относительно других окон.

Значения положения окна (`pWndInsertAfter`) могут быть такими:

- `wndBottom` — разместить окно под другими окнами;
- `wndTop` — разместить окно над другими окнами;
- `wndTopMost` — разместить окно над всеми другими окнами, имеющими статус `wndTopMost`;
- `wndNoTopMost` — разместить окно выше всех окон, имеющих статус `wndTopMost`, но ниже окон, имеющих статус `wndNoTopMost`.

Параметры размера и позиции окна (`nFlags`) могут быть следующими:

- `SWP_DRAWFRAME` — задает рамку вокруг окна, определенную в классе окна;
- `SWP_FRAMECHANGED` — посылает сообщение `WM_NCCALCSIZE` в окно, даже если размер окна не изменяется. Если этот флаг не определен, `WM_NCCALCSIZE` посылается, только когда размер окна изменяется;

- SWP_HIDEWINDOW — делает окно невидимым;
- SWP_NOACTIVATE — блокирует активизацию окна;
- SWP_NOMOVE — сохранение текущей позиции (параметры x, y игнорируются);
- SWP_NOREDRAW — запрещает перерисовывать окно;
- SWP_NOSENDCHANGING — не дает окну получать сообщение WM_WINDOWPOSCHANGING;
- SWP_NOSIZE — сохранение текущих размеров (параметры cx, cy игнорируются);
- SWP_NOZORDER — сохранение текущего расположения окна относительно других окон (параметр pWndInsertAfter игнорируется);
- SWP_SHOWWINDOW — отображает окно.

Можно изменять названия закладок и добавлять к ним иконки *(см. разд. 8.1.1)*. Изменим название первой вкладки на "123" и добавим к нему первую иконку из списка изображений. Изменения в файле приведены в листинге 8.16.

Листинг 8.16. Изменения в файле MyDlg.cpp для задания нового заголовка вкладки

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Изменения
   // Получить указатель на список страницы
   CTabCtrl *pt = sheet.GetTabControl();
   // Связать список страниц со списком изображений
   pt->SetImageList(&image);
   TCITEM st;
   // Получить текущие свойства первой страницы (с индексом 0)
   st.mask = TCIF_TEXT | TCIF_IMAGE;
   pt->GetItem(0, &st);
```

```
    // Задать новый заголовок и иконку
    st.mask = TCIF_TEXT | TCIF_IMAGE;
    st.pszText = "123";
    st.cchTextMax = 4;
    st.iImage = 0;
    // Изменить свойства по заданным значениям
    pt->SetItem(0,&st);
}
```

Получить указатель на список страниц можно функцией:

```
CTabCtrl* CPropertySheet::GetTabControl() const;
```

Добавим обработку выбранной страницы. Изменения в файле приведены в листинге 8.17.

Листинг 8.17. Изменения в файле MyDlg.cpp для обработки выбранной вкладки при закрытии окна диалога

```
// ...
void MyDlg::OnBnClickedOk()
{
    // ...
    // CPropertySheet
    // Получить индекс текущей диалоговой вкладки
    CTabCtrl *pt = sheet.GetTabControl();
    a = pt->GetCurSel();
    s.Format("Диалог-вкладка %d", a+1);
    AfxMessageBox(s);

    OnOK();
}
```

В диалоговые окна страниц (`DD_PROPPAGE_SMALL1`, `IDD_PROPPAGE_SMALL2`) теперь легко можно добавлять элементы управления. Здесь в окно страницы 1 была добавлена кнопка, а в окно страницы 2 — переключатель. Изменения в файлах приведены в листингах 8.18 и 8.19.

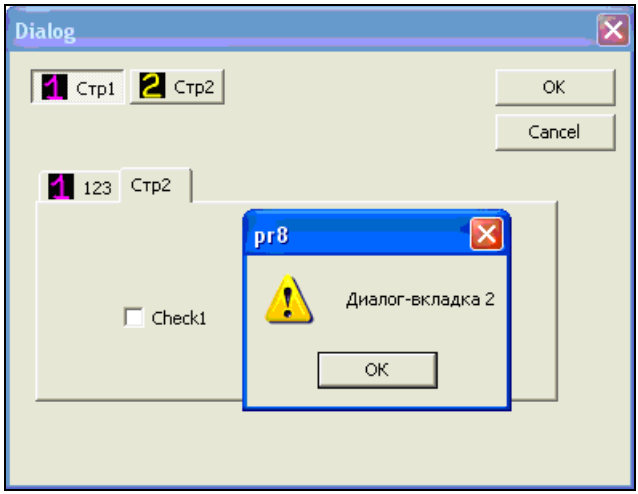
Листинг 8.18. Изменения в файле Resource.h для обработки состояния управляющих элементов на вкладках

```
// ...
#define IDC_BUTTON1                     1001
#define IDC_CHECK1                      1002
// ...
```

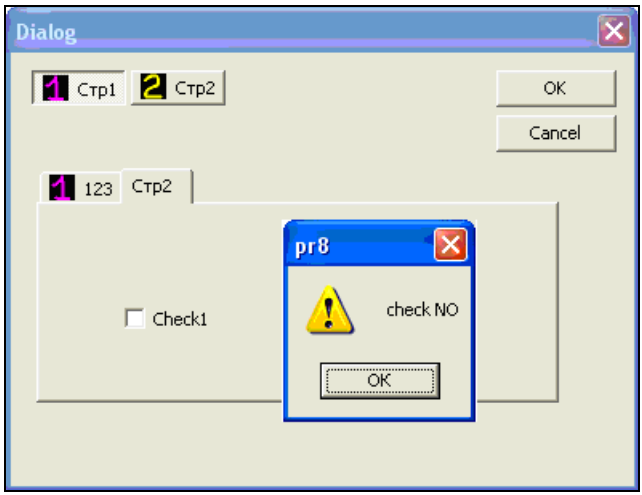
Листинг 8.19. Изменения в файле MyDlg.cpp для обработки состояния управляющих элементов на вкладках

```
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // CPropertySheet
   // Получить индекс текущей диалоговой вкладки
   CTabCtrl *pt = sheet.GetTabControl();
   a = pt->GetCurSel();
   s.Format("Диалог-вкладка %d", a+1);
   AfxMessageBox(s);
   // Узнать состояние переключателя check вкладки 2
   CButton *pc = (CButton *)page2.GetDlgItem(IDC_CHECK1);
   if(pc->GetCheck())
      AfxMessageBox("check YES");
   else
      AfxMessageBox("check NO");

   OnOK();
}
```

Результат работы программы показан на рис. 8.14.



а

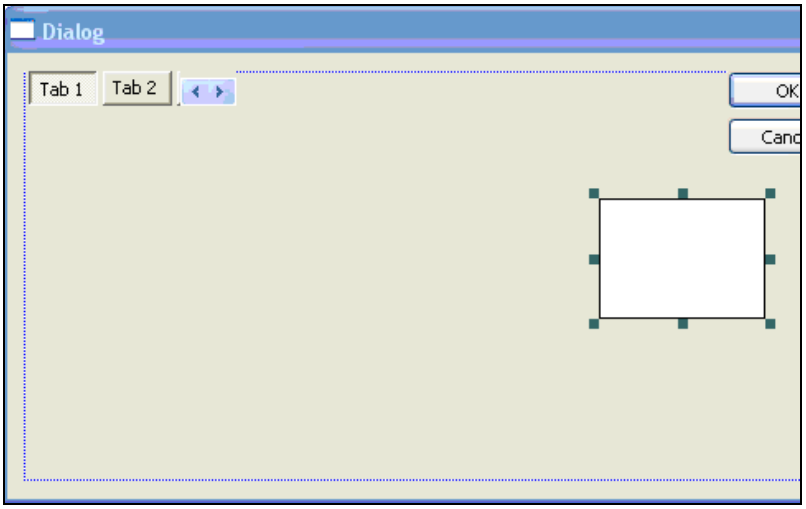


б

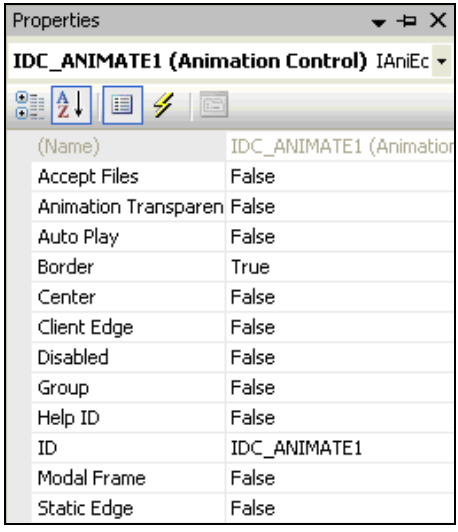
Рис. 8.14. Работа с окнами-вкладками: обработка выбранной вкладки (*а*) и состояния переключателя на ней (*б*)

8.1.3. Анимация (*Animation Control*)

Добавим анимацию **Animation Control** (рис. 8.15) и переменную (категории **Control**) `anim1` (рис. 8.16) для работы с ней. Изменения в файлах приведены в листингах 8.20—8.22.



а



б

Рис. 8.15. Добавление анимации в окно диалога (*а*) и просмотр ее свойств (*б*)

Листинг 8.20. Изменения в файле Resource.h для работы с анимацией

```
// ...
#define IDC_ANIMATE1                    1003
// ...
```

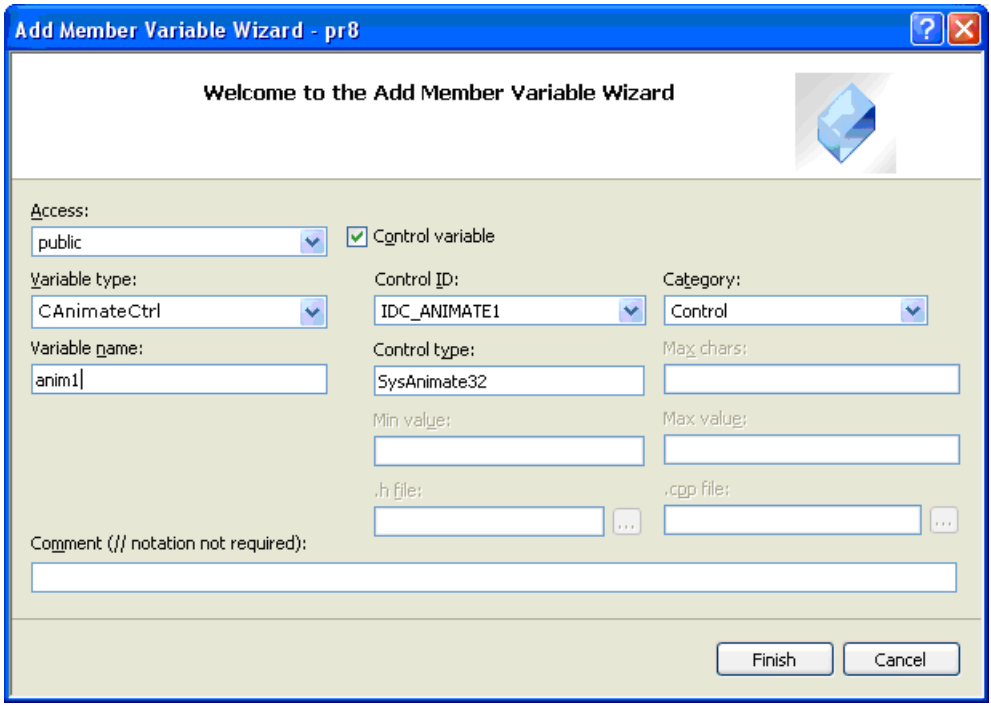


Рис. 8.16. Добавление переменной для работы с анимацией

Листинг 8.21. Изменения в файле MyDlg.h для работы с анимацией

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CAnimateCtrl anim1;                    // Для работы с анимацией
};
```

Листинг 8.22. Изменения в файле MyDlg.cpp для работы с анимацией

```
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
   // ...
   DDX_Control(pDX, IDC_ANIMATE1, anim1);
}
```

Класс `CAnimateCtrl` предназначен для работы с видеоклипами в формате AVI (Audio Video Interleaved — стандартный формат Windows). В каталоге проекта pr8\res на диске был скопирован файл с видеоклипом dillo.avi. Для работы с файлом надо в программе выполнить его открытие. Изменения в файле приведены в листинге 8.23.

Листинг 8.23. Изменения в файле MyDlg.cpp для открытия файла с видеоклипом

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // animal
   anim1.Open("res\\dillo.avi");
}
```

Открытие файла с видеоклипом выполняется функцией:

```
BOOL CAnimateCtrl::Open(          // 0 - ошибка
   LPCTSTR lpszFileName);         // Имя файла
```

В данной программе файл dillo.avi находится в каталоге pr8\res. Если запускать программу из среды Microsoft Visual Studio 2005, файл будет найден. Если запускать программу из командной строки Windows, то надо изменить в функции `Open()` путь к файлу. Если файл открывается в программе, то надо его потом и закрыть. Лучше всего это сделать при закрытии (разрушении) окна диалога. Для этого в класс диалога надо добавить функцию `DestroyWindow()` (рис. 8.17):

- в окне **Class View** вызвать контекстное меню для класса `MyDlg` и выполнить команду **Properties**;

- в окне свойств **Properties** выбрать вкладку **Overrides**;
- в списке функций найти `DestroyWindow` и выбрать **<Add> DestroyWindow**.

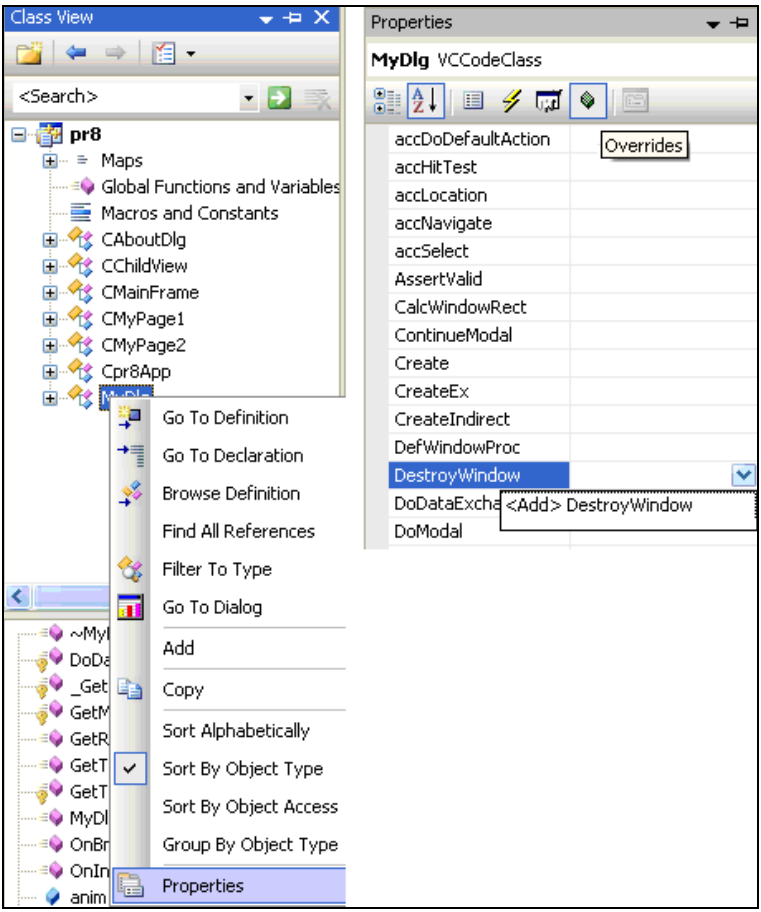


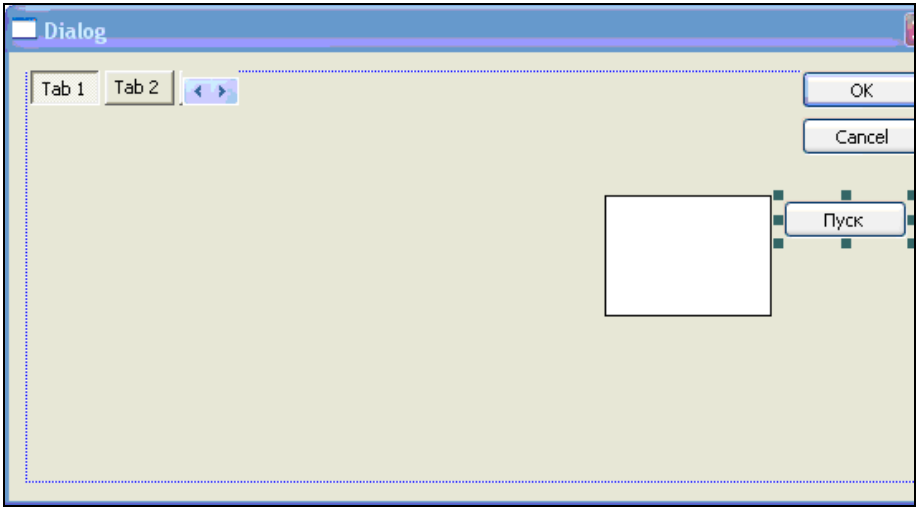
Рис. 8.17. Добавление в класс диалога функции разрушения окна

Изменения в файлах приведены в листингах 8.24 и 8.25.

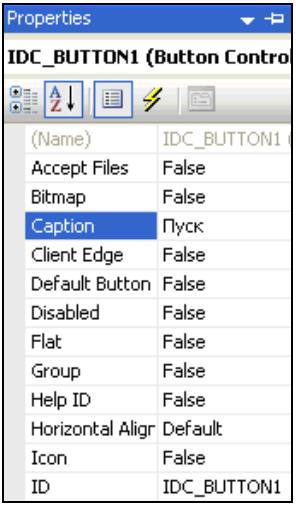
Листинг 8.24. Изменения в файле MyDlg.h для обработки сообщения о разрушении окна диалога

```
// ...
class MyDlg : public CDialog
```

```
{
// ...
public:
   virtual BOOL DestroyWindow();
};
```

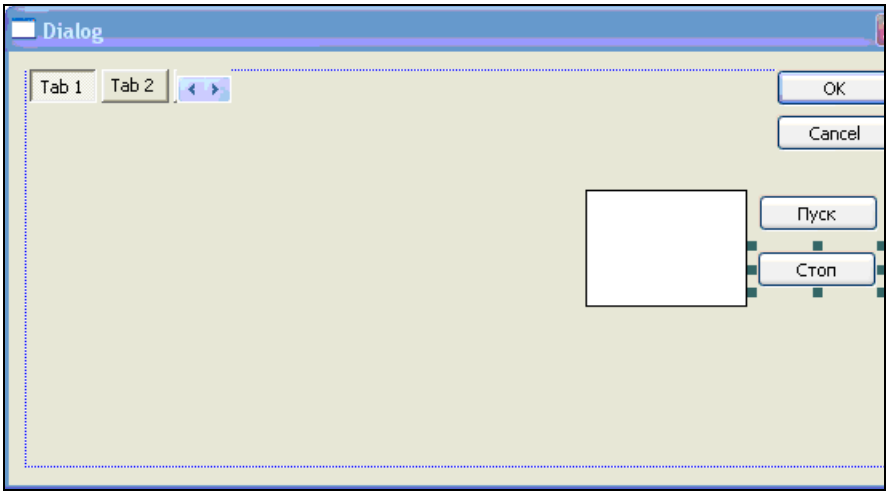


а

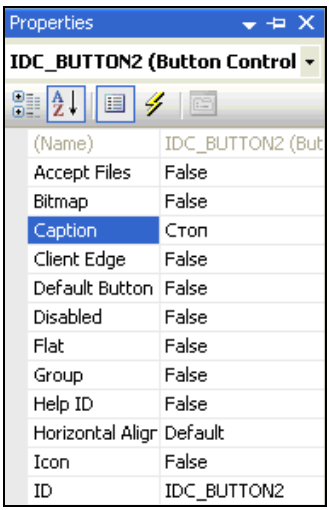


б

Рис. 8.18. Кнопка запуска видеоролика (*а*) и ее свойства (*б*)



в



г

Рис. 8.18. Кнопка остановки видеоролика (*в*) и ее свойства (*г*)

Листинг 8.25. Изменения в файле MyDlg.cpp для обработки сообщения о разрушении окна диалога

```
// ...
BOOL MyDlg::DestroyWindow()
{
    // TODO: Add your specialized code here and/or call the base class
```

```
    anim1.Close();
    return CDialog::DestroyWindow();
}
```

Функция разрушения окна определена как:

```
virtual BOOL CWnd::DestroyWindow();          // 0 - ошибка
```

Закрытие файла с видеоклипом выполняется функцией:

```
BOOL CAnimateCtrl::Close();                  // 0 - ошибка
```

Если надо, чтобы видеоклип автоматически начинал проигрываться при создании окна диалога, нужно задать в окне свойств элемента анимации в поле **Auto Play** (Автоматическое проигрывание) значение `True`. Сделаем так, чтобы ролик начинал проигрываться по кнопке **Пуск** (`IDC_BUTTON1`) и останавливался по кнопке **Стоп** (`IDC_BUTTON2`). Добавим соответствующие кнопки (рис. 8.18) и обработку их нажатия (дважды щелкнув левой кнопкой мыши по нужной кнопке).

Изменения в файлах приведены в листингах 8.26—8.28.

Листинг 8.26. Изменения в файле Resource.h для запуска и остановки видеоролика

```
// ...
#define IDC_BUTTON1                          1001
#define IDC_BUTTON2                          1004
// ...
```

Листинг 8.27. Изменения в файле MyDlg.h для запуска и остановки видеоролика

```
// ...
class MyDlg : public CDialog
{
// ...
public:
    afx_msg void OnBnClickedButton1();
public:
    afx_msg void OnBnClickedButton2();
};
```

Листинг 8.28. Изменения в файле MyDlg.cpp для запуска и остановки видеоролика

```
// ...
BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   // ...
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
   ON_BN_CLICKED(IDC_BUTTON2, &MyDlg::OnBnClickedButton2)
END_MESSAGE_MAP()
// ...
void MyDlg::OnBnClickedButton1()
{
   // TODO: Add your control notification handler code here
   anim1.Play(0, -1, -1);
}
void MyDlg::OnBnClickedButton2()
{
   // TODO: Add your control notification handler code here
   anim1.Stop();
}
```

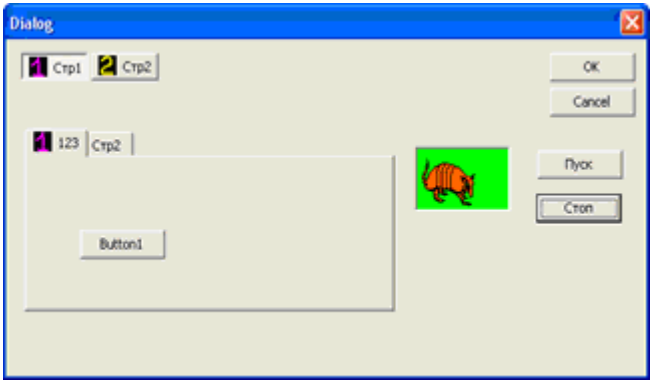


Рис. 8.19. Работа с анимацией

Проигрывание видеоролика выполняется функцией:

```
BOOL CAnimateCtrl::Play(          // 0 - ошибка
   UINT nFrom,                    // Начальный кадр
```

```
    UINT nTo,                          // Конечный кадр
    UINT nRe);                         // Число проигрываний
```

Значения `nFrom` и `nTo` лежат в пределах от 0 до 65535. Если `nTo = -1`, то проигрывание будет до конца клипа (т. е. не обязательно знать полное количество кадров). Если `nRep = -1`, то проигрывание будет бесконечным.

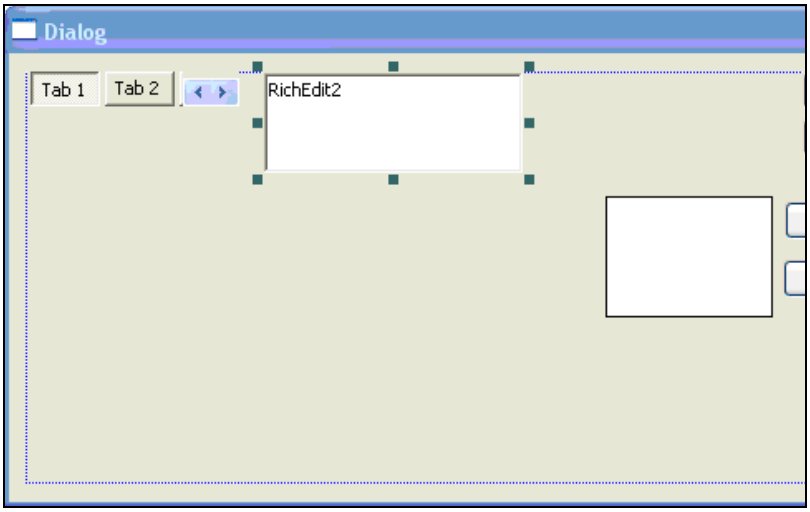
Остановка видеоролика выполняется с помощью функции:

```
BOOL CAnimateCtrl::Stop();             // 0 - ошибка
```

Результат работы программы показан на рис. 8.19.

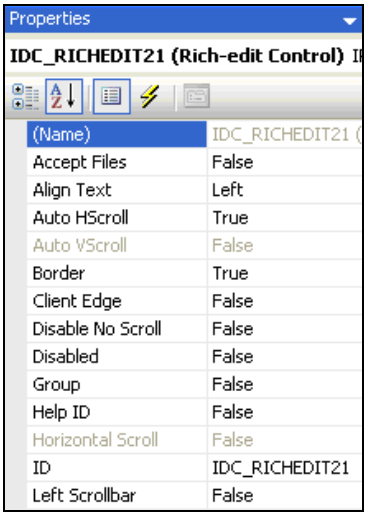
8.1.4. Расширенный редактор (*Rich Edit 2.0 Control*)

Добавим расширенный редактор **Rich Edit 2.0 Control** (рис. 8.20). Для работы с окном редактирования добавим переменную (категории **Control**) `redit` (рис. 8.21, *а*). При нажатии кнопки **Finish** появится сообщение о необходимости вызова в программе функции `AfxInitRichEdit2()` для работы с редактором (рис. 8.21, *б*). Для хранения содержимого окна редактирования добавим переменную (категории **Value**) `redit_val` (рис. 8.22). Изменения в файлах приведены в листингах 8.29—8.31.



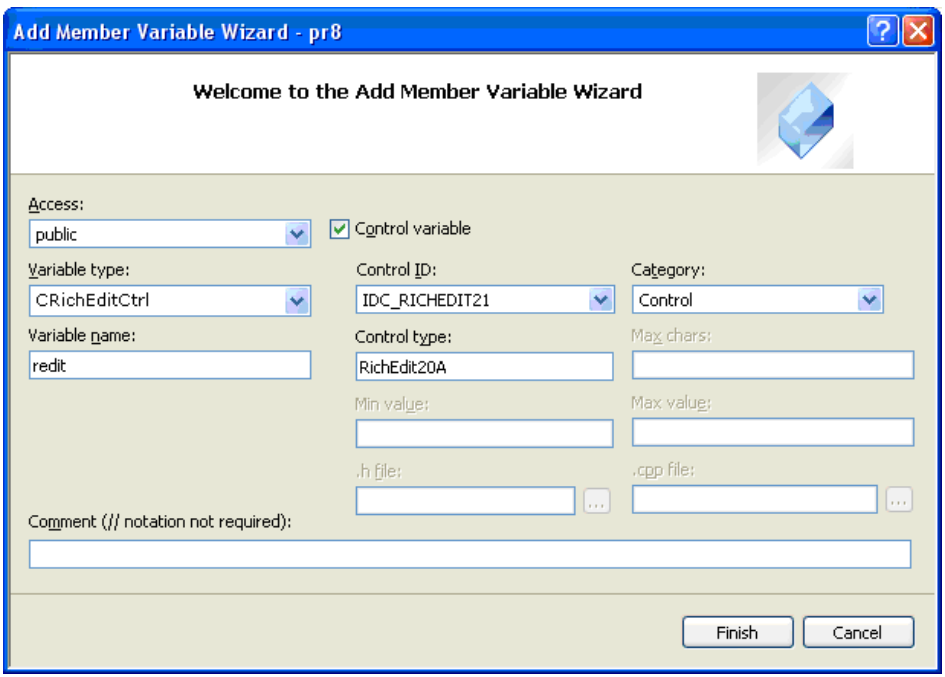
а

Рис. 8.20. Добавление расширенного редактора в окно диалога (*а*)



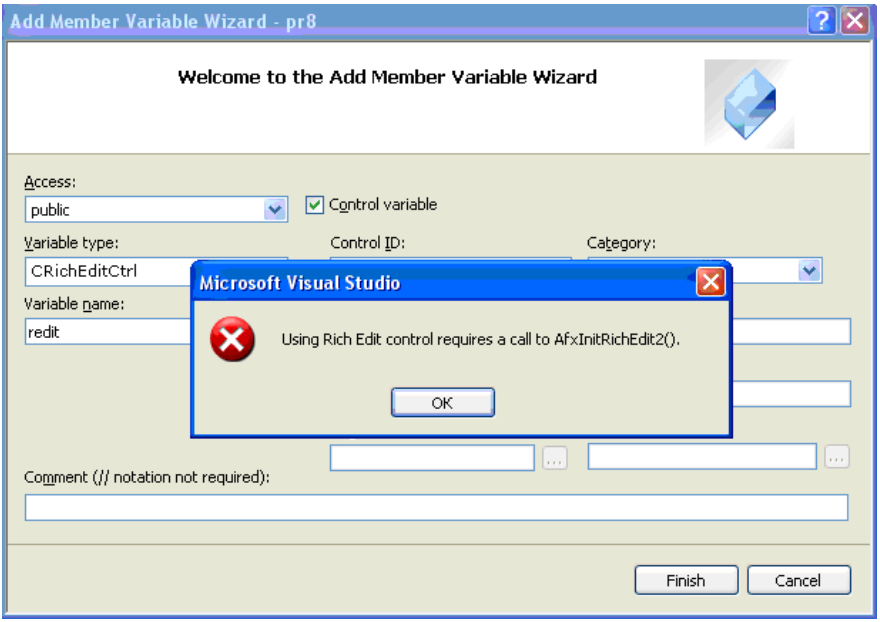
б

Рис. 8.20. Просмотр свойств расширенного редактора (*б*)



а

Рис. 8.21. Добавление переменной
для работы с расширенным редактором (*а*)



б

Рис. 8.21. Сообщение о необходимости вызова функции `AfxInitRichEdit2()` для работы с редактором (*б*)

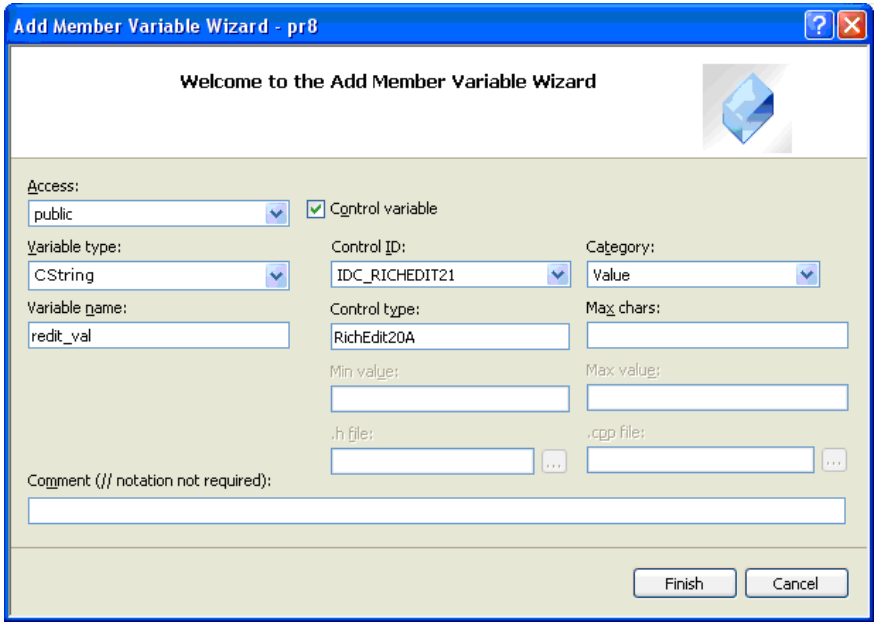


Рис. 8.22. Переменная для хранения значения расширенного редактора

Листинг 8.29. Изменения в файле Resource.h для работы с расширенным редактором

```
// ...
#define IDC_RICHEDIT21                  1005
// ...
```

Листинг 8.30. Изменения в файле MyDlg.h для работы с расширенным редактором

```
// ...
class MyDlg : public CDialog
{
// ...
public:
   CRichEditCtrl redit;          // Контроль окна редактора
public:
   CString redit_val;            // Содержимое окна редактора
};
```

Листинг 8.31. Изменения в файле MyDlg.cpp для работы с расширенным редактором

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , redit_val(_T(""))
{
}
// ...
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_RICHEDIT21, redit);
   DDX_Text(pDX, IDC_RICHEDIT21, redit_val);
}
```

Для работы с расширенным окном редактирования надо вызвать функцию `AfxInitRichEdit2()` (см. сообщение на рис. 8.21, *б*). Если этого не сделать, то диалоговое окно вообще не будет появляться. Добавим вручную вызов этой функции (в классе приложения для этого мастером даже было обозначено место). Изменения в файле приведены в листинге 8.32.

Листинг 8.32. Изменения в файле pr8.cpp: вызов функции, обязательной для работы с расширенным редактором

```
// ...
BOOL Cpr8App::InitInstance()
{
//TODO: call AfxInitRichEdit2() to initialize richedit2 library.
   AfxInitRichEdit2();
   //...
}
```

Функция инициализации расширенного редактора определена как:

```
BOOL AFXAPI AfxInitRichEdit2();                    // 0 - ошибка
```

Работа с расширенным редактором аналогична работе с `CEdit`. В этой программе обработка всех элементов управления производится до закрытия окна диалога, поэтому нельзя использовать переменную `redit_val`, т. к. она получит свое значение только после закрытия диалога (после завершения работы функции `OnOK()`). Поэтому здесь доступ к содержимому редактора осуществляется через переменную `redit`.

> ***Примечание***
>
> При работе в окне редактора, для перехода на другую строчку надо нажимать комбинацию клавиш <Ctrl>+<Enter>.

Изменения в файле приведены в листинге 8.33.

Листинг 8.33. Изменения в файле MyDlg.cpp для анализа содержимого редактора при закрытии окна диалога

```
// ...
void MyDlg::OnBnClickedOk()
{
```

```
    // ...
    // CRichEditCtrl
    if(!redit.GetTextLength())
       AfxMessageBox("Текст не был введен");
    else
    {
       redit.GetWindowTextA(s);
       AfxMessageBox(s);
    }

    OnOK();
}
```

Функции получения длины текста в окне редактирования определены как:

```
long CRichEditCtrl::GetTextLength() const;
int CWnd::GetWindowTextLength() const;
```

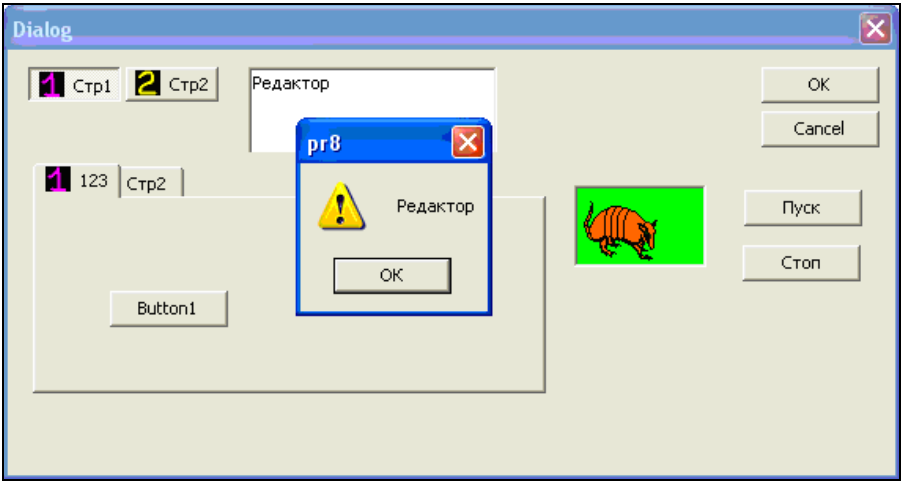


Рис. 8.23. Работа с расширенным редактором

Функция получения содержимого окна редактирования (текста):

```
int CWnd::GetWindowText(
   LPTSTR lpszStringBuf,          // Строка для получения содержимого
```

```
   int nMaxCount) const;          // Максимальное количество символов
                                  // для lpszStringBuf
```

или

```
void CWnd::GetWindowText(
   CString& rString) const;       // Строка для получения содержимого
```

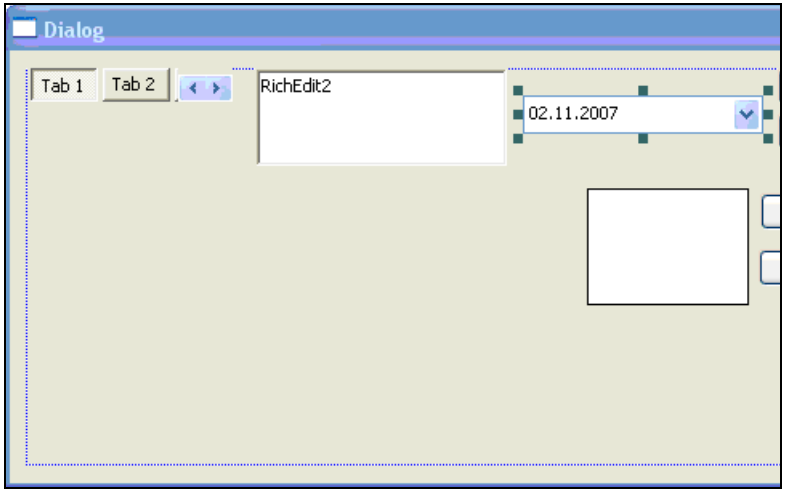
Функция для занесения текста в окно редактирования выглядит так:

```
void CWnd::SetWindowText(
   LPCTSTR lpszString);           // Строка, которая будет выведена
                                  // в окне редактирования
```

Результаты работы программы показаны на рис. 8.23.

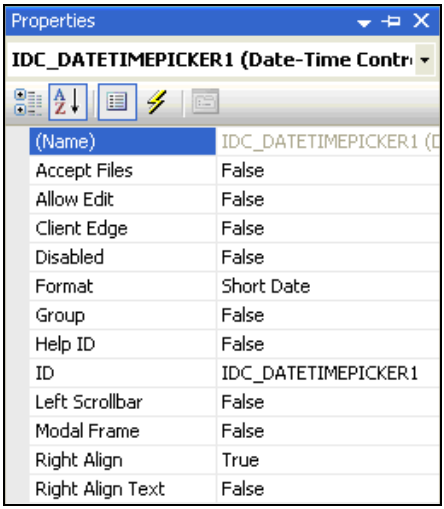
8.1.5. Дата и время (*Date Time Picker*)

Добавим элемент управления даты и времени **Date Time Picker** (рис. 8.24), переменную `datetime` (категории **Control**) для работы с датой и временем (рис. 8.25) и переменную `datetime_val` (категории **Value**) для сохранения значений даты и времени (рис. 8.26). Изменения в файлах приведены в листингах 8.34—8.36.



а

Рис. 8.24. Добавление элемента с датой и временем (*а*)



б

Рис. 8.24. Свойства элемента с датой и временем (*б*)

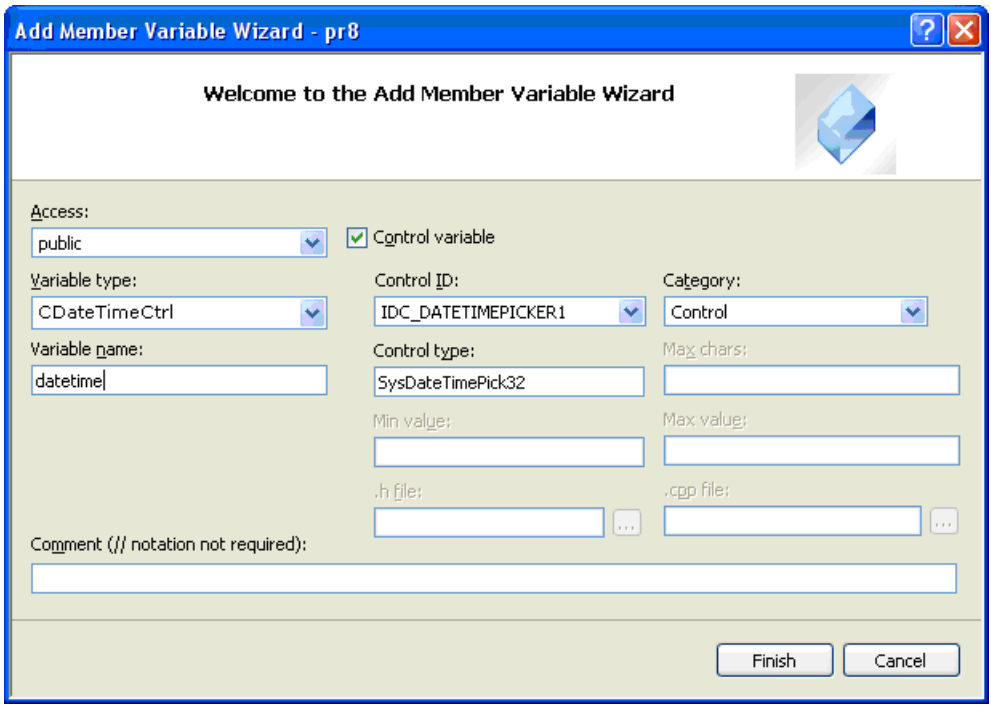


Рис. 8.25. Переменная для работы с датой и временем

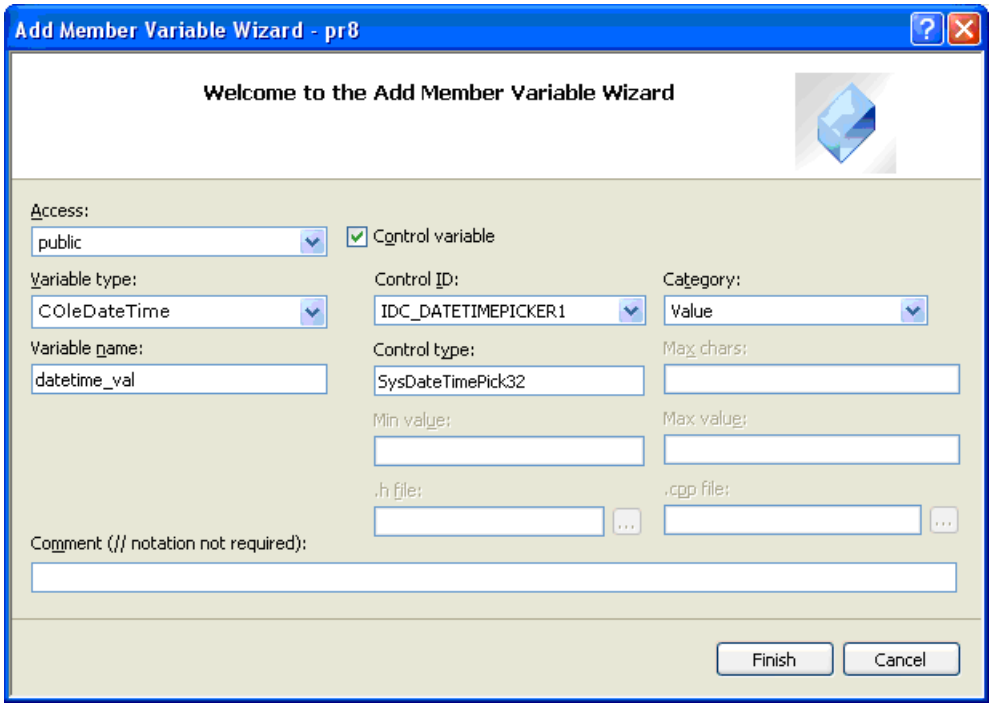


Рис. 8.26. Переменная для сохранения значения даты и времени

Листинг 8.34. Изменения в файле Resource.h для работы с датой и временем

```
// ...
#define IDC_DATETIMEPICKER1             1006
// ...
```

Листинг 8.35. Изменения в файле MyDlg.h для работы с датой и временем

```
// ...
class MyDlg : public CDialog
{
// ...
public:                         // Для работы с датой и временем
   CDateTimeCtrl datetime;
```

```
public:
   COleDateTime datetime_val;
};
```

Листинг 8.36. Изменения в файле MyDlg.cpp для работы с датой и временем

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , redit_val(_T(""))
   , datetime_val(COleDateTime::GetCurrentTime())
{
}
void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_DATETIMEPICKER1, datetime);
   DDX_DateTimeCtrl(pDX, IDC_DATETIMEPICKER1, datetime_val);
}
```

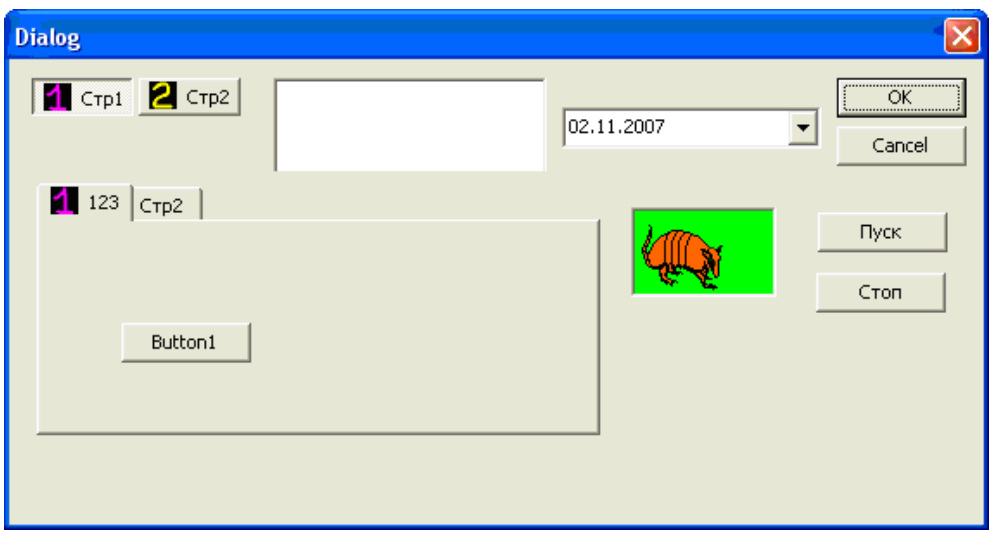


Рис. 8.27. Работа с датой и временем

Функция получения текущего времени выглядит так:

```
static COleDateTime WINAPI COleDateTime::GetCurrentTime() throw();
```

Использовать ее можно, например, так:

```
if(COleDateTime::GetCurrentTime().GetDayOfWeek() == 6)
   AfxMessageBox("Сегодня пятница");
```

Здесь день недели 6 — это пятница, т. к. у англичан неделя начинается с воскресенья (1), понедельник (2) и т. д.

Результат работы программы показан на рис. 8.27. Поле даты и времени показывает только дату. Чтобы в нем выводилось и время, нужно изменить формат. Изменения в файле приведены в листинге 8.37.

Листинг 8.37. Изменения в файле MyDlg.cpp для задания формата даты и времени

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Формат для отображения даты и времени
   datetime.SetFormat("H:mm дата: d/M/yyyy");

   return TRUE;  // return TRUE unless you set the focus to a control
   // EXCEPTION: OCX Property Pages should return FALSE
}
```

Функция установки формата даты и времени определена как:

```
BOOL CDateTimeCtrl::SetFormat(        // 0 - ошибка
   LPCTSTR pstrFormat);               // Строка формата
```

Строка формата может содержать как текст, так и специальные символы. Специальные символы могут быть такими:

- "d" — для дня используется одна или две цифры;
- "dd" — для дня всегда используются две цифры (для чисел от 1 до 9 слева добавляется 0);
- "ddd" — отображается сокращенное обозначение дня недели;

- "dddd" — отображается полное обозначение дня недели;
- "h" — для часов используется одна или две цифры в 12-часовом формате (AM/PM);
- "hh" — для часов всегда используются две цифры в 12-часовом формате (для чисел от 1 до 9 слева добавляется 0);
- "H" — для часов используется одна или две цифры в 24-часовом формате;
- "HH" — для часов всегда используются две цифры в 24-часовом формате (для чисел от 1 до 9 слева добавляется 0);
- "m" — для минут используется одна или две цифры;
- "mm" — для минут всегда используются две цифры (для чисел от 1 до 9 слева добавляется 0);
- "M" — для месяца используется одна или две цифры;
- "MM" — для месяца всегда используются две цифры (для чисел от 1 до 9 слева добавляется 0);
- "t" — для отображения режима AM или PM используется одна буква ("A" или "P");
- "tt" — для отображения режима AM или PM используются две буквы ("AM" или "PM");
- "yy" — для года используются только две последние цифры;
- "yyy" — полное обозначение года (четыре цифры).

Добавим обработку значений даты и времени для двух случаев — до закрытия окна диалога (в файле MyDlg.cpp) и после закрытия (в файле ChildView.cpp) окна диалога. Изменения в файлах приведены в листингах 8.38 и 8.39.

Листинг 8.38. Изменения в файле MyDlg.cpp для обработки значений даты и времени до закрытия окна диалога

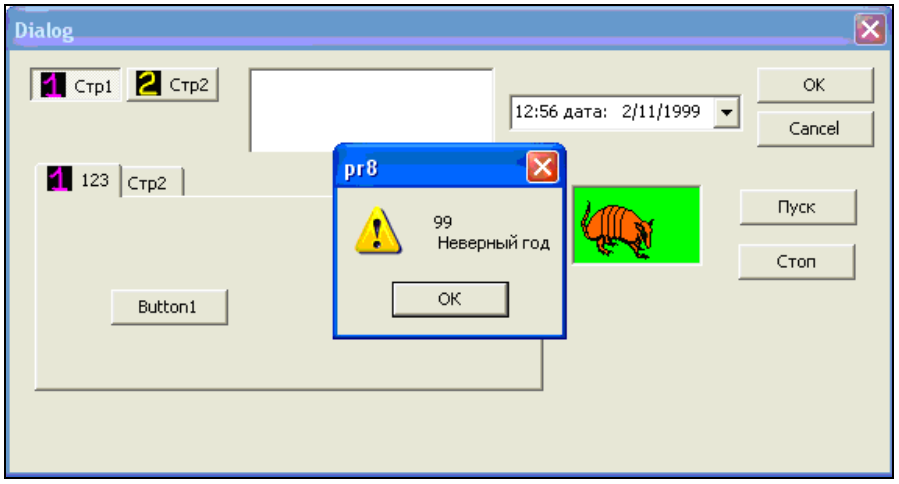
```
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // Дата и время
```

```
   CTime time;
   datetime.GetTime(time);    // Получить значения даты и времени
   s = time.Format("%y");     // Взять из них только последние две цифры
                              // года
   a = atoi(s.GetBuffer());   // Преобразовать из строки в целое
   if(a != 7)                 // Сравнить с 2007 годом
      s += "\n Неверный год";
   AfxMessageBox(s);

   OnOK();
}
```

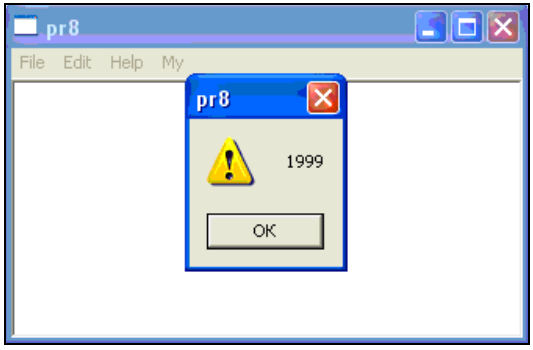
Листинг 8.39. Изменения в файле ChildView.cpp для обработки значений даты и времени после закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;
   // dlg.DoModal()
   if(dlg.DoModal() == IDOK)
   {
      CString s;
      int a = dlg.datetime_val.GetYear();        // Получить год
      s.Format("%d",a);                          // Преобразовать его
                                                 // в строку
      AfxMessageBox(s);
   }
}
```

Описание класса `CTime` и функции `CTime::Format()` были приведены в *разд. 7.1.6.*



а



б

Рис. 8.28. Обработка даты и времени до (*а*) и после (*б*) закрытия окна диалога

Функция получения значений из окна даты и времени выглядит так:

```
DWORD CDateTimeCtrl::GetTime(            // 0 - ошибка
   CTime& timeDest) const;               // Для заполнения значений
```

Получение значения года выполняется с помощью функции:

```
int COleDateTime::GetYear() const throw();  // Значение года
                                            // (от 100 до 9999)
```

Получение значения дня:

```
int COleDateTime::GetDay() const throw();   // Значение от 1 до 31
```

Получение значения месяца:

```
int COleDateTime::GetMonth() const throw(); // Значение от 1 до 12
```

Получение значения года:

```
int COleDateTime::GetHour() const throw();  // Значение от 0 до 23
```

Получение значения минут:

```
int COleDateTime::GetMinute() const throw();  // Значение от 0 до 59
```

Получение значения секунд:

```
int COleDateTime::GetSecond() const throw();  // Значение от 0 до 59
```

Получение значения дня в неделе:

```
int COleDateTime::GetDayOfWeek() const throw();   // Значение от 1 до 7
                                                  // 1=Sunday,
2=Monday,...
```

Получение значения дня в году:

```
int COleDateTime::GetDayOfYear() const throw();  // Значение от 1 до 366
                                                 // January 1 = 1
```

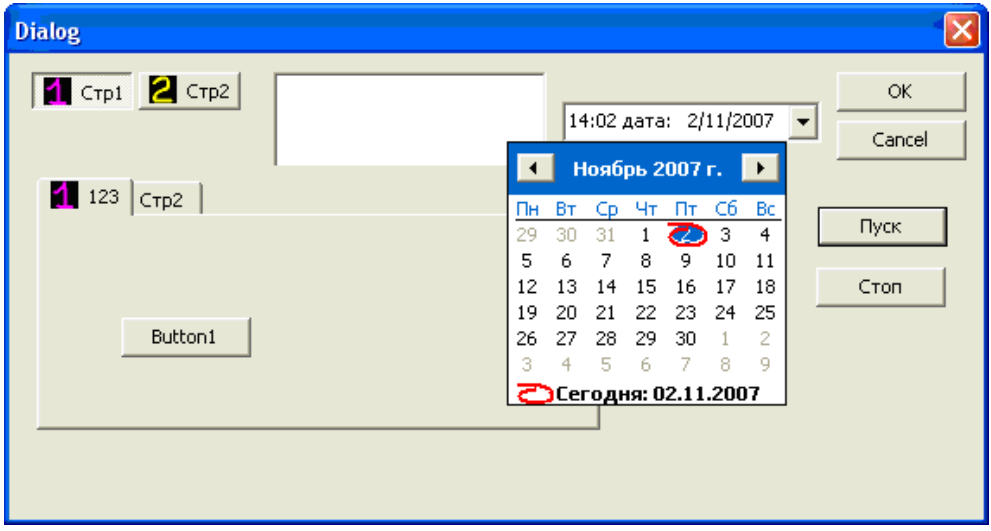


Рис. 8.29. Выбор даты и времени
с помощью раскрывающегося календаря

Результат работы программы показан на рис. 8.28. Для проверки работы после запуска программы в поле даты и времени год был изменен с 2007 на 1999. Изменять значения даты можно не только вручную, но и с помощью

календаря, для этого надо нажать на стрелку справа от поля даты и времени (рис. 8.29).

8.1.6. Календарь (*Month Calendar Control*)

Добавим календарь **Month Calendar Control** (рис. 8.30), переменную `month` (категории **Control**) для работы с календарем (рис. 8.31) и переменную `month_val` (категории **Value**) для сохранения значений календаря (рис. 8.32). Изменения в файлах приведены в листингах 8.40—8.42.

Листинг 8.40. Изменения в файле Resource.h для работы с календарем

```
// ...
#define IDC_MONTHCALENDAR1              1007
// ...
```

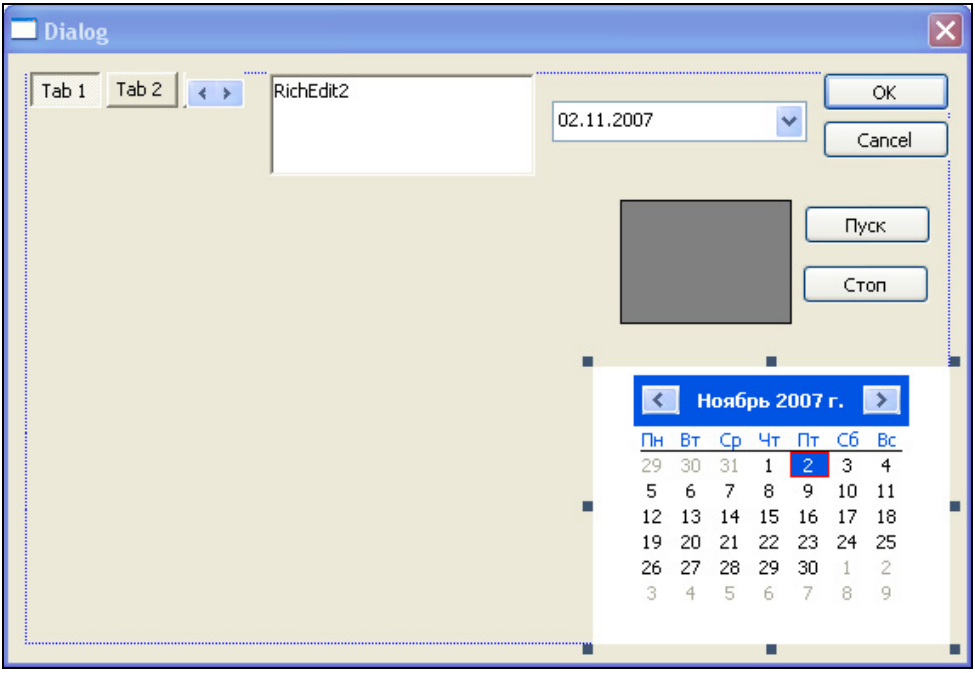


Рис. 8.30. Добавление календаря

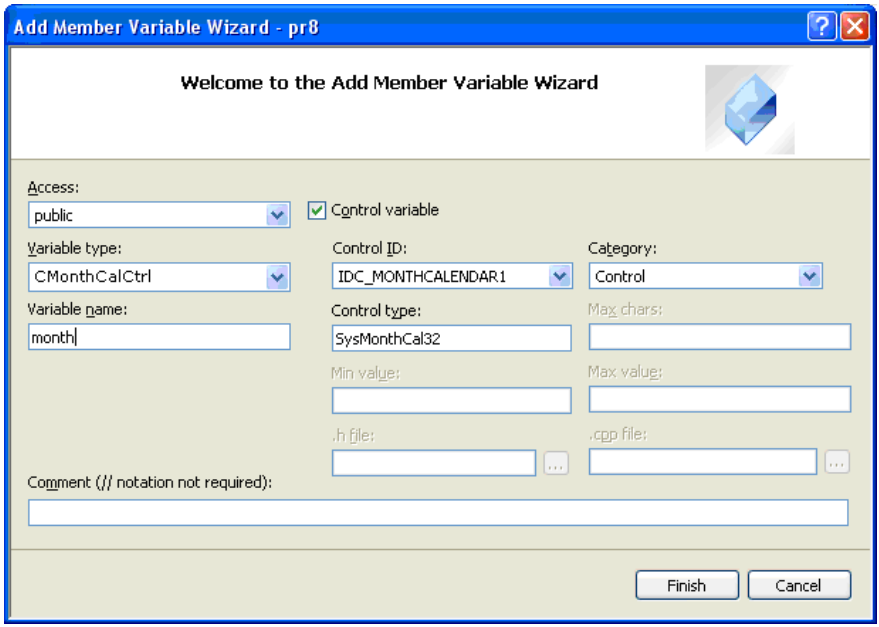


Рис. 8.31. Переменная для работы с календарем

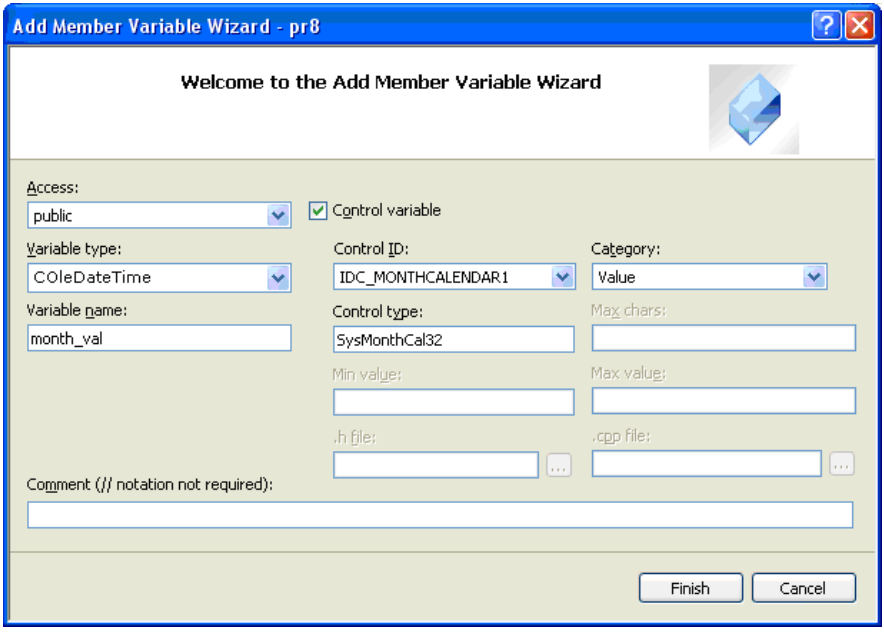


Рис. 8.32. Переменная для сохранения значения календаря

Листинг 8.41. Изменения в файле MyDlg.h для работы с календарем

```
// ...
class MyDlg : public CDialog
{
// ...
public:                                  // Для работы с календарем
   CMonthCalCtrl month;
public:
   COleDateTime month_val;
};
```

Листинг 8.42. Изменения в файле MyDlg.cpp для работы с календарем

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/)
   : CDialog(MyDlg::IDD, pParent)
   , redit_val(_T(""))
   , datetime_val(COleDateTime::GetCurrentTime())
   , month_val(COleDateTime::GetCurrentTime())
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_MONTHCALENDAR1, month);
   DDX_MonthCalCtrl(pDX, IDC_MONTHCALENDAR1, month_val);
}
```

Результат работы программы показан на рис. 8.33.

Добавим обработку значений даты для двух случаев — до закрытия окна диалога (в файле MyDlg.cpp) и после закрытия окна диалога (в файле ChildView.cpp). Изменения в файлах приведены в листингах 8.43 и 8.44.

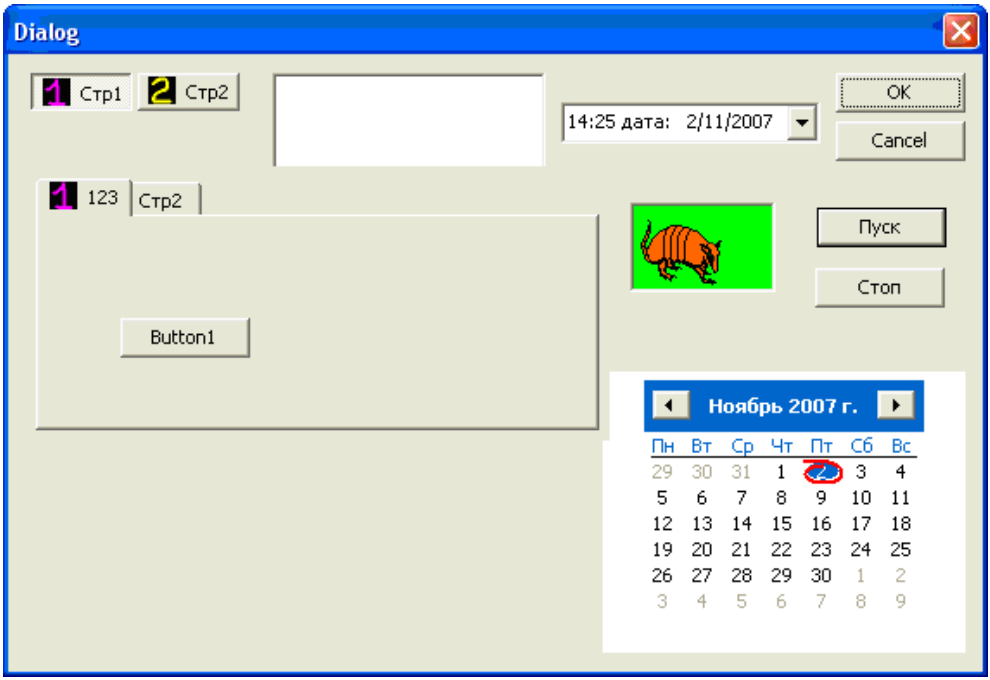
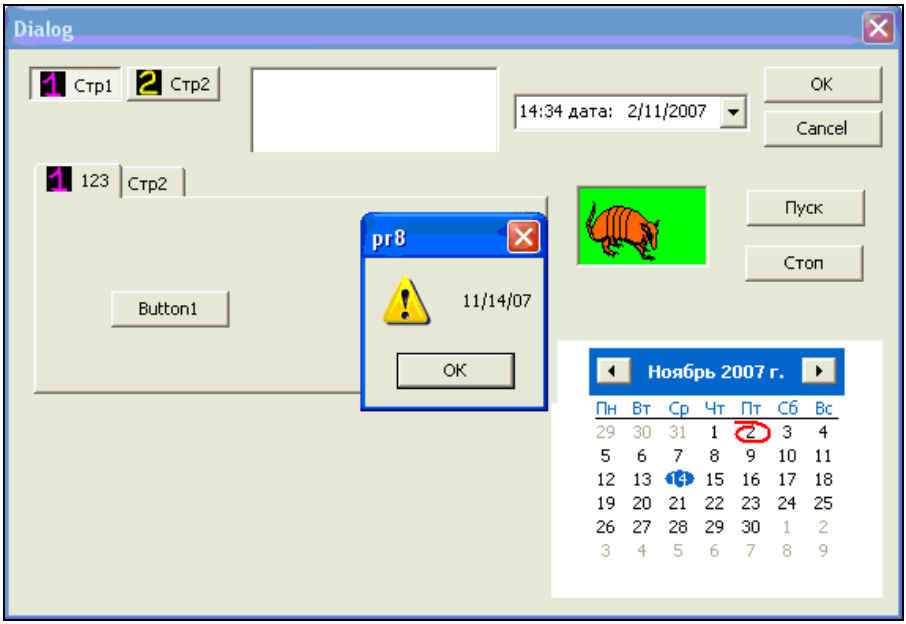


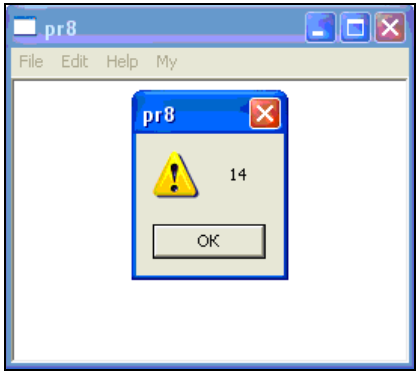
Рис. 8.33. Работа с календарем

Листинг 8.43. Изменения в файле MyDlg.cpp для обработки значения календаря до закрытия окна диалога

```
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // Календарь
   month.GetCurSel(time);
   s = time.Format("%x");       // Взять данные в формате месяц/число/год
   AfxMessageBox(s);

   OnOK();
}
```



а



б

Рис. 8.34. Обработка календаря до (*а*) и после (*б*) закрытия окна диалога

Листинг 8.44. Изменения в файле ChildView.cpp для обработки значения календаря после закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;
```

```
   if(dlg.DoModal() == IDOK )
   {
      // ...
      // Календарь
      a = dlg.month_val.GetDay();          // Получить число
      s.Format("%d",a);                    // Преобразовать его в строку
      AfxMessageBox(s);
   }
}
```

Результат работы программы показан на рис. 8.34. Для проверки работы после запуска программы в календаре дата была изменена с 2 на 14.

8.1.7. IP-адрес (*IP Address Control*)

Добавим элемент управления **IP Address Control** (рис. 8.35), переменную `ipa` (категории **Control**) для работы с IP-адресом (рис. 8.36) и переменную `ipa_val` (категории **Value**) для сохранения значения IP-адреса (рис. 8.37). Изменения в файлах приведены в листингах 8.45—8.47.

Листинг 8.45. Изменения в файле Resource.h для работы с IP-адресом

```
// ...
#define IDC_IPADDRESS1              1008
// ...
```

Листинг 8.46. Изменения в файле MyDlg.h для работы с IP-адресом

```
// ...
class MyDlg : public CDialog
{
// ...
public:                                 // Для работы с IP-адресом
   CIPAddressCtrl ipa;
public:
   DWORD ipa_val;
};
```

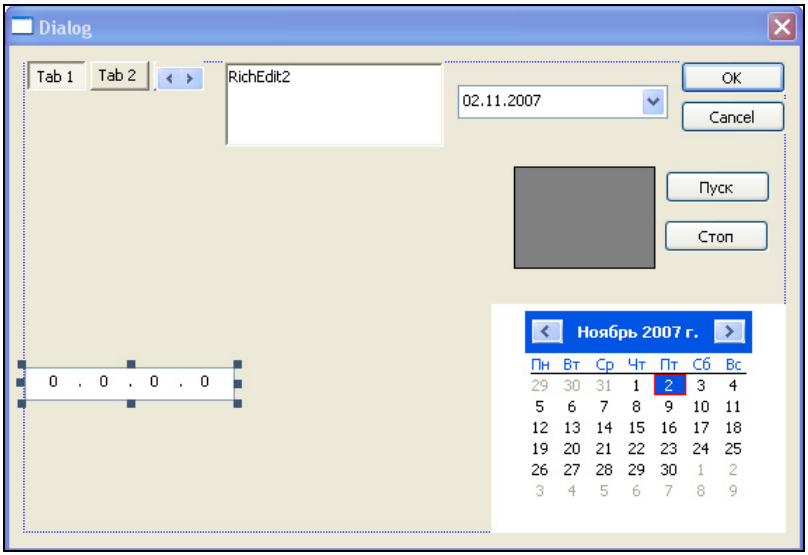


Рис. 8.35. Добавление поля IP-адреса

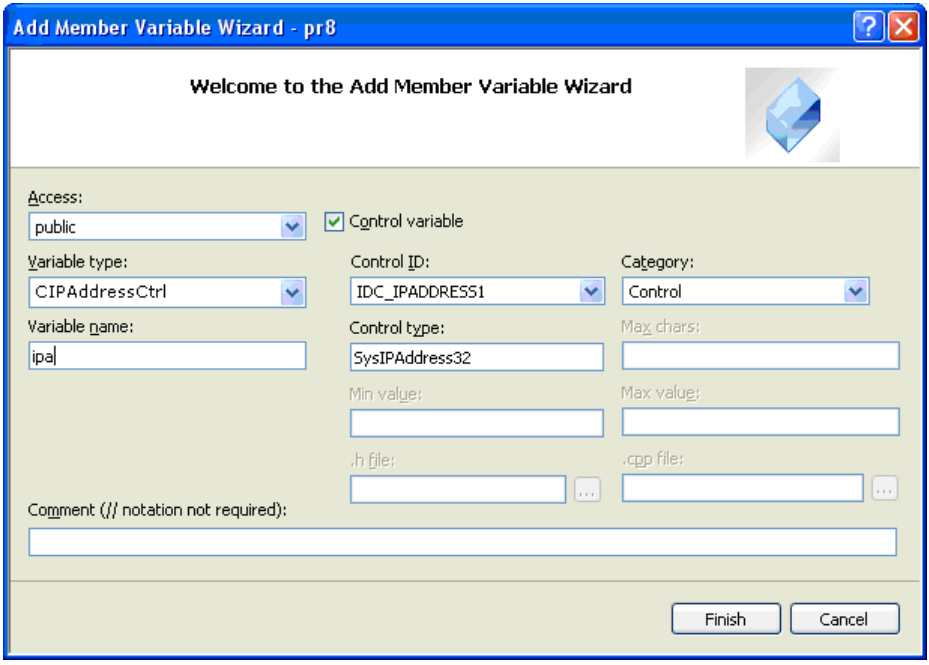


Рис. 8.36. Переменная для работы с IP-адресом

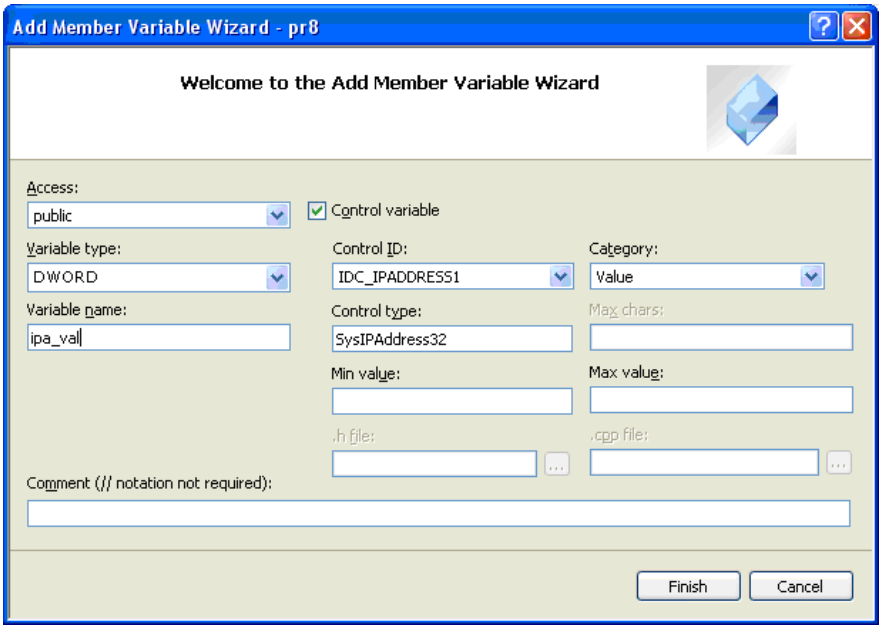


Рис. 8.37. Переменная для сохранения значения IP-адреса

Листинг 8.47. Изменения в файле MyDlg.cpp для работы с IP-адресом

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , redit_val(_T(""))
   , datetime_val(COleDateTime::GetCurrentTime())
   , month_val(COleDateTime::GetCurrentTime())
   , ipa_val(0)
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_IPADDRESS1, ipa);
   DDX_IPAddress(pDX, IDC_IPADDRESS1, ipa_val);
}
```

Добавим обработку значений IP-адреса для двух случаев — до закрытия окна диалога (в файле MyDlg.cpp) и после закрытия (в файле ChildView.cpp) окна диалога. Изменения в файлах приведены в листингах 8.48 и 8.49.

Листинг 8.48. Изменения в файле MyDlg.cpp для обработки значения IP-адреса до закрытия окна диалога

```
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // IP адрес
   BYTE p1,p2,p3,p4;                  // Четыре поля адреса (p1.p2.p3.p4)
   CString p;
   ipa.GetAddress(p1,p2,p3,p4);       // Получить значения адреса
   p.Format("%d.", p1);               // Преобразовать их в строку
   s = p;
   p.Format("%d.", p2);
   s += p;
   p.Format("%d.", p3);
   s += p;
   p.Format("%d", p4);
   s += p;
   AfxMessageBox(s);

   OnOK();
}
```

Листинг 8.49. Изменения в файле ChildView.cpp для обработки значения IP-адреса после закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;
```

```
   if(dlg.DoModal() == IDOK)
   {
      // ...
      // IP адрес
      union UN           // Для извлечения значений из DWORD dlg.ipa_val
      {
         BYTE p[4];
         DWORD dw;
      } ip;
      CString st;
      ip.dw = dlg.ipa_val;
      st.Format("%d.", ip.p[3]);
      s = st;
      st.Format("%d.", ip.p[2]);
      s += st;
      st.Format("%d.", ip.p[1]);
      s += st;
      st.Format("%d", ip.p[0]);
      s += st;
      AfxMessageBox(s);
   }
}
```

Функция получения IP-адреса выглядит так:

```
int CIPAddressCtrl::GetAddress(     // Число ненулевых полей адреса
   BYTE& nField0,                   // Значения первого,
   BYTE& nField1,                   // второго,
   BYTE& nField2,                   // третьего
   BYTE& nField3);                  // и четвертого полей адреса
```

или

```
int CIPAddressCtrl::GetAddress(
   DWORD& dwAddress);              // Полное значение адреса
                                   // (unsigned long)
```

Функция установки значений адреса определена следующим образом:

```
void CIPAddressCtrl::SetAddress(
   BYTE& nField0,                    // Значения первого,
   BYTE& nField1,                    // второго,
   BYTE& nField2,                    // третьего
   BYTE& nField3);                   // и четвертого полей адреса
```

или

```
void CIPAddressCtrl::SetAddress(
   DWORD dwAddress);                 // Полное значение адреса
```

Результат работы программы показан на рис. 8.38 и 8.39. Для проверки работы после запуска программы IP-адрес был изменен с 0.0.0.0 на 111.122.133.144.

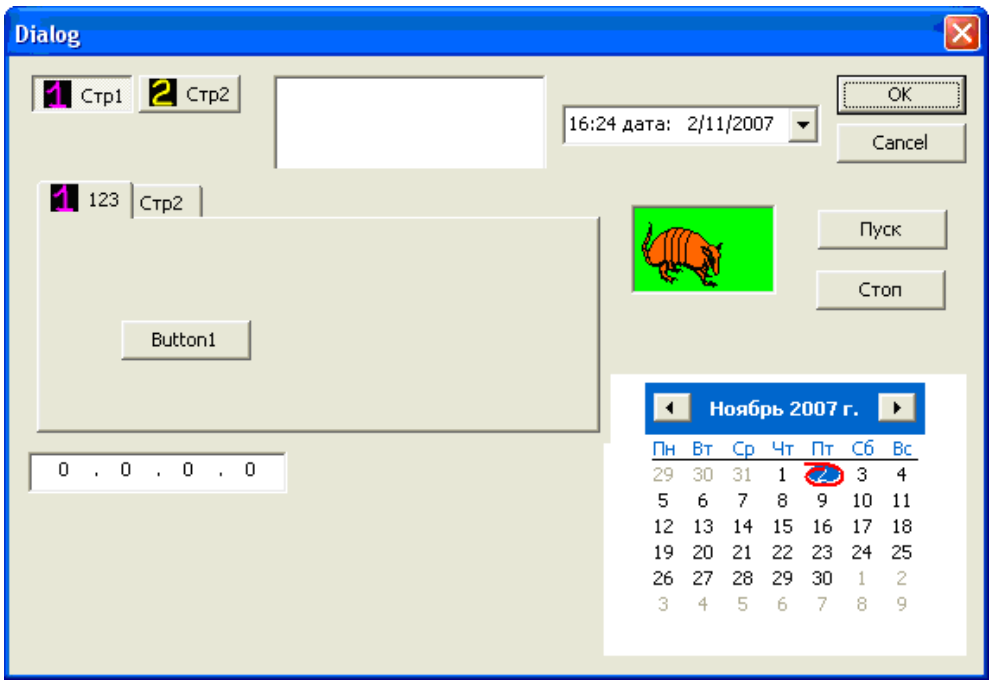


Рис. 8.38. Значение IP-адреса при запуске программы

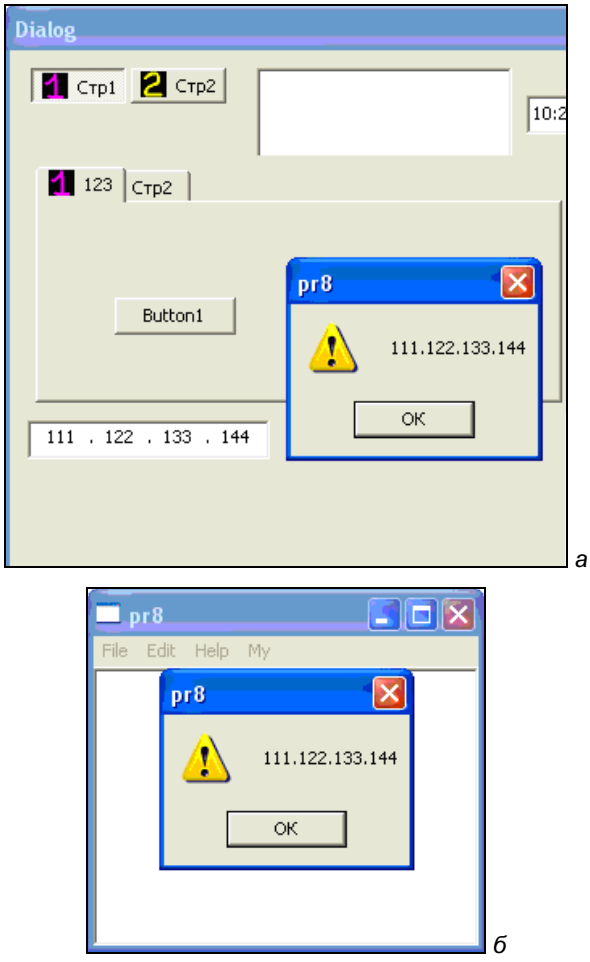


Рис. 8.39. Обработка значения IP-адреса до закрытия окна диалога (*а*) и после его закрытия (*б*)

8.1.8. Расширенное поле со списком (*Extended Combo Box*)

Добавим список **Extended Combo Box** (рис. 8.40), переменную `box` (категории **Control**) для работы со списком (рис. 8.41) и переменную `box_val` (категории **Value**) для сохранения значения списка (рис. 8.42). Изменения в файлах приведены в листингах 8.50—8.52.

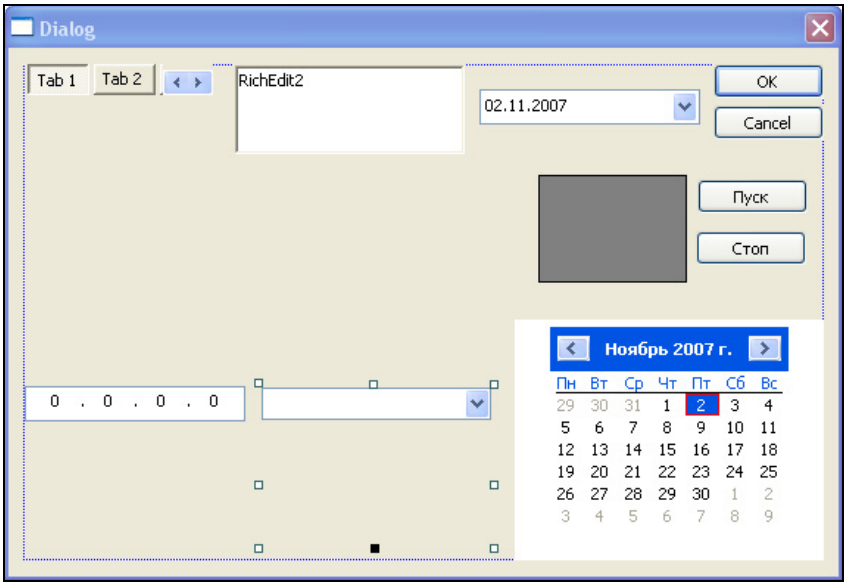


Рис. 8.40. Добавление расширенного списка

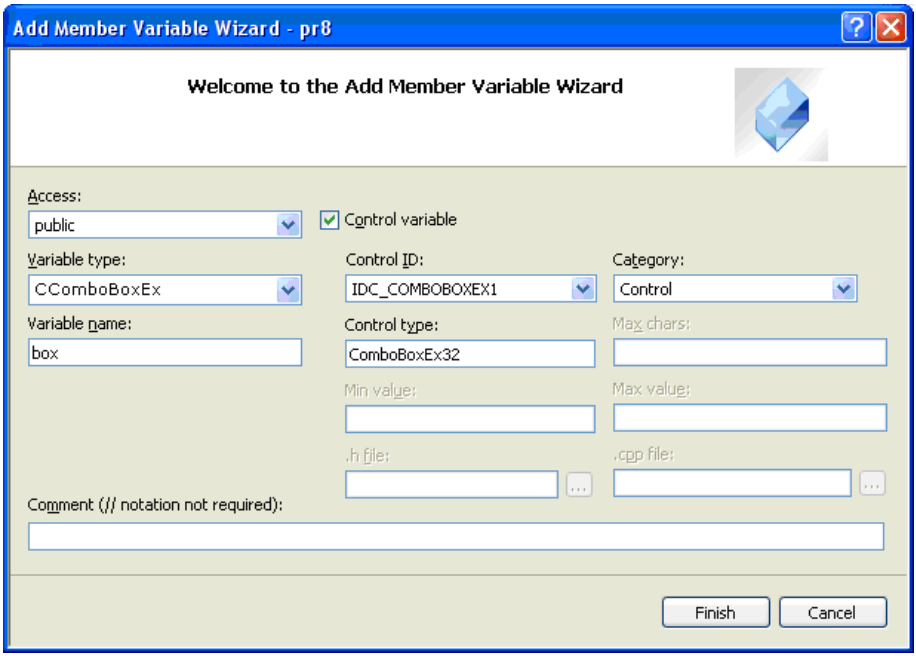


Рис. 8.41. Переменная для работы с расширенным списком

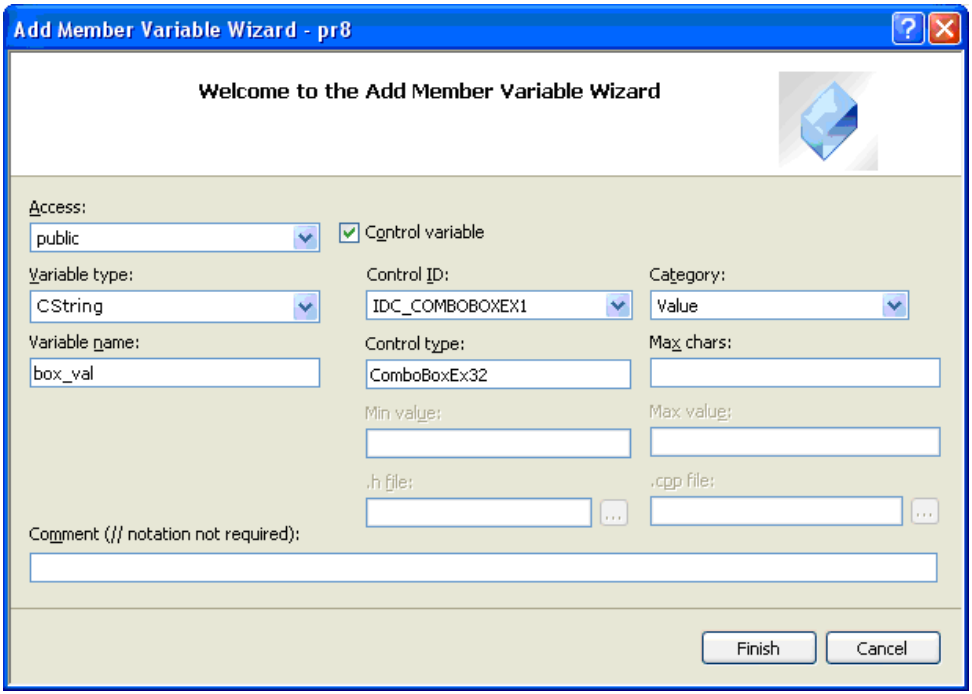


Рис. 8.42. Переменная для сохранения значения расширенного списка

Листинг 8.50. Изменения в файле Resource.h для работы с расширенным списком

```
// ...
#define IDC_COMBOBOXEX1                 1009
// ...
```

Листинг 8.51. Изменения в файле MyDlg.h для работы с расширенным списком

```
// ...
class MyDlg : public CDialog
{
// ...
public:                                 // Для работы со списком
   CComboBoxEx box;
```

```
public:
   CString box_val;
};
```

Листинг 8.52. Изменения в файле MyDlg.cpp для работы с расширенным списком

```
// ...
MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , redit_val(_T(""))
   , datetime_val(COleDateTime::GetCurrentTime())
   , month_val(COleDateTime::GetCurrentTime())
   , ipa_val(0)
   , box_val(_T(""))
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
{
   // ...
   DDX_Control(pDX, IDC_COMBOBOXEX1, box);
   DDX_CBString(pDX, IDC_COMBOBOXEX1, box_val);
}
```

Заполним список. Изменения в файле приведены в листинге 8.53.

Листинг 8.53. Изменения в файле MyDlg.cpp для начальной инициализации списка

```
// ...
BOOL MyDlg::OnInitDialog()
{
   // ...
   // Список с иконками
   image.Create(16, 16, NULL, 0, 1);
   image.Add(theApp.LoadIconA(IDI_ICON1));
```

```
   image.Add(AfxGetApp()->LoadIconA(IDI_ICON2));
   // ...
   // Заполнение комбинированного списка
   box.SetImageList(&image);     // Связать список со списком изображений
   COMBOBOXEXITEM cb;
   // Подготовка первого элемента списка ([0])
   cb.mask = CBEIF_IMAGE | CBEIF_TEXT;
   cb.iImage = 0;                // Индекс элемента в списке
   cb.pszText = "строка1";       // Текст элемента
   cb.cchTextMax = 8;            // Длина текста элемента
   cb.iItem = 0;                 // Картинка из списка изображений ([0])
   box.InsertItem(&cb);          // Занесение элемента в список
   cb.mask = CBEIF_IMAGE | CBEIF_TEXT;     // Подготовка второго элемента
   cb.iImage = 1;
   cb.pszText = "строка2";
   cb.cchTextMax = 8;
   cb.iItem = 1;
   box.InsertItem(&cb);          // Занесение элемента в список
   box.SetCurSel(0);             // Сделать текущим первый элемент

   return TRUE;  // return TRUE unless you set the focus to a control
   // EXCEPTION: OCX Property Pages should return FALSE
}
```

Структура, содержащая данные об элементе списка, определена так:

```
typedef struct
{
   UINT mask;             // Значение, определяющее, какие переменные
                          // структуры должны быть возвращены или
                          // установлены
   INT_PTR iItem;         // Индекс элемента в списке
   LPTSTR pszText;        // Текст элемента
   int cchTextMax;        // Длина текста
   int iImage;            // Индекс рисунка из подключенного списка
                          // изображений
```

```
   int iSelectedImage;     // Индекс рисунка списка изображений, который
                           // будет отображаться, когда элемент будет
                           // выбран
   int iOverlay;           // Индекс рисунка списка изображений, который
                           // будет отображаться, когда элемент недоступен
   int iIndent;            // Количество отступов для элемента
                           // (1 отступ = 10 пикселов)
   LPARAM lParam;          // Дополнительные параметры
} COMBOBOXEXITEM, *PCOMBOBOXEXITEM;
```

Значения маски `mask` (могут комбинироваться) бывают такими:

- `CBEIF_IMAGE` — поле `iImage` должно быть заполнено;
- `CBEIF_INDENT` — поле `iIndent` должно быть заполнено;
- `CBEIF_LPARAM` — поле `lParam` должно быть заполнено;
- `CBEIF_OVERLAY` — поле `iOverlay` должно быть заполнено;
- `CBEIF_SELECTEDIMAGE` — поле `iSelectedImage` должно быть заполнено;
- `CBEIF_TEXT` — поле `pszText` должно быть заполнено.

Функция занесения элемента в список выглядит так:

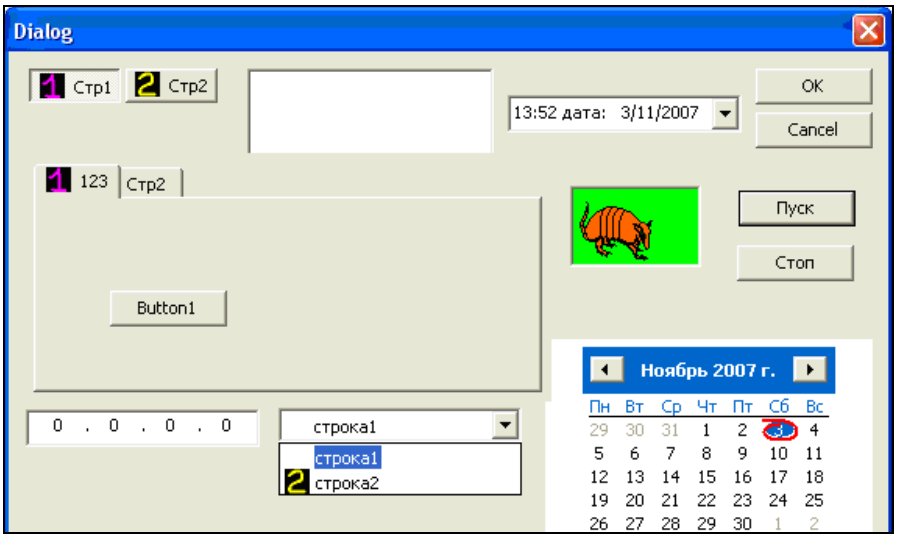
```
int CComboBoxEx::InsertItem(      // Индекс нового элемента или
                                  // -1 при ошибке
   const COMBOBOXEXITEM* pCBItem); // Указатель на структуру (см. выше)
```

Добавим обработку значений списка для двух случаев — до закрытия окна диалога (в файл MyDlg.cpp) и после закрытия (в файл ChildView.cpp) окна диалога. Изменения в файлах приведены в листингах 8.54 и 8.55.

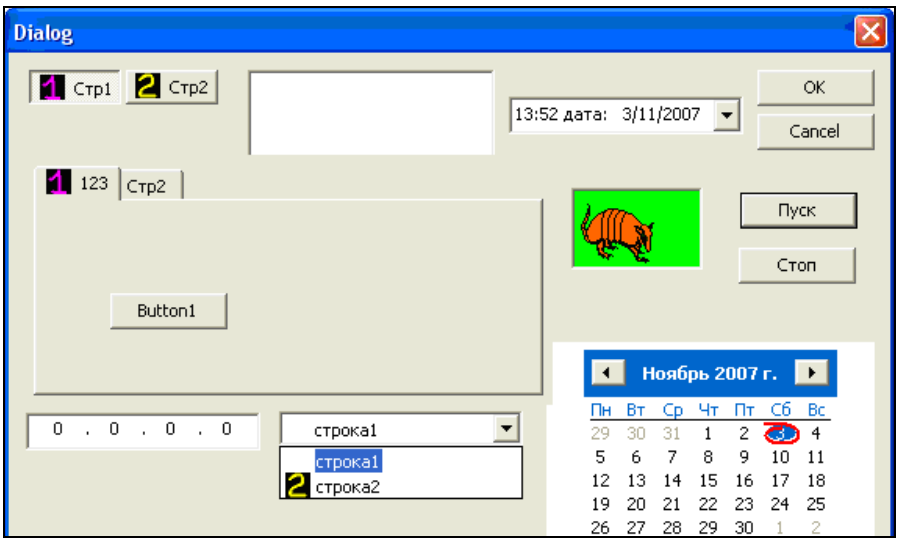
Листинг 8.54. Изменения в файле MyDlg.cpp для обработки выбранного элемента списка до закрытия окна диалога

```
// ...
void MyDlg::OnBnClickedOk()
{
   // ...
   // Список
   if(!box.GetWindowTextLengthA())
   {
      AfxMessageBox("Выберите название из списка!!!");
```

```
        return;
    }

    OnOK();
}
```

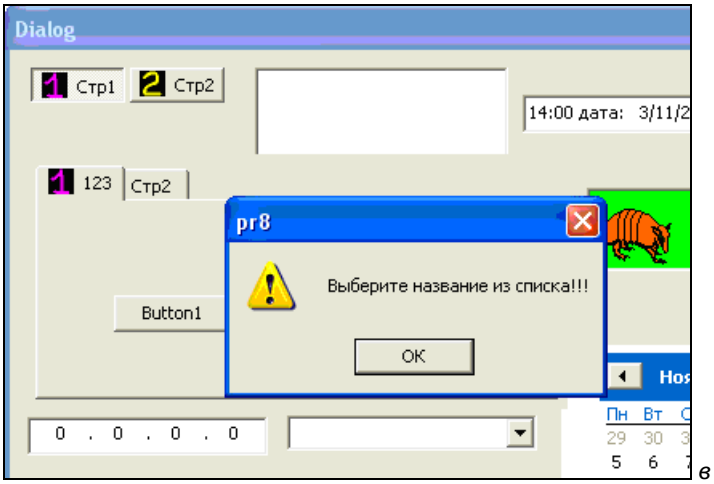


а



б

Рис. 8.43. Значение элемента в списке по умолчанию (*а*), контроль выбора элемента в списке (*б*)



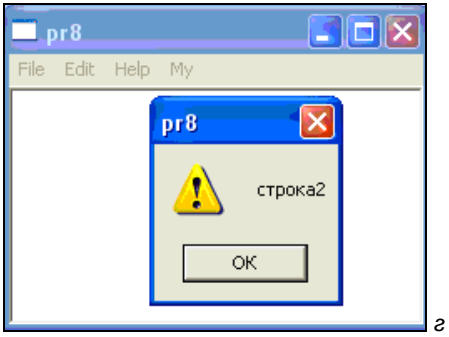


Рис. 8.43. Выбор элемента в списке (*в*)
и обработка выбранного из списка элемента (*г*)

Листинг 8.55. Изменения в файле ChildView.cpp для обработки выбранного элемента списка после закрытия окна диалога

```
// ...
void CChildView::OnMyDlg()
{
   // TODO: Add your command handler code here
   MyDlg dlg;
   if(dlg.DoModal() == IDOK)
   {
      // ...
```

```
      // Список
      if(dlg.box_val.GetLength())
         AfxMessageBox(dlg.box_val);
   }
}
```

Функция получения длины текста объекта выглядит так:

```
int CWnd::GetWindowTextLength() const;
```

Результат работы программы показан на рис. 8.43. Для проверки работы в поле выбора списка была удалена вся информация и нажата кнопка **OK**. Затем после сообщения о том, что надо выбрать элемент списка, была выбрана **строка2** и снова нажата кнопка **OK**.

8.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 8.56. Файл Resource.h (идентификаторы ресурсов приложения)

```
//...
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDD_PROPPAGE_SMALL2             102
#define IDR_MAINFRAME                   128
#define IDR_pr8TYPE                     129
#define IDD_DIALOG1                     130
#define IDI_ICON1                       131
#define IDI_ICON2                       132
#define IDD_PROPPAGE_SMALL1             133
#define IDC_TAB1                        1000
#define IDC_BUTTON1                     1001
#define IDC_CHECK1                      1002
#define IDC_ANIMATE1                    1003
#define IDC_BUTTON2                     1004
```

```
#define IDC_RICHEDIT21                          1005
#define IDC_DATETIMEPICKER1                     1006
#define IDC_MONTHCALENDAR1                      1007
#define IDC_IPADDRESS1                          1008
#define IDC_COMBOBOXEX1                         1009
#define ID_MY_DLG                               32771
// ...
```

Листинг 8.57. Файл MyDlg.h (объявление класса диалога)

```
// MyDlg.h
#pragma once
#include "afxcmn.h"
#include "MyPage1.h"
#include "MyPage2.h"
#include "afxdtctl.h"

// MyDlg dialog

class MyDlg : public CDialog
{
   DECLARE_DYNAMIC(MyDlg)

public:
   MyDlg(CWnd* pParent = NULL);   // standard constructor
   virtual ~MyDlg();

// Dialog Data
   enum { IDD = IDD_DIALOG1 };

protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

   DECLARE_MESSAGE_MAP()
public:
   virtual BOOL OnInitDialog();
```

```
public:
   CTabCtrl tab1;                    // Набор вкладок
   CImageList image;                 // Список изображений
public:
   afx_msg void OnBnClickedOk();

   CPropertySheet sheet;             // Класс страницы свойств
   CMyPage1 page1;                   // Класс первой вкладки-диалога
   CMyPage2 page2;                   // Класс второй вкладки-диалога

public:
   CAnimateCtrl anim1;               // Для работы с анимацией
public:
   virtual BOOL DestroyWindow();     // Разрушение окна (для закрытия файла
                                     // анимации)
public:
   afx_msg void OnBnClickedButton1();     // Кнопка запуска анимации
public:
   afx_msg void OnBnClickedButton2();     // Кнопка остановки анимации

public:
   CRichEditCtrl redit;                   // Контроль окна редактора
public:
   CString redit_val;                     // Содержимое окна редактора

public:                                 // Для работы с датой и временем
   CDateTimeCtrl datetime;
public:
   COleDateTime datetime_val;

public:                                  // Для работы с календарем
   CMonthCalCtrl month;
public:
   COleDateTime month_val;

public:                                 // Для работы с IP=адресом
   CIPAddressCtrl ipa;
```

```
public:
   DWORD ipa_val;

public:                                    // Для работы со списком
   CComboBoxEx box;
public:
   CString box_val;
};
```

Листинг 8.58. Файл MyDlg.cpp (определение класса диалога)

```
// MyDlg.cpp : implementation file
//
#include "stdafx.h"
#include "pr8.h"
#include "MyDlg.h"

// MyDlg dialog

IMPLEMENT_DYNAMIC(MyDlg, CDialog)

MyDlg::MyDlg(CWnd* pParent /*=NULL*/) : CDialog(MyDlg::IDD, pParent)
   , redit_val(_T(""))
   , datetime_val(COleDateTime::GetCurrentTime())
   , month_val(COleDateTime::GetCurrentTime())
   , ipa_val(0)
   , box_val(_T(""))
{
}

MyDlg::~MyDlg()
{
}

void MyDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
   CDialog::DoDataExchange(pDX);

   DDX_Control(pDX, IDC_TAB1, tab1);

   DDX_Control(pDX, IDC_ANIMATE1, anim1);

   DDX_Control(pDX, IDC_RICHEDIT21, redit);
   DDX_Text(pDX, IDC_RICHEDIT21, redit_val);

   DDX_Control(pDX, IDC_DATETIMEPICKER1, datetime);
   DDX_DateTimeCtrl(pDX, IDC_DATETIMEPICKER1, datetime_val);

   DDX_Control(pDX, IDC_MONTHCALENDAR1, month);
   DDX_MonthCalCtrl(pDX, IDC_MONTHCALENDAR1, month_val);

   DDX_Control(pDX, IDC_IPADDRESS1, ipa);
   DDX_IPAddress(pDX, IDC_IPADDRESS1, ipa_val);

   DDX_Control(pDX, IDC_COMBOBOXEX1, box);
   DDX_CBString(pDX, IDC_COMBOBOXEX1, box_val);
}

BEGIN_MESSAGE_MAP(MyDlg, CDialog)
   ON_BN_CLICKED(IDOK, &MyDlg::OnBnClickedOk)
   ON_BN_CLICKED(IDC_BUTTON1, &MyDlg::OnBnClickedButton1)
   ON_BN_CLICKED(IDC_BUTTON2, &MyDlg::OnBnClickedButton2)
END_MESSAGE_MAP()

// MyDlg message handlers

BOOL MyDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // TODO:  Add extra initialization here
   // Список с иконками
```

```
image.Create(16, 16, NULL, 0, 1);
image.Add(theApp.LoadIconA(IDI_ICON1));
image.Add(AfxGetApp()->LoadIconA(IDI_ICON2));          // Можно и так
tab1.SetImageList(&image);
// Заполнение страниц (вкладок)
tab1.InsertItem(0,"Стр1",0);
tab1.InsertItem(1,"Стр2",1);

// Добавление страниц-диалогов
sheet.AddPage(&page1);
sheet.AddPage(&page2);
// Создание страницы свойств
sheet.Create(this, WS_CHILD | WS_VISIBLE, WS_EX_CONTROLPARENT);
// Положение относительно родительского окна диалога (IDD_DIALOG1)
sheet.SetWindowPos(this, 10,65,0,0, SWP_NOZORDER | SWP_NOSIZE |
                   SWP_NOACTIVATE);
// Изменения
// Получить указатель на список страницы
CTabCtrl *pt = sheet.GetTabControl();
// Связать список страниц со списком изображений
pt->SetImageList(&image);
TCITEM st;
// Получить текущие свойства первой страницы (с индексом 0)
st.mask = TCIF_TEXT | TCIF_IMAGE;
pt->GetItem(0, &st);
// Задать новый заголовок и иконку
st.mask = TCIF_TEXT | TCIF_IMAGE;
st.pszText = "123";
st.cchTextMax = 4;
st.iImage = 0;   // Изменить свойства по заданным значениям
pt->SetItem(0,&st);

// animal
anim1.Open("res\\dillo.avi");
// Формат для отображения даты и времени
```

```
    datetime.SetFormat("H:mm дата: d/M/yyyy");

    // Заполнение комбинированного списка
    box.SetImageList(&image);     // Связать список со списком изображений
    COMBOBOXEXITEM cb;
    cb.mask = CBEIF_IMAGE | CBEIF_TEXT;   // Подготовка первого элемента
                                          // списка ([0])
    cb.iImage = 0;                        // Индекс элемента в списке
    cb.pszText = "строка1";               // Текст элемента
    cb.cchTextMax = 8;                    // Длина текста элемента
    cb.iItem = 0;                         // Картинка из списка
                                          // изображений [0]
    box.InsertItem(&cb);                  // Занесение элемента в список
    cb.mask = CBEIF_IMAGE | CBEIF_TEXT;   // Подготовка второго
                                          // элемента списка ([1])
    cb.iImage = 1;
    cb.pszText = "строка2";
    cb.cchTextMax = 8;
    cb.iItem = 1;
    box.InsertItem(&cb);                 // Занесение элемента в список
    box.SetCurSel(0);                    // Сделать текущим первый элемент

    return TRUE;  // return TRUE unless you set the focus to a control
    // EXCEPTION: OCX Property Pages should return FALSE
}

void MyDlg::OnBnClickedOk()
{
    // TODO: Add your control notification handler code here
    // TabControl
    // Получить индекс текущего выбранного элемента
    int a = tab1.GetCurSel();
    CString s;
    s.Format("Выбрана страница %d", a+1);
    AfxMessageBox(s);
    // CPropertySheet
```

```
// Получить индекс текущей диалоговой вкладки
CTabCtrl *pt = sheet.GetTabControl();
a = pt->GetCurSel();
s.Format("Диалог-вкладка %d", a+1);
AfxMessageBox(s);
// Если была выбрана вкладка 2 - узнать состояние переключателя
//check вкладки 2
CButton *pc = (CButton *)page2.GetDlgItem(IDC_CHECK1);
if( pc->GetCheck())
   AfxMessageBox("check YES");
else
   AfxMessageBox("check NO");

// CRichEditCtrl
if(!redit.GetTextLength())
   AfxMessageBox("Текст не был введен");
else
{
   redit.GetWindowTextA(s);
   AfxMessageBox(s);
}

// Дата и время
CTime time;
datetime.GetTime(time);       // Получить значения даты и времени
s = time.Format("%y");        // Взять из них только последние две
                              // цифры года
a = atoi(s.GetBuffer());      // Преобразовать из строки в целое
if(a != 7)                    // Сравнить с 2007 годом
   s += "\n Неверный год";
AfxMessageBox(s);

// Календарь
month.GetCurSel(time);
s = time.Format("%x");        // Взять данные в формате месяц/число/год
```

```
    AfxMessageBox(s);

    // IP адрес
    BYTE p1,p2,p3,p4;                // Четыре поля адреса (p1.p2.p3.p4)
    CString p;
    ipa.GetAddress(p1,p2,p3,p4);     // Получить значения адреса
    p.Format("%d.", p1);             // Преобразовать их в строку
    s = p;
    p.Format("%d.", p2);
    s += p;
    p.Format("%d.", p3);
    s += p;
    p.Format("%d", p4);
    s += p;
    AfxMessageBox(s);

    // Список
    if(!box.GetWindowTextLengthA())
    {
       AfxMessageBox("Выберите название из списка!!!");
       return;
    }

    OnOK();
}

BOOL MyDlg::DestroyWindow()
{
    // TODO: Add your specialized code here and/or call the base class
    anim1.Close();
    return CDialog::DestroyWindow();
}
void MyDlg::OnBnClickedButton1()
{
```

```
    // TODO: Add your control notification handler code here
    anim1.Play(0, -1, -1);
}

void MyDlg::OnBnClickedButton2()
{
    // TODO: Add your control notification handler code here
    anim1.Stop();
}
```

Листинг 8.59. Файл ChildView.h (объявление класса представления)

```
// ...
class CChildView : public CWnd
{
// ...
public:
    afx_msg void OnMyDlg();
};
```

Листинг 8.60. Файл ChildView.cpp (определение класса представления)

```
// ChildView.cpp : implementation of the CChildView class
//
#include "stdafx.h"
#include "pr8.h"
#include "ChildView.h"
#include "MyDlg.h"
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
    ON_WM_PAINT()
    ON_COMMAND(ID_MY_DLG, &CChildView::OnMyDlg)
END_MESSAGE_MAP()
// ...
void CChildView::OnMyDlg()
```

```
{
   // TODO: Add your command handler code here
   MyDlg dlg;
   if(dlg.DoModal() == IDOK)
   {
      // Дата и время
      CString s;
      int a = dlg.datetime_val.GetYear();     // Получить год
      s.Format("%d",a);                       // Преобразовать его в стоку
      AfxMessageBox(s);

      // Календарь
      a = dlg.month_val.GetDay();             // Получить число
      s.Format("%d",a);                       // Преобразовать его в стоку
      AfxMessageBox(s);

      // IP адрес
      union UN           // Для извлечения значений из DWORD dlg.ipa_val
      {
         BYTE p[4];
         DWORD dw;
      } ip;
      CString st;
      ip.dw = dlg.ipa_val;
      st.Format("%d.", ip.p[3]);
      s = st;
      st.Format("%d.", ip.p[2]);
      s += st;
      st.Format("%d.", ip.p[1]);
      s += st;
      st.Format("%d", ip.p[0]);
      s += st;
      AfxMessageBox(s);
      // Список
```

```
      if(dlg.box_val.GetLength())
         AfxMessageBox(dlg.box_val);
   }
}
```

Листинг 8.61. Файл MyPage1.h (объявление класса вкладки 1)

```
// MyPage1.h
#pragma once

// CMyPage1 dialog

class CMyPage1 : public CPropertyPage
{
   DECLARE_DYNAMIC(CMyPage1)

public:
   CMyPage1();
   virtual ~CMyPage1();

// Dialog Data
   enum { IDD = IDD_PROPPAGE_SMALL1 };

protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

   DECLARE_MESSAGE_MAP()
};
```

Листинг 8.62. Файл MyPage1.cpp (определение класса вкладки 1)

```
// MyPage1.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "pr8.h"
#include "MyPage1.h"

// CMyPage1 dialog

IMPLEMENT_DYNAMIC(CMyPage1, CPropertyPage)

CMyPage1::CMyPage1() : CPropertyPage(CMyPage1::IDD)
{
}

CMyPage1::~CMyPage1()
{
}

void CMyPage1::DoDataExchange(CDataExchange* pDX)
{
   CPropertyPage::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CMyPage1, CPropertyPage)
END_MESSAGE_MAP()

// CMyPage1 message handlers
```

Листинг 8.63. Файл MyPage2.h (объявление класса вкладки 2)

```
// MyPage2.h
#pragma once

// CMyPage2 dialog

class CMyPage2 : public CPropertyPage
{
   DECLARE_DYNAMIC(CMyPage2)
```

```
public:
   CMyPage2();
   virtual ~CMyPage2();

// Dialog Data
   enum { IDD = IDD_PROPPAGE_SMALL2 };

protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

   DECLARE_MESSAGE_MAP()
};
```

Листинг 8.64. Файл MyPage2.cpp (определение класса вкладки 2)

```
// MyPage2.cpp : implementation file
//
#include "stdafx.h"
#include "pr8.h"
#include "MyPage2.h"

// CMyPage2 dialog

IMPLEMENT_DYNAMIC(CMyPage2, CPropertyPage)

CMyPage2::CMyPage2() : CPropertyPage(CMyPage2::IDD)
{
}

CMyPage2::~CMyPage2()
{
}

void CMyPage2::DoDataExchange(CDataExchange* pDX)
{
```

```
   CPropertyPage::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CMyPage2, CPropertyPage)
END_MESSAGE_MAP()

// CMyPage2 message handlers
```

Листинг 8.65. Файл pr8.cpp (определение класса приложения)

```
// ...
BOOL Cpr8App::InitInstance()
{
//TODO: call AfxInitRichEdit2() to initialize richedit2 library.
   AfxInitRichEdit2();
   // ...
}
```

Глава 9

Панель инструментов и строка состояния

Создадим новый проект, на примере которого рассмотрим возможности работы с панелью инструментов и строкой состояния.

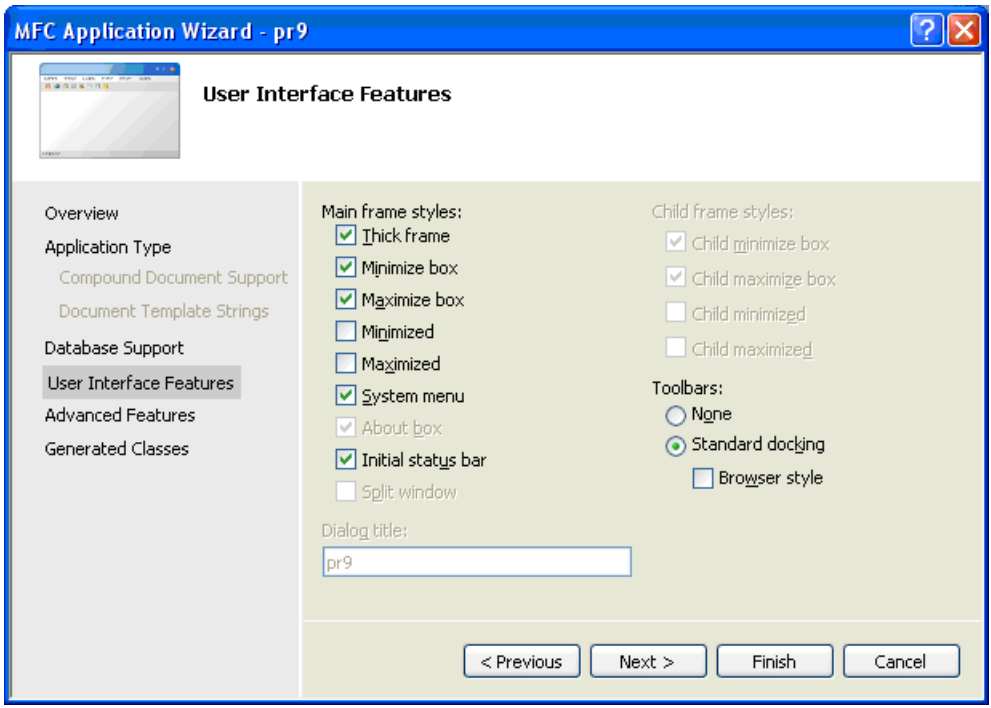


Рис. 9.1. Использование панели инструментов и строки статуса

Создайте проект **pr9** аналогично проекту **pr1** *(см. разд. 1.1)*, *отключив* Unicode, *не отключая* **Initial status bar** (строка состояния) и оставив переключатель **Toolbars** в положении **Standard docking** (панель инструментов) в окне настроек проекта **User Interface Features** (рис. 9.1). На вкладке **Application Type** нужно изменить следующие опции относительно изначально предложенных мастером:

- SDI-документ;
- без поддержки архитектуры документ/представление;
- без Unicode (отключить флажок **Use Unicode libraries**).

9.1. Описание программы

9.1.1. Панель инструментов (*ToolBar*)

При использовании панели инструментов в исходный код будет автоматически добавлен код, отвечающий за работу панели инструментов. Изменения в файлах приведены в листингах 9.1 и 9.2.

Листинг 9.1. Изменения в файле MainFrm.h для поддержки работы с панелью инструментов и строкой состояния

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:  // control bar embedded members
   CToolBar  m_wndToolBar;
   // ...
};
```

Листинг 9.2. Изменения в файле MainFrm.cpp для поддержки работы с панелью инструментов и строкой состояния

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
   if(CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;
   // ...
   if(!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
                             CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
                             CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
      !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
   {
      TRACE0("Failed to create toolbar\n");
      return -1;                // fail to create
   }

   // ...
   // TODO: Delete these three lines if you don't want the toolbar to be
dockable
   m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
   EnableDocking(CBRS_ALIGN_ANY);
   DockControlBar(&m_wndToolBar);

   return 0;
}
```

Класс для работы с панелью инструментов определен как:

```
class CToolBar : public CControlBar
class CControlBar : public CWnd
```

Создание панели инструментов и присоединение ее к объекту выполняется с помощью функции:

```
virtual BOOL CToolBar::CreateEx(      // 0 - ошибка
   CWnd* pParentWnd,                  // Указатель на родительское окно
   DWORD dwCtrlStyle = TBSTYLE_FLAT,  // Дополнительные стили (для
                                      // создания встроенного объекта
                                      // класса CToolBarCtrl
   DWORD dwStyle = WS_CHILD |         // Стиль окна TB
                   WS_VISIBLE | CBRS_ALIGN_TOP,
```

```
CRect rcBorders = CRect(0,0,0,0),  // Размер панели инструментов
UINT nID = AFX_IDW_TOOLBAR);       // Идентификатор окна панели
                                   // инструментов
```

Дополнительные стили панели инструментов (`dwCtrlStyle`) могут быть такими:

- `TBSTYLE_ALTDRAG` — пользователь может изменять положение кнопок при нажатой клавише <Alt>. При этом для кнопок должен быть установлен стиль `CCS_ADJUSTABLE`;
- `TBSTYLE_CUSTOMERASE` — когда панель инструментов получит сообщение `WM_ERASEBKGND` (о перерисовке окна, например, при изменении размеров), родительскому окну будет посылаться сообщение `WM_CUSTOMDRAW` (об изменении и перерисовке);
- `TBSTYLE_FLAT` — панель инструментов с плоскими кнопками. Название кнопки появляется под ней;
- `TBSTYLE_LIST` — текст кнопки будет располагаться справа от нее;
- `TBSTYLE_REGISTERDROP` — генерируется сообщение `TBN_GETOBJECT`, когда курсор "проходит" по кнопке;
- `TBSTYLE_TOOLTIPS` — с кнопкой ассоциируется всплывающая подсказка;
- `TBSTYLE_TRANSPARENT` — прозрачная панель инструментов, в которой кнопки являются непрозрачными. Название кнопки появляется под ней;
- `TBSTYLE_WRAPABLE` — панель инструментов может иметь несколько линий кнопок;
- `TBSTYLE_AUTOSIZE` — ширина кнопок рассчитывается автоматически;
- `TBSTYLE_BUTTON` — стандартная кнопка;
- `TBSTYLE_CHECK` — кнопка с фиксацией ("залипает" при нажатии, чтобы вернуть ее в исходное состояние, надо нажать на нее еще раз);
- `TBSTYLE_CHECKGROUP` — отмечает начало группы кнопок с фиксацией. Кнопка в группе остается нажатой, пока не будет нажата другая кнопка из группы;
- `TBSTYLE_DROPDOWN` — кнопка с раскрывающимся списком;
- `TBSTYLE_GROUP` — отмечает начало группы стандартных кнопок;
- `TBSTYLE_NOPREFIX` — с кнопкой не ассоциирован никакой акселератор;

- `TBSTYLE_SEP` — разделитель-сепаратор (вертикальная полоска между кнопками).

Стили окна панели инструментов (`dwStyle`) бывают следующие:

- `CBRS_TOP` — панель инструментов находится в верхней части окна фрейма;
- `CBRS_BOTTOM` — панель инструментов находится в нижней части фрейма;
- `CBRS_LEFT` — панель инструментов находится в левой части фрейма;
- `CBRS_RIGHT` — панель инструментов находится в правой части фрейма;
- `CBRS_NOALIGN` — панель инструментов не перемещается, когда родительское окно меняет размеры;
- `CBRS_FLYBY` — строка состояния отображает информацию о кнопках панели инструментов;
- `CBRS_TOOLTIPS` — панель инструментов отображает подсказки;
- `CBRS_SIZE_DYNAMIC` — панель инструментов является динамической. Можно менять форму панели инструментов и передвигать ее кнопки, если панель является перемещаемой (но не прикреплена);
- `CBRS_SIZE_FIXED` — панель инструментов является фиксированной;
- `CBRS_FLOATING` — "плавающая" панель инструментов;
- `CBRS_HIDE_INPLACE` — "скрытая" (невидимая) панель инструментов;
- `CBRS_GRIPPER` — для отображения маркера перемещения панели инструментов (серая вертикальная рамка слева от первой кнопки панели инструментов).

Размер панели инструментов установлен по умолчанию `CRect(0,0,0,0)`, т. е. панель инструментов не ограничена фиксированными размерами.

Загрузка панели инструментов выполняется с помощью функции:

```
BOOL CToolBar::LoadToolBar(                  // 0 - ошибка
   LPCTSTR lpszResourceName);                // Указатель на строку с именем
                                             // ресурса панели инструментов
```

или

```
BOOL CToolBar:: LoadToolBar(
   UINT nIDResource);                        // Идентификатор ресурса панели
                                             // инструментов
```

По умолчанию положение панели инструментов можно изменить только программно. Но можно разрешить пользователю самостоятельно перемещать панель инструментов и располагать ее в других местах фреймового окна. О том, что такая возможность разрешена, надо сообщить как панели инструментов, так и фреймовому окну. Для этого надо вызвать функции `CControlBar::EnableDocking()` и `CFrameWnd:: EnableDocking()`.

Задание "прикрепляемости" панели инструментов выполняется функцией:

```
void CControlBar::EnableDocking(
   DWORD dwDockStyle);      // Стиль стыковки панели инструментов с окном
                            // фрейма и возможность ее перемещения
```

Стили стыковки панели инструментов с фреймовым окном (`dwDockStyle`) могут быть такие:

- `CBRS_ALIGN_TOP` — панель инструментов находится в верхней части окна фрейма;
- `CBRS_ALIGN_BOTTOM` — панель инструментов находится в нижней части фрейма;
- `CBRS_ALIGN_LEFT` — панель инструментов находится в левой части фрейма;
- `CBRS_ALIGN_RIGHT` — панель инструментов находится в правой части фрейма;
- `CBRS_ALIGN_ANY` — пользователь может прикрепить панель инструментов к любому краю окна фрейма;
- `CBRS_FLOAT_MULTI` — панель инструментов может перемещаться в единственном мини-фреймовом окне.

Если задать `dwDockStyle = 0`, то панель инструментов не сможет быть прикреплена к фреймовому окну.

Задание возможности перемещения панели инструментов в окне фрейма выполняется с помощью функции:

```
void CFrameWnd::EnableDocking(
   DWORD dwDockStyle);  // Стиль - см. выше (все, кроме CBRS_FLOAT_MULTI)
```

Пользователь может самостоятельно закрепить панель инструментов или оставить ее плавающей. Для программного закрепления панели инструментов надо вызвать функцию `CFrameWnd:: DockControlBar()`, а чтобы сделать ее плавающей — `CFrameWnd::FloatControlBar()`.

Привязка панели инструментов к фреймовому окну выполняется функцией:

```
void CFrameWnd::DockControlBar(
   CControlBar* pBar,              // Указатель на панель инструментов
   UINT nDockBarID = 0,            // Сторона фреймового окна для
                                   // прикрепления панели инструментов
   LPCRECT lpRect = NULL);         // Координаты экрана, где вне рабочей
                                   // области фрейма будет зафиксировано
                                   // окно, содержащее плавающую панель
                                   // инструментов
```

Значения для привязки панели инструментов (`nDockBarID`) могут быть такие:

- `AFX_IDW_DOCKBAR_TOP` — фиксация у верхней стороны фрейма;
- `AFX_IDW_DOCKBAR_BOTTOM` — фиксация у нижней стороны фрейма;
- `AFX_IDW_DOCKBAR_LEFT` — фиксация у левой стороны фрейма;
- `AFX_IDW_DOCKBAR_RIGHT` — фиксация у правой стороны фрейма.

Если задать `nDockBarID = 0`, то сторона может быть любой из числа предварительно определенных в функциях `CControlBar::EnableDocking()` и `CFrameWnd:: EnableDocking()`.

Установка панели инструментов в плавающее состояние выполняется функцией:

```
void CFrameWnd::FloatControlBar(
   CControlBar * pBar,             // Указатель на панель инструментов
   CPoint point,                   // Координаты положения левого
                                   // верхнего угла панели инструментов
   DWORD dwStyle = CBRS_ALIGN_TOP); // Стиль положения панели
                                    //инструментов внутри фреймового окна
```

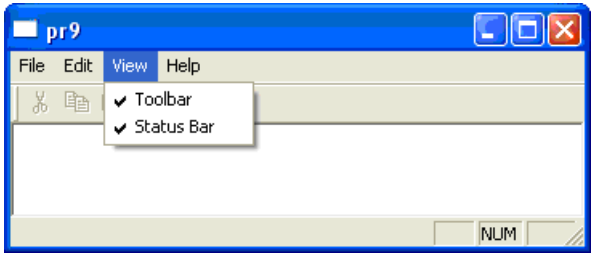


Рис. 9.2. Главное окно с панелью инструментов

Стили положения панели инструментов (`dwStyle`) бывают такими:

- `CBRS_ALIGN_TOP`, `CBRS_ALIGN_BOTTOM` — вертикальное положение;
- `CBRS_ALIGN_LEFT`, `CBRS_ALIGN_RIGHT` — горизонтальное положение.

Результаты работы программы показаны на рис. 9.2.

9.1.2. Строка состояния (*StatusBar*)

При использовании строки состояния в исходный код будет автоматически добавлен код, отвечающий за работу строки состояния. Изменения в файлах приведены в листингах 9.3 и 9.4.

Листинг 9.3. Изменения в файле MainFrm.h для поддержки работы строки состояния

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:  // control bar embedded members
   CStatusBar  m_wndStatusBar;
// ...
};
```

Листинг 9.4. Изменения в файле MainFrm.cpp для поддержки работы строки состояния

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_CREATE()
   ON_WM_SETFOCUS()
END_MESSAGE_MAP()

static UINT indicators[] =
{
   ID_SEPARATOR,                  // status line indicator
   ID_INDICATOR_CAPS,
```

```
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    // ...
    if(!m_wndStatusBar.Create(this) ||
       !m_wndStatusBar.SetIndicators(indicators,
                                     sizeof(indicators)/sizeof(UINT)))
    {
       TRACE0("Failed to create status bar\n");
       return -1;                      // fail to create
    }
    // ...
    return 0;
}
```

Класс для работы со строкой статуса определен так:

```
class CStatusBar : public CControlBar
class CControlBar : public CWnd
```

Строка состояния находится внизу окна, выглядит так, как показано на рис. 9.3, и определяется в массиве `indicators[]`:

Ready (поле, где появляются подсказки) | **CAP** | **NUM** | **SCRL**

Идентификаторы полей строки состояния могут принимать такие значения:

- `ID_SEPARATOR` — вертикальная черта (|), отделяющая поле подсказок (**Ready**) от начала поля индикаторов (**CAP**);
- `ID_INDICATOR_CAPS` — поле **CAP**. Появляется при нажатой клавише <Caps Lock>;
- `ID_INDICATOR_NUM` — поле **NUM**. Появляется при нажатой клавише <Num Lock>;
- `ID_INDICATOR_SCRL` — поле **SCRL**. Появляется при нажатой клавише <Scroll Lock>.

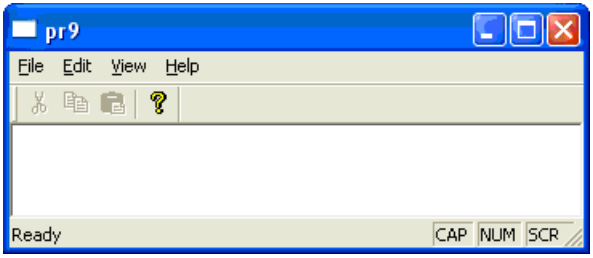


Рис. 9.3. Главное окно со строкой состояния

Создание строки состояния выполняется с помощью функции:

```
virtual BOOL CStatusBar:: Create(   // 0 - ошибка
   CWnd* pParentWnd,                // Указатель на родительское окно
   DWORD dwStyle = WS_CHILD |       // Стиль окна строки состояния
                   WS_VISIBLE | CBRS_BOTTOM,
   UINT nID = AFX_IDW_STATUS_BAR); // Идентификатор окна строки состояния
```

Стили окна строки состояния (`dwStyle`) могут быть такими:

- `CBRS_TOP` — строка состояния находится в верхней части окна фрейма;
- `CBRS_BOTTOM` — строка состояния находится в нижней части окна фрейма;
- `CBRS_NOALIGN` — строка состояния не перемещается, когда родительское окно меняет размеры.

Установка индикаторов каждой области строки состояния выполняется функцией:

```
BOOL CStatusBar::SetIndicators(    // 0 - ошибка
   const UINT* lpIDArray,          // Массив индикаторов
   int nIDCount);                  // Число элементов в массиве индикаторов
```

9.1.3. Добавление кнопок на панель инструментов

Добавим новое меню **Menu** с пунктами **Menu1** и **Menu2** (рис. 9.4) и обработку этих пунктов в класс фреймового окна (рис. 9.5). Изменения в файлах приведены в листингах 9.5—9.7.

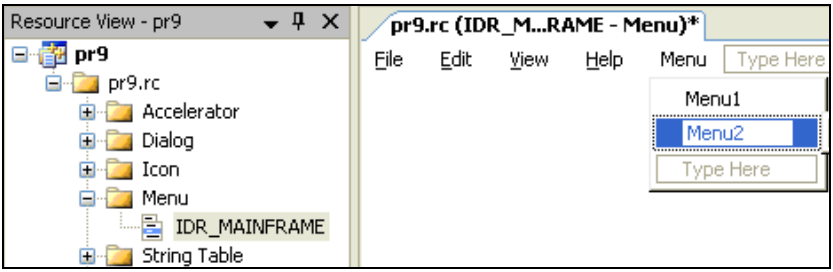
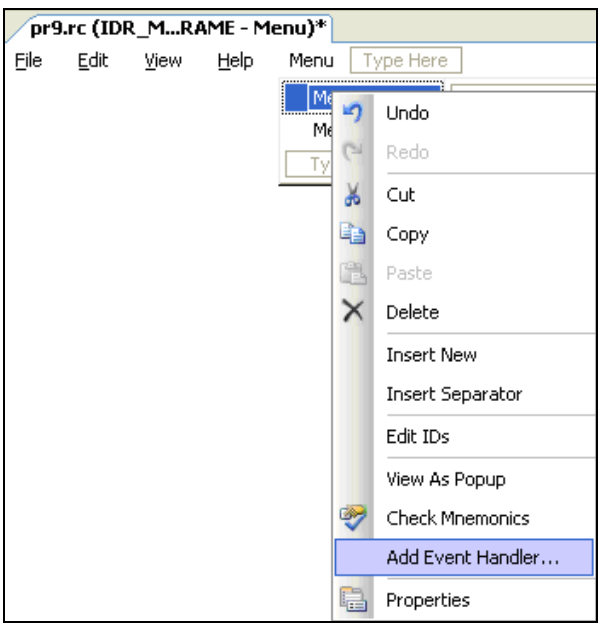


Рис. 9.4. Добавление нового меню

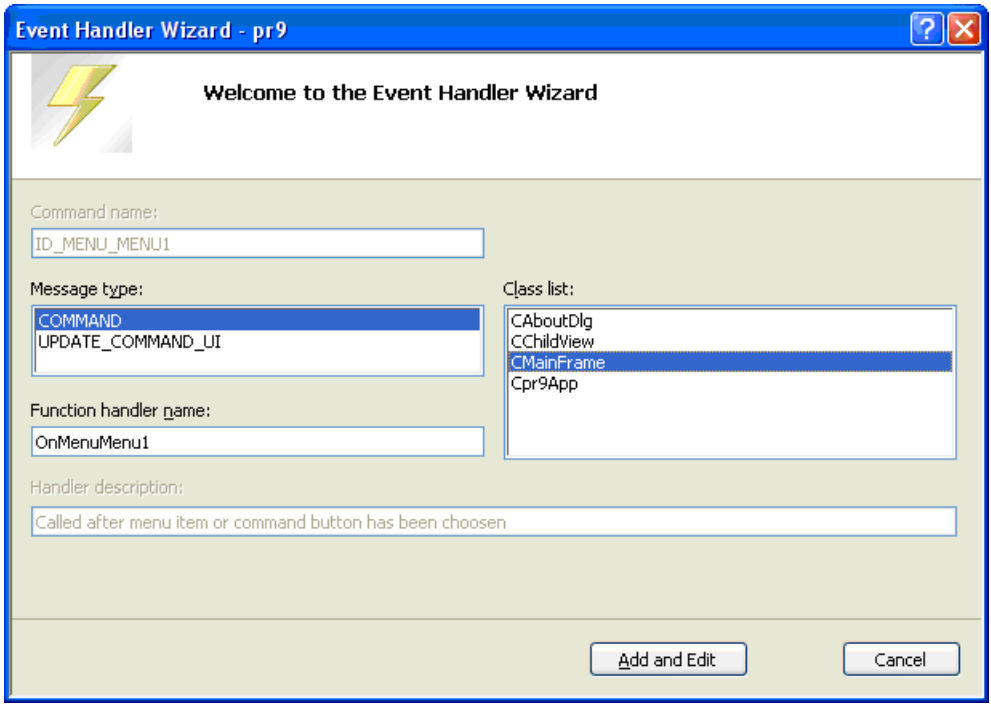
Листинг 9.5. Изменения в файле Resource.h для работы с новыми пунктами меню

```
// ...
#define ID_MENU_MENU1                   32771
#define ID_MENU_MENU2                   32772
// ...
```



а

Рис. 9.5. Добавление обработки сообщения меню (*а*)



б

Рис. 9.5. Функция обработки сообщения меню (*б*)

Листинг 9.6. Изменения в файле MainFrm.h для работы с новыми пунктами меню

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnMenuMenu1();
public:
   afx_msg void OnMenuMenu2();
};
```

Листинг 9.7. Изменения в файле MainFrm.cpp для работы с новыми пунктами меню

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_SETFOCUS()
    ON_COMMAND(ID_MENU_MENU1, &CMainFrame::OnMenuMenu1)
    ON_COMMAND(ID_MENU_MENU2, &CMainFrame::OnMenuMenu2)
END_MESSAGE_MAP()
// ...
void CMainFrame::OnMenuMenu1()
{
    // TODO: Add your command handler code here
    AfxMessageBox("Menu1");

}
void CMainFrame::OnMenuMenu2()
{
    // TODO: Add your command handler code here
    AfxMessageBox("Menu2");
}
```

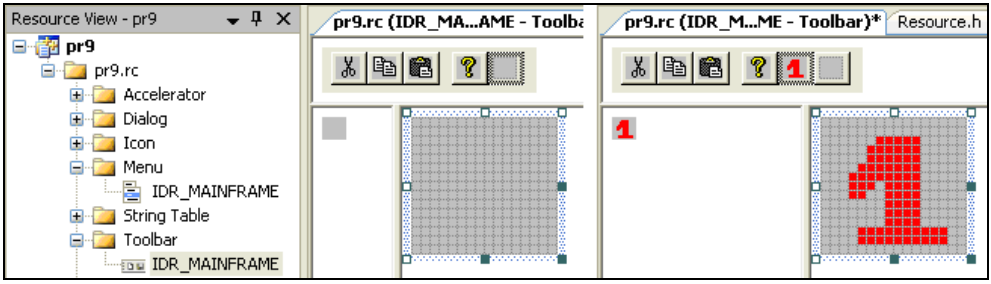


Рис. 9.6. Добавление кнопки **1** в панель инструментов

Откроем в окне ресурсов папку **Toolbar** и создадим новую кнопку **1** (рис. 9.6). Теперь надо связать ее с пунктом меню **Menu1**. Для этого изменим в ее свойствах идентификатор **ID** на `ID_MENU_MENU1` (рис. 9.7). Аналогично

создадим кнопку **2** и свяжем ее с `ID_MENU_MENU2` (рис. 9.8). Результаты работы программы показаны на рис. 9.9. При выборе пункта меню **Menu1** или при нажатии кнопки **1** выведется соответствующее сообщение. Аналогично с меню **Menu2** и кнопкой **2**.

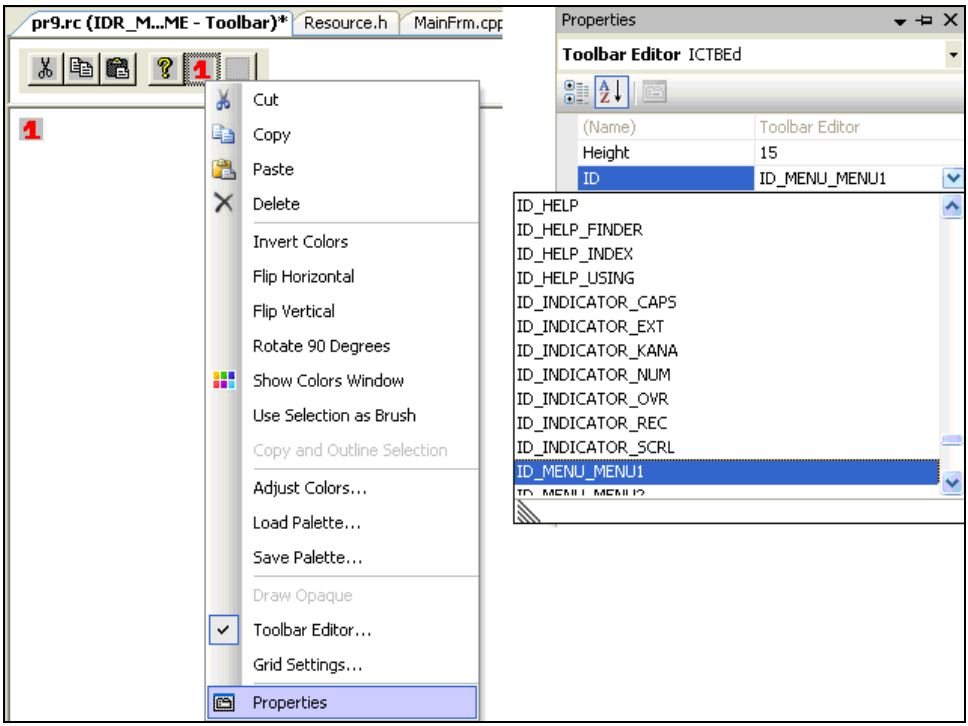


Рис. 9.7. Привязка кнопки **1** к **Menu1**

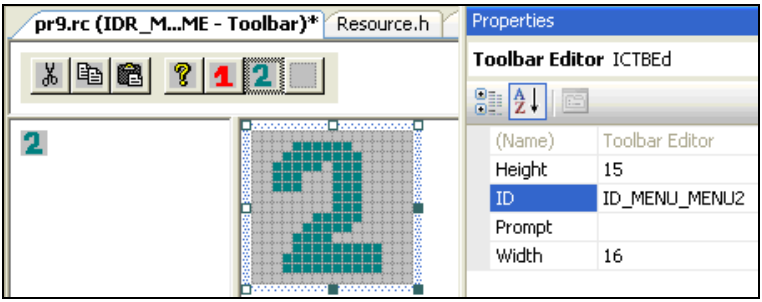
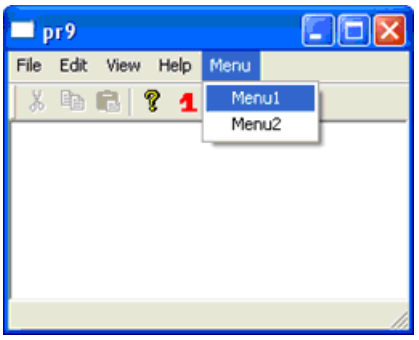
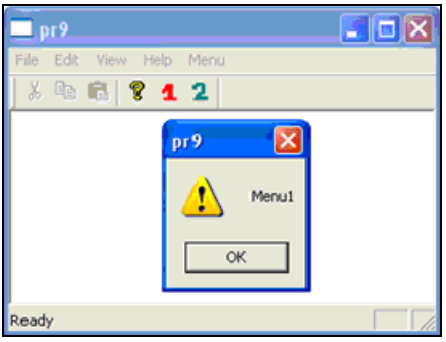


Рис. 9.8. Добавление кнопки **2** в панель инструментов



а



б

Рис. 9.9. Выбор пункта меню (*а*) и обработка его сообщения (*б*)

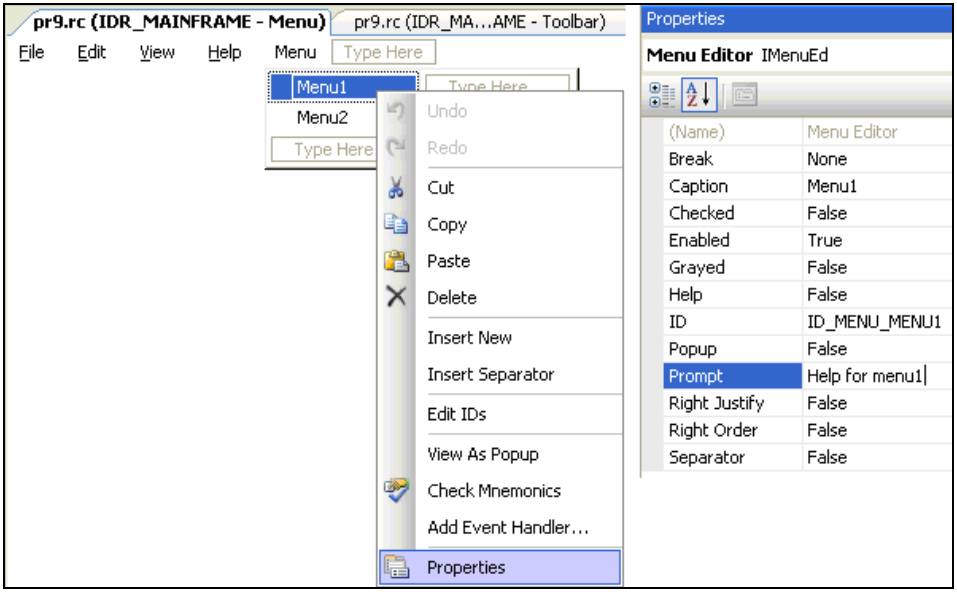


Рис. 9.10. Добавление подсказки, появляющейся в строке статуса

Примечание

В окне ресурсов **Toolbar** можно перетаскивать кнопки ("зацепив" их левой кнопкой мыши) и объединять (или разъединять) их с группой. Чтобы удалить кнопку из панели инструментов, надо "зацепить" ее левой кнопкой мыши и переместить за пределы окна панели инструментов (например, вверх). Удалить кнопку с помощью контекстного меню нельзя, хотя там и есть команда **Delete**.

Добавим появление подсказки для кнопок и меню, которая будет появляться в строке статуса. Для этого в окне ресурсов **Menu** откроем окно свойств для **Menu1** и в поле **Prompt** (Подсказка) напишем текст подсказки `Help for menu1` (рис. 9.10). Аналогично добавим подсказку для **Menu2** (`Help for menu2`). Теперь, при работе программы, если пользователь выделил пункт меню (**Menu1** или **Menu2**) или навел курсор на кнопку **1** или кнопку **2**, в строке статуса появится соответствующая подсказка (рис. 9.11).

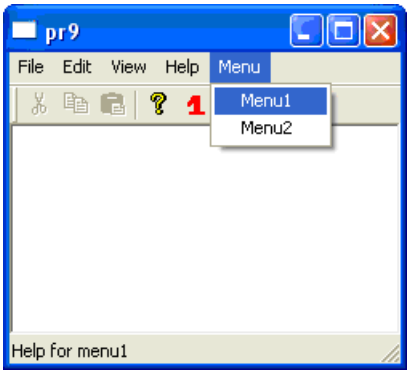


Рис. 9.11. Работа подсказки

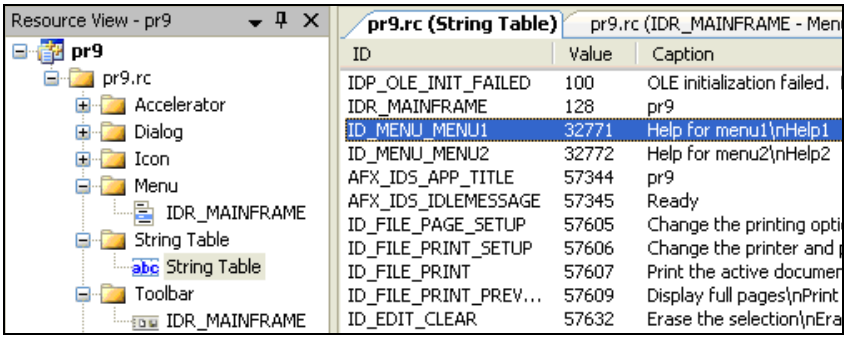


Рис. 9.12. Добавление всплывающей подсказки в окне ресурсов **String Table**

Добавим всплывающую подсказку для кнопок **1** и **2**. Это можно сделать, добавив в окне свойств для **Menu1** в поле **Prompt** в конец строки разделитель `'\n'` и текст подсказки (`"Help1"`): `Help for menu1\nHelp1` или в окне ресурсов **String Table** (рис. 9.12). Между разделителем и текстом подсказки *не*

должно быть пробелов. Аналогично для кнопки **2** — `Help for menu2\nHelp2`. Теперь если при работе программы навести курсор на кнопку **1** или на кнопку **2**, будет появляться соответствующая всплывающая подсказка (рис. 9.13).

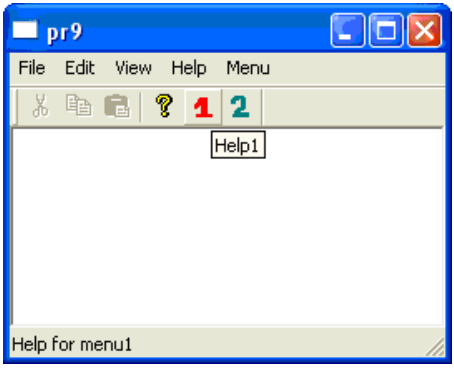


Рис. 9.13. Работа всплывающей подсказки

9.1.4. Отображение и скрытие кнопки на панели инструментов

Добавим еще одно меню **Format | ViewBut1** (рис. 9.14), при выборе которого будет показываться или скрываться кнопка **1** (если меню помечено "галочкой", то кнопка **1** будет видна, если нет — то скрыта). Зададим в окне свойств для меню **ViewBut1** в поле **Checked** (Отметить) значение `True`, чтобы этот пункт меню был изначально помечен "галочкой" (рис. 9.15).

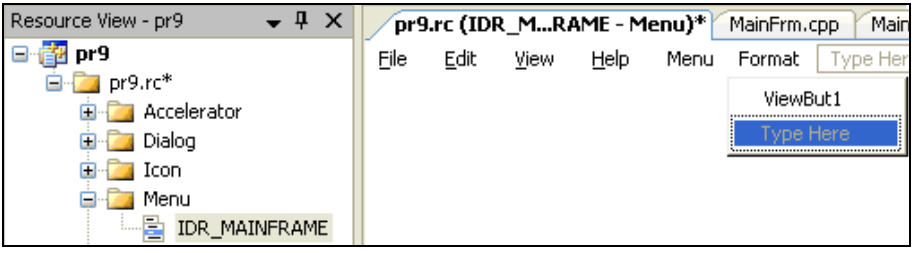


Рис. 9.14. Добавление меню для отображения и скрытия кнопки **1**

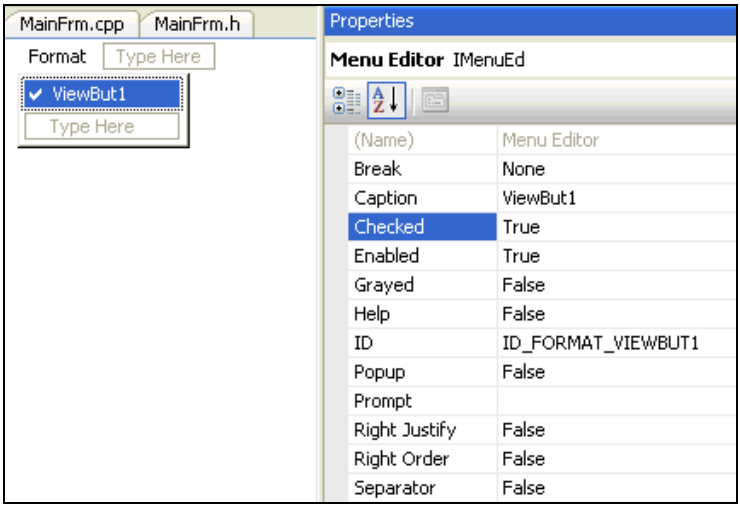


Рис. 9.15. Изменение начального вида меню

Добавим функцию обработки этого пункта меню в класс окна фрейма. Изменения в файлах приведены в листингах 9.8—9.10.

Листинг 9.8. Изменения в файле Resource.h для работы с пунктом меню, с помощью которого отображается или скрывается кнопка 1

```
// ...
#define ID_FORMAT_VIEWBUT1                    32775
// ...
```

Листинг 9.9. Изменения в файле MainFrm.h для работы с пунктом меню, с помощью которого отображается или скрывается кнопка 1

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnFormatViewbut1();
};
```

Листинг 9.10. Изменения в файле MainFrm.cpp для работы с пунктом меню, с помощью которого отображается или скрывается кнопка 1

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   // ...
   ON_COMMAND(ID_FORMAT_VIEWBUT1, &CMainFrame::OnFormatViewbut1)
END_MESSAGE_MAP()
// ...

void CMainFrame::OnFormatViewbut1()
{
   // TODO: Add your command handler code here
}
```

Теперь сделаем так, чтобы при выборе этого пункта меню "галочка" пропадала или появлялась. Если меню отмечено "галочкой", кнопка **1** будет видна, если без "галочки" — кнопка **1** будет скрыта. Изменения в файлах приведены в листингах 9.11 и 9.12.

Листинг 9.11. Изменения в файле MainFrm.h для отображения и скрытия кнопки 1

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   CMenu  *pmenu,              // Меню приложения
          *psubm;              // Выпадающее меню Format | ViewBut1
};
```

Листинг 9.12. Изменения в файле MainFrm.cpp для отображения и скрытия кнопки 1

```
// ...
void CMainFrame::OnFormatViewbut1()
```

```
{
   // TODO: Add your command handler code here
   static int ch = 1;              // Вначале меню помечено "галочкой"
   pmenu = this->GetMenu();        // Получить указатель меню приложения
   // Получить указатель на выпадающее меню "Format"
   psubm = pmenu->GetSubMenu(5);
   // Получить ссылку на встроенный объект класса CToolBarCtrl
   CToolBarCtrl &tbc = this->m_wndToolBar.GetToolBarCtrl();

   if(!ch)                    // Если меню не было помечено
   {
      // Пометить меню
      psubm->CheckMenuItem(ID_FORMAT_VIEWBUT1, MF_CHECKED);
      ch = 1;
      // Показать кнопку 1
      tbc.SetState(ID_MENU_MENU1, TBSTATE_ENABLED);
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
   else                       // Если меню было помечено
   {
      // Снять пометку
      psubm->CheckMenuItem(ID_FORMAT_VIEWBUT1, MF_UNCHECKED);
      ch = 0;
      // Скрыть кнопку1
      tbc.SetState(ID_MENU_MENU1, TBSTATE_HIDDEN);
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
}
```

Функции `GetMenu()`, `GetSubMenu()` и `CheckMenuItem()` были описаны в *разд. 4.1.3* и *4.1.5*. Выбран индекс меню 5 (в `GetSubMenu(5)`), т. к. индексация пунктов меню начинается с 0 (**File** (0), **Edit** (1), **View** (2), **Help** (3), **Menu** (4), **Format** (5)).

Функция получения ссылки на встроенный объект класса `CToolBarCtrl` определена как:

```
CToolBarCtrl& CToolBar::GetToolBarCtrl() const;
```

Класс `CToolBarCtrl` позволяет воспользоваться многими дополнительными функциями для работы с панелью инструментов:

```
class CToolBarCtrl : public CWnd
```

С любой кнопкой панели инструментов связана структура:

```
typedef struct _TBBUTTON
{
   int iBitmap;    // Индекс иконки (для сепаратора(разделителя) = 0)
   int idCommand;  // Идентификатор команды, который будет передаваться
                   // родителю с сообщением WM_COMMAND,
                   // когда пользователь нажмет на эту кнопку
                   // (для сепаратора = 0)
   BYTE fsState;   // Комбинация флагов состояния кнопки
   BYTE fsStyle;   // Комбинация стилей кнопки
   DWORD dwData;   // Дополнительные данные, связанные с кнопкой
   int iString;    // Индекс строки для надписи на кнопке
                   // (если кнопка без надписи = -1)
} TBBUTTON;
```

Значения флагов состояния кнопки (`fsState`) могут быть следующими:

- `TBSTATE_CHECKED` — кнопка имеет стиль `TBSTYLE_CHECKED` (кнопка с фиксацией) и отображается в нажатом состоянии;
- `TBSTATE_ENABLED` — кнопка доступна для пользователя. Если этот флаг не установлен, то кнопка недоступна и отображается серым цветом;
- `TBSTATE_HIDDEN` — кнопка невидима;
- `TBSTATE_INDETERMINATE` — кнопка отображается серым цветом (но может оставаться доступной);
- `TBSTATE_PRESSED` — кнопка отображается в нажатом состоянии;
- `TBSTATE_WRAP` — все следующие кнопки после этой будут находиться на следующей строке (этот флаг можно использовать только с `TBSTATE_ENABLED`).

Значения стилей кнопки (`fsStyle`) (описание стилей *см. разд. 9.1.1* `CToolBar::CreateEx()`) могут быть такими:

- `TBSTYLE_BUTTON`;
- `TBSTYLE_CHECK`;
- `TBSTYLE_CHECKGROUP`;
- `TBSTYLE_GROUP`;
- `TBSTYLE_SEP`.

Информацию о кнопке можно получить с помощью функции:

```
BOOL CToolBarCtrl::GetButton(    // 0 - ошибка
   int nIndex,                   // Индекс кнопки
   LPTBBUTTON lpButton) const;   // Указатель на структуру TBBUTTON
                                 // для заполнения
```

Функция, устанавливающая состояние кнопки, определена как:

```
BOOL CToolBarCtrl::SetState(     // 0 - ошибка
   int nID,                      // Идентификатор кнопки
   UINT nState);                 // Флаг состояния кнопки
                                 // (могут комбинироваться)
                                 // (см. fsState структуры TBBUTTON)
```

При скрытии кнопки **1** должен измениться (уменьшиться) размер панели инструментов, а при восстановлении кнопки **1** размер панели инструментов должен увеличиться. Сообщение об изменении размеров панели инструментов обрабатывается только при обработке сообщения об изменении размеров родительского (фреймового) окна. Поэтому чтобы корректно изменить размеры панели инструментов, можно или свернуть и развернуть окно программы, или программно вызвать обработку сообщения об изменении размеров фрейма (`this->SendMessageA(WM_SIZE)`).

Функция, посылающая сообщение в объект `CWnd` (непосредственно вызывает оконную процедуру и не выходит из нее, пока не обработается сообщение), выглядит так:

```
LRESULT CWnd::SendMessage(    // Результат обработки сообщения; величина
                              // зависит от посланного сообщения
   UINT message,              // Посылаемое сообщение
   WPARAM wParam = 0,         // Дополнительные параметры
   LPARAM lParam = 0);
```

Того же можно добиться, если вместо `this->SendMessageA(WM_SIZE)` вызвать `this->RecalcLayout()`.

Функция пересчета позиции элемента управления определена как:

```
virtual void CFrameWnd::RecalcLayout(
   BOOL bNotify = TRUE);      // Получит ли элемент управления (панель
                              // инструментов) сообщение об изменениях
```

Результаты работы программы показаны на рис. 9.16.

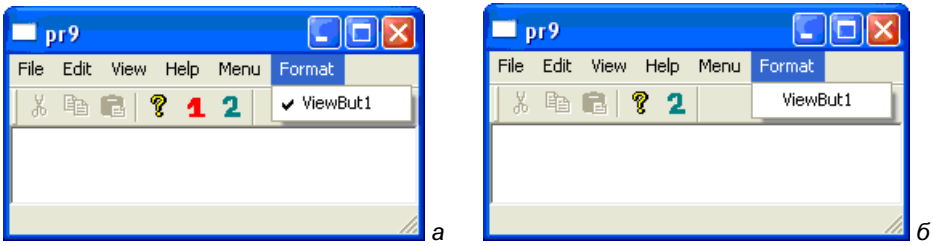


Рис. 9.16. Результат работы программы с отображением (*а*) и скрытием (*б*) кнопки **1**

9.1.5. Удаление и добавление кнопок на панели инструментов

Добавим в меню пункт **Format | Add2But2** (аналогично **Format | ViewBut1**, только в свойствах этот пункт меню не надо делать "помеченным" — он будет обычным) и добавим его обработку в класс фреймового окна (рис. 9.17). Изменения в файлах приведены в листингах 9.13—9.15.

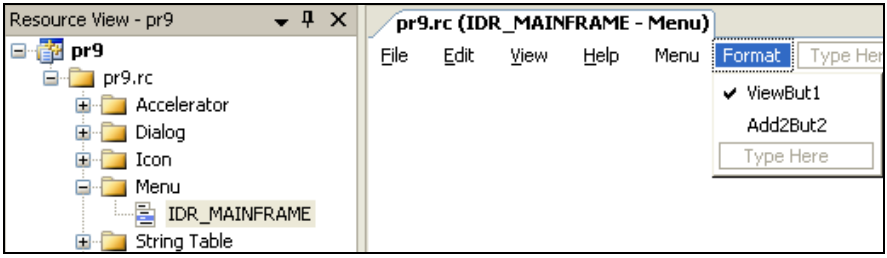


Рис. 9.17. Меню для добавления и удаления кнопок в панели инструментов

Листинг 9.13. Изменения в файле Resource.h для работы с меню, позволяющим добавлять и удалять кнопку на панели инструментов

```
// ...
#define ID_FORMAT_ADD2BUT2                      32776
// ...
```

Листинг 9.14. Изменения в файле MainFrm.h для работы с меню, позволяющим добавлять и удалять кнопку на панели инструментов

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnFormatAdd2but2();
};
```

Листинг 9.15. Изменения в файле MainFrm.cpp для работы с меню, позволяющим добавлять и удалять кнопку на панели инструментов

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   // ...
   ON_COMMAND(ID_FORMAT_ADD2BUT2, &CMainFrame::OnFormatAdd2but2)
END_MESSAGE_MAP()
// ...
void CMainFrame::OnFormatAdd2but2()
{
   // TODO: Add your command handler code here
}
```

Теперь сделаем так, чтобы при выборе этого пункта меню оно помечалось "галочкой" и в панели инструментов появлялись две серые кнопки **2**: одна вначале панели инструментов, другая — в конце. Если у меню снять "галоч-

ку", эти кнопки будут удалены из панели инструментов. Изменения в файлах приведены в листингах 9.16 и 9.17.

Листинг 9.16. Изменения в файле MainFrm.h для добавления и удаления кнопок на панели инструментов

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnFormatAdd2but2();
   TBBUTTON tb2;                  // Информация о кнопке 2
};
```

Листинг 9.17. Изменения в файле MainFrm.cpp для добавления и удаления кнопок на панели инструментов

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // ...
   // Получить ссылку на встроенный объект класса CToolBarCtrl
   CToolBarCtrl &tbc = this->m_wndToolBar.GetToolBarCtrl();
   // Получить данные о кнопке 2
   tbc.GetButton(tbc.CommandToIndex(ID_MENU_MENU2), &tb2);

   return 0;
}

void CMainFrame::OnFormatAdd2but2()
{
   // TODO: Add your command handler code here
   static int ch = 0;              // Вначале меню не помечено "галочкой"
   pmenu = this->GetMenu();        // Получить указатель меню приложения
   // Получить указатель на выпадающее меню "Format"
```

```
   psubm = pmenu->GetSubMenu(5);
   // Получить ссылку на встроенный объект класса CToolBarCtrl
   CToolBarCtrl &tbc = this->m_wndToolBar.GetToolBarCtrl();

   if(!ch)                          // Если меню не было помечено
   {
      // Пометить меню
      psubm->CheckMenuItem(ID_FORMAT_ADD2BUT2, MF_CHECKED);
      ch = 1;
      // Установить для кнопки 2 "серый" стиль
      tb2.fsState  = TBSTATE_INDETERMINATE;
      tbc.InsertButton(7, &tb2);    // Вставить кнопку 2 в конец
      tbc.InsertButton(0, &tb2);    // Вставить кнопку 2 в начало
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
   else
   {
      // Снять "галочку"
      psubm->CheckMenuItem(ID_FORMAT_ADD2BUT2, MF_UNCHECKED);
      ch = 0;
      // Удалить кнопки 2 с конца и из начала
      tbc.DeleteButton(8);
      tbc.DeleteButton(0);
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
}
```

Функция получения индекса кнопки в панели инструментов по ее идентификатору определена как:

```
UINT CToolBarCtrl::CommandToIndex(   // Индекс кнопки или 0 - если
                                     // с кнопкой не ассоциируется
                                     // никакой идентификатор
   UINT nID) const;                  // Идентификатор кнопки
```

Добавление новой кнопки в панель инструментов выполняется функцией:

```
BOOL CToolBarCtrl::InsertButton(     // 0 - ошибка
   int nIndex,              // Индекс кнопки в панели инструментов, слева
                            // от которой (перед которой) будет добавлена
                            // новая кнопка
   LPTBBUTTON lpButton);    // Указатель на структуру TBBUTTON
                            // с информацией о добавляемой кнопке
```

В этой программе кнопки панели инструментов имеют следующие индексы: **Cut** — 0; **Copy** — 1; **Paste** — 2; (|) (сепаратор) — 3; (**?**) — 4; (**1**) — 5; (**2**) — 6; (|) — 7.

Для вставки новой кнопки в начало панели инструментов можно было воспользоваться идентификатором кнопки **Cut**:

```
tbc.InsertButton(CommandToIndex(ID_EDIT_CUT), &tb2);
```

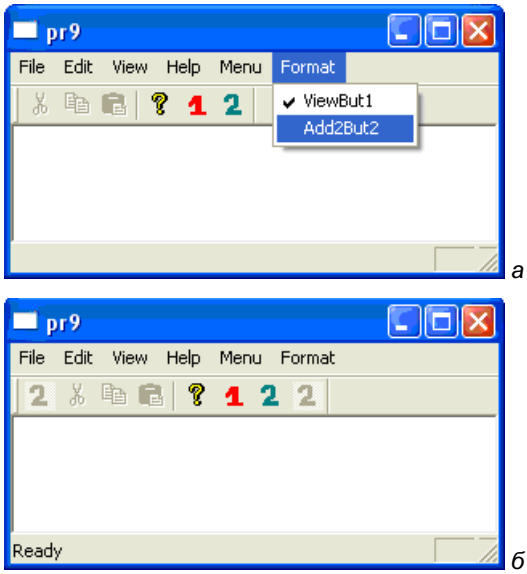


Рис. 9.18. Выбор меню для добавления кнопок на панель инструментов (*а*) и добавление кнопок в начало и конец панели инструментов (*б*)

Для всех сепараторов (|) идентификаторы равны 0. Поэтому при использовании функции `CommandToIndex()` для вставки в конец панели инструментов (`tbc.InsertButton(CommandToIndex(0), &tb2)`) новая кнопка была бы

добавлена перед первым сепаратором, т. е. после кнопки **Paste**. Порядок добавления кнопок тоже важен. Если сначала добавить кнопку в начало, то при добавлении кнопки в конец надо будет задать индекс 8, а не 7 (т. к. количество кнопок уже было бы увеличено).

Удаление кнопки из панели инструментов выполняется функцией:

```
BOOL CToolBarCtrl::DeleteButton(      // 0 - ошибка
   int nIndex);                       // Индекс удаляемой кнопки
```

Результаты работы программы показаны на рис. 9.18.

9.1.6. Добавление и удаление своей панели инструментов

Добавим свою панель инструментов. Для этого нужно в окне **Resource View** вызвать контекстное меню для папки **Toolbar** и выполнить команду **Insert Toolbar** (рис. 9.19).

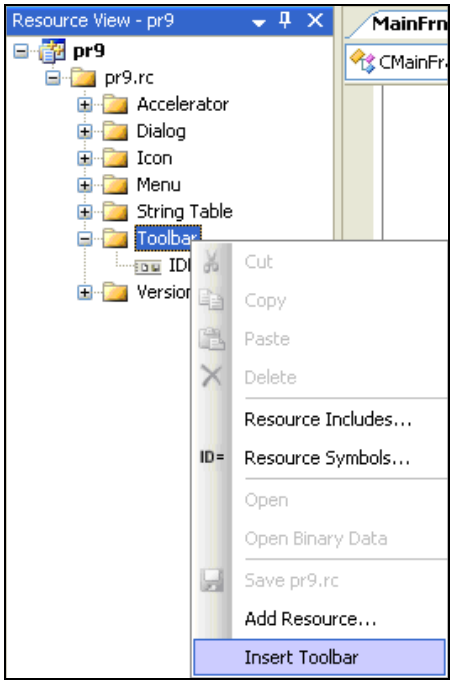


Рис. 9.19. Добавление новой панели инструментов

Создадим на новой панели инструментов кнопку **3** (для простоты в качестве обработчика в свойствах кнопки зададим `ID_MENU_MENU1`, т. е. она будет работать аналогично кнопке **1** основной панели инструментов) (рис. 9.20).

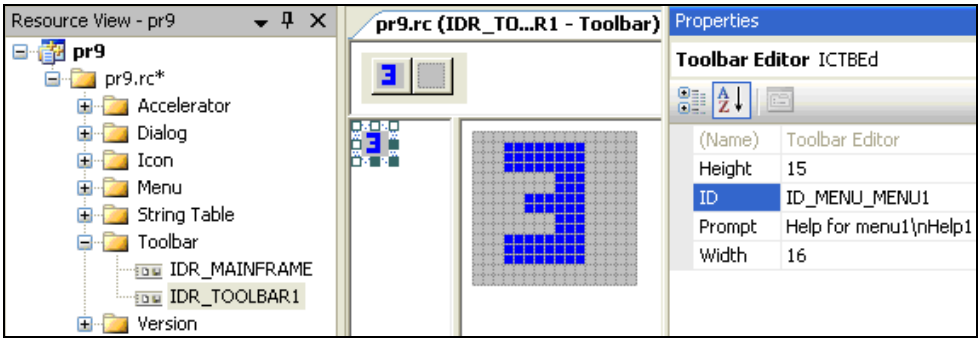


Рис. 9.20. Добавление кнопки в новую панель инструментов

Добавим в меню пункт **Format | AddTB2** (аналогично **Format | Add2But2**) и добавим его обработку в класс фреймового окна (рис. 9.21). Изменения в файлах приведены в листингах 9.18—9.20.

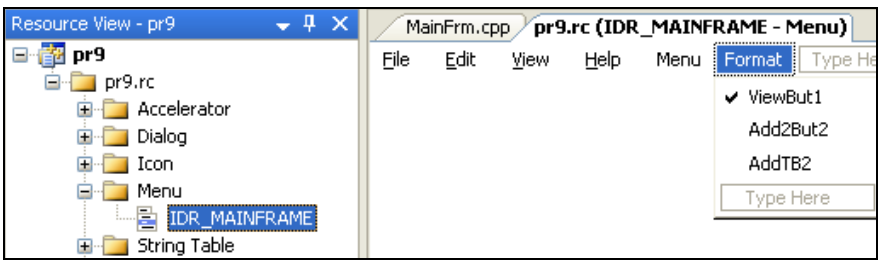


Рис. 9.21. Добавление меню для добавления и удаления новой панели инструментов

Листинг 9.18. Изменения в файле Resource.h для работы со второй панелью инструментов

```
// ...
#define IDR_TOOLBAR1                    130
#define ID_FORMAT_ADDTB2                32778
// ...
```

Листинг 9.19. Изменения в файле MainFrm.h для работы со второй панелью инструментов

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnFormatAddtb2();
};
```

Листинг 9.20. Изменения в файле MainFrm.cpp для работы со второй панелью инструментов

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   // ...
   ON_COMMAND(ID_FORMAT_ADDTB2, &CMainFrame::OnFormatAddtb2)
   END_MESSAGE_MAP()
// ...
void CMainFrame::OnFormatAddtb2()
{
   // TODO: Add your command handler code here
}
```

Теперь сделаем так, чтобы при выборе этого пункта меню оно помечалось "галочкой" и у левой стороны фреймового окна появлялась новая панель инструментов. Если у меню снять пометку — новая панель инструментов будет удалена. Изменения в файлах приведены в листингах 9.21 и 9.22.

Листинг 9.21. Изменения в файле MainFrm.h для добавления и удаления второй панели инструментов

```
// ...
class CMainFrame : public CFrameWnd
{
```

```
// ...
public:
   afx_msg void OnFormatAddtb2();
   CToolBar myToolBar;              // Новая панель инструментов
};
```

Листинг 9.22. Изменения в файле MainFrm.cpp для добавления и удаления второй панели инструментов

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // ...
   // Создание новой "невидимой" панели инструментов у левой стороны
   // фреймового окна
   if(!myToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | CBRS_LEFT |
                          CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
                          CBRS_SIZE_DYNAMIC) ||
      !myToolBar.LoadToolBar(IDR_TOOLBAR1))
   {
      TRACE0("Failed to create toolbar\n");
      return -1;
   }
   myToolBar.EnableDocking(CBRS_ALIGN_ANY);
   EnableDocking(CBRS_ALIGN_ANY);
   DockControlBar(&myToolBar);

   return 0;
}
// ...
void CMainFrame::OnFormatAddtb2()
{
   // TODO: Add your command handler code here
   static int ch=0;              // Вначале меню не помечено "галочкой"
   pmenu = this->GetMenu();      // Получить указатель на меню приложения
```

```
   // Получить указатель на выпадающее меню "Format"
   psubm = pmenu->GetSubMenu(5);
   if(!ch)                          // Если меню не было помечено
   {
      // Пометить меню
      psubm->CheckMenuItem(ID_FORMAT_ADDTB2, MF_CHECKED);
      ch = 1;
      // Показать новую панель
      this->ShowControlBar(&myToolBar, 1, 0);
   }
   else
   {
      // Снять "галочку"
      psubm->CheckMenuItem(ID_FORMAT_ADDTB2, MF_UNCHECKED);
      ch = 0;
      // Скрыть новую панель
      this->ShowControlBar(&myToolBar, 0, 0);
   }
}
```

Функция определения видимости контрольной панели выглядит так:

```
void CFrameWnd::ShowControlBar(
   CControlBar* pBar,     // Указатель на контрольную панель
   BOOL bShow,            // TRUE – показать панель, FALSE – скрыть
   BOOL bDelay);          // TRUE – перед выводом на экран нужна задержка
```

Можно не запоминать предыдущее состояние панели инструментов, а узнать и изменить его. Тогда это выглядело бы так:

```
void CMainFrame::OnFormatAddtb2()
{
   // ...
   if(!ch)
   {
      psubm->CheckMenuItem(ID_FORMAT_ADDTB2, MF_CHECKED);
      ch = 1;
   }
```

```
    else
    {
       psubm->CheckMenuItem(ID_FORMAT_ADDTB2, MF_UNCHECKED);
       ch = 0;
    }
    // Узнать, было ли окно видимым (его текущее состояние)
    DWORD style = myToolBar.GetStyle() & WS_VISIBLE ;
    BOOL show;
    if(style == 0)          // Было скрыто - надо показать
       show = 1;
    else                    // Было видимо - надо скрыть
       show = 0;
    this->ShowControlBar(&myToolBar, show, 0);
}
```

Функция получения стиля окна определена как:

```
DWORD CWnd::GetStyle() const;
```

Результат работы программы показан на рис. 9.22.

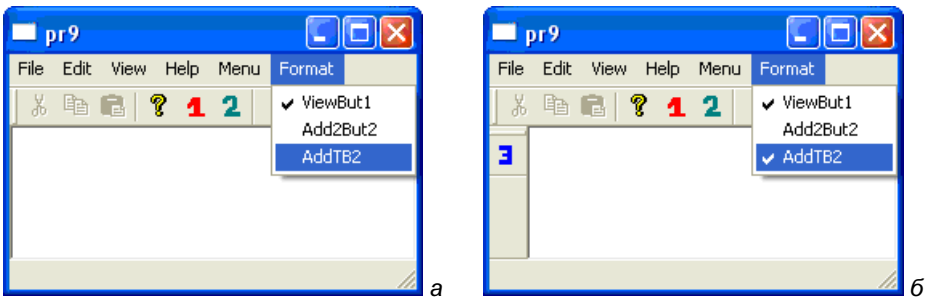


Рис. 9.22. Результат работы программы с выбором меню для добавления новой панели инструментов (*а*) и ее отображение (*б*)

9.1.7. Добавление новых полей в строку статуса

Сделаем так, чтобы при перемещении мыши в строке статуса показывались ее координаты. Для этого надо в ресурс таблицы строк (**String Table**) доба-

вить две новых строки (рис. 9.23). Добавляемые строки приведены в табл. 9.1. Изменения в файле приведены в листинге 9.23.

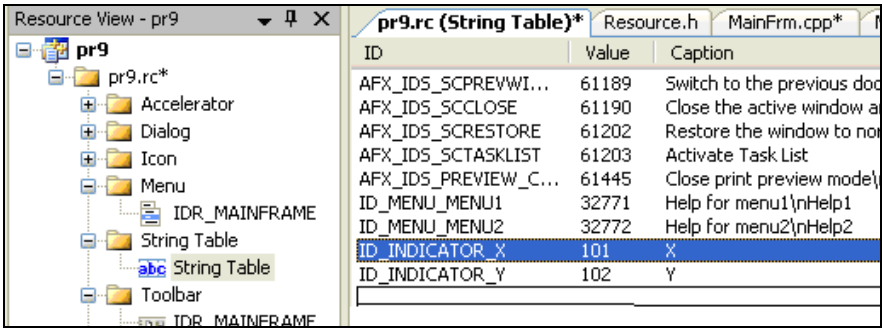


Рис. 9.23. Добавление новых ресурсов для строки статуса

Таблица 9.1. *Добавление новых ресурсов для строки статуса*

ID	Value	Caption
ID_INDICATOR_X	101	X
ID_INDICATOR_Y	102	Y

Листинг 9.23. Изменения в файле Resource.h для добавления новых полей в строку статуса

```
// ...
#define ID_INDICATOR_X          101
#define ID_INDICATOR_Y          102
// ...
```

Теперь надо создать и подключить два новых поля (`X` и `Y`) к строке статуса. Изменения в файле приведены в листинге 9.24.

Листинг 9.24. Изменения в файле MainFrm.cpp для добавления новых полей в строку статуса

```
// ...
static UINT indicators[] =
```

```
{
   ID_SEPARATOR,                    // status line indicator
   ID_INDICATOR_X,
   ID_INDICATOR_Y,
   ID_INDICATOR_CAPS,
   ID_INDICATOR_NUM,
   ID_INDICATOR_SCRL,
};
```

Результат работы программы показан на рис. 9.24.

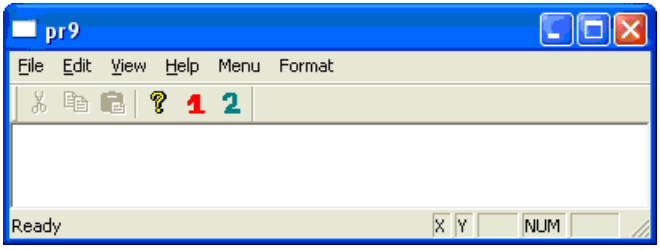


Рис. 9.24. Результат добавления новых полей в строку статуса

Размеры и стили панелей можно задавать и изменять. Расширим новые поля. Изменения в файле приведены в листинге 9.25.

Листинг 9.25. Изменения в файле MainFrm.cpp для изменения размеров полей строки статуса

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   // ...
   // Задание стилей двух новых полей строки статуса
   m_wndStatusBar.SetPaneInfo(
                              m_wndStatusBar.CommandToIndex(ID_INDICATOR_X),
                              ID_INDICATOR_X, SBPS_NORMAL, 50);
```

```
   m_wndStatusBar.SetPaneInfo(
                              m_wndStatusBar.CommandToIndex(ID_INDICATOR_Y),
                              ID_INDICATOR_Y, SBPS_NORMAL, 50);

   return 0;
}
```

Функция установки стилей для полей строки статуса выглядит так:

```
void CStatusBar::SetPaneInfo(
   int nIndex,         // Индекс поля
   UINT nID,           // Новое значение идентификатора поля
   UINT nStyle,        // Стиль поля
   int cxWidth );      // Ширина поля
```

Значения стилей полей строки статуса (`nStyle`) могут принимать значения:

- `SBPS_NOBORDERS` — вокруг поля нет трехмерной рамки;
- `SBPS_POPOUT` — текст рисуется с рамкой, которая выглядит приподнятой над поверхностью строки состояния;
- `SBPS_DISABLED` — запрещен вывод текста в поле;
- `SBPS_STRETCH` — "растягивающееся" поле. Может занимать ту область строки состояния, которую не занимают другие поля. Этот стиль может иметь только одно поле строки состояния;
- `SBPS_NORMAL` — обычное поле (без рамок и "растяжения").

Функция получения индекса поля по его идентификатору определена как:

```
int CStatusBar::CommandToIndex(   // Индекс поля или -1 - если
                                  // поле не найдено
   UINT nIDFind) const;           // Идентификатор поля
```

Теперь надо добавить в класс окна представления обработку сообщения о перемещении мыши (рис. 9.25):

- в окне **Class View** вызвать контекстное меню для класса `CChildView` и выполнить команду **Properties**;
- в окне **Properties** выбрать вкладку **Messages**;
- в списке сообщений найти `WM_MOUSEMOVE` и выбрать **<Add> OnMouseMove**.

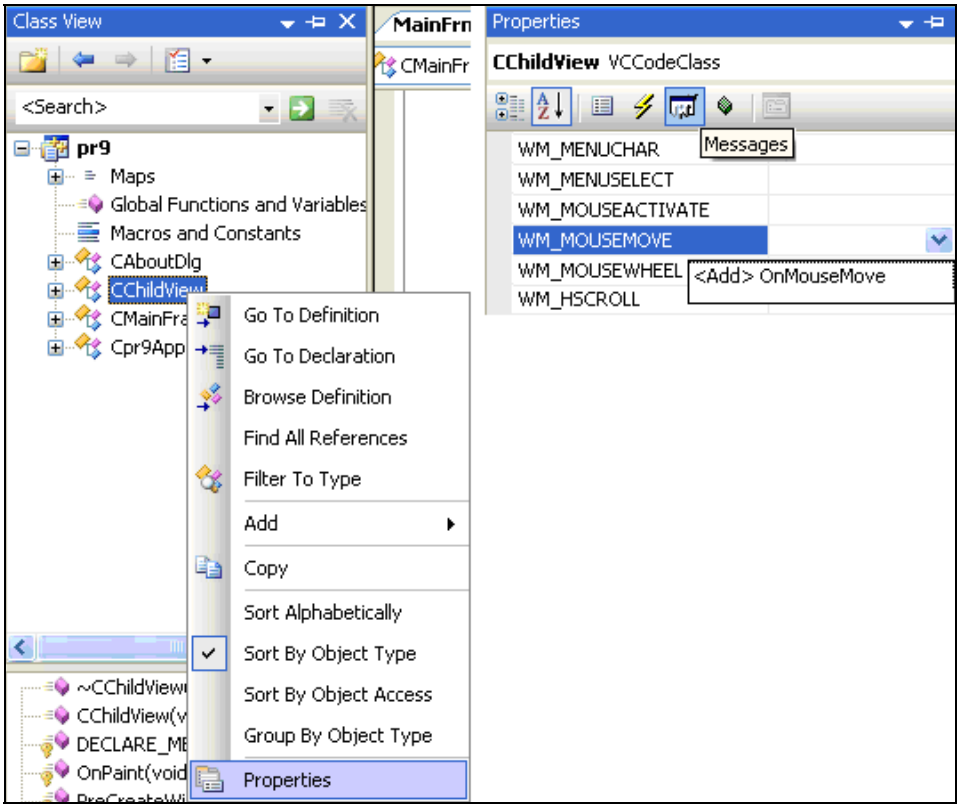


Рис. 9.25. Добавление в класс окна представления обработки сообщения о перемещении мыши

Изменения в файлах приведены в листингах 9.26 и 9.27.

Листинг 9.26. Изменения в файле ChildView.h для обработки сообщения о перемещении мыши

```
// ...
class CChildView : public CWnd
{
   // ...
public:
   afx_msg void OnMouseMove(UINT nFlags, CPoint point);
};
```

Листинг 9.27. Изменения в файле ChildView.cpp для обработки сообщения о перемещении мыши

```
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()
// ...
void CChildView::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default

   CWnd::OnMouseMove(nFlags, point);
}
```

Обработка сообщения о перемещении мыши выполняется функцией:

```
afx_msg void CWnd::OnMouseMove(
   UINT nFlags,            // Состояние кнопок в момент сообщения мыши
   CPoint point);          // Координаты курсора в момент сообщения мыши
```

Значения флагов кнопок (`nFlags`) могут быть такими:

- `MK_CONTROL` — в момент нажатия кнопки мыши была нажата клавиша <Ctrl>;
- `MK_LBUTTON` — была нажата клавиша левая кнопка мыши;
- `MK_MBUTTON` — была нажата клавиша средняя кнопка мыши;
- `MK_RBUTTON` — была нажата клавиша правая кнопка мыши;
- `MK_SHIFT` — в момент нажатия кнопки мыши была нажата клавиша <Shift>.

Значения флагов могут комбинироваться (т. е. одновременно более одного). Использовать значения флага можно, например, так:

```
if(nFlags && MK_SHIFT)
{
   // Была нажата клавиша <Shift>
}
```

Теперь надо добавить обработку этого сообщения так, чтобы текущие координаты отображались в строке статуса. Строка статуса является защищенным членом класса фреймового окна. Чтобы окно представления имело к ней доступ, надо переместить ее из защищенной секции в общую. Изменения в файлах приведены в листингах 9.28 и 9.29.

Листинг 9.28. Изменения в файле MainFrm.h для отображения текущих координат мыши в строке статуса

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:  // control bar embedded members
   // CStatusBar m_wndStatusBar;
public:
   CStatusBar m_wndStatusBar;
// ...
};
```

Листинг 9.29. Изменения в файле ChildView.cpp для отображения текущих координат мыши в строке статуса

```
// ...
#include "MainFrm.h"
// ...
void CChildView::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   // Получить указатель на фреймовое окно
   CMainFrame* pFrm = (CMainFrame* )theApp.m_pMainWnd;

   CString s;
   s.Format("x= %d",point.x);           // Записать в строку координату X
   pFrm->m_wndStatusBar.SetPaneText(    // Вывести ее в строку статуса
               pFrm->m_wndStatusBar.CommandToIndex(ID_INDICATOR_X), s);
   s.Format("y= %d",point.y);           // Записать в строку координату Y
```

```
    pFrm->m_wndStatusBar.SetPaneText(     // Вывести ее в строку статуса
                pFrm->m_wndStatusBar.CommandToIndex(ID_INDICATOR_Y), s);

    CWnd::OnMouseMove(nFlags, point);
}
```

Запись текста в панель строки состояния выполняется с помощью функции:

```
BOOL CStatusBar::SetPaneText(           // 0 - ошибка
    int nIndex,                 // Индекс поля, куда выводить текст
    LPCTSTR lpszNewText,        // Выводимый текст
    BOOL bUpdate = TRUE);       // TRUE - после записи текста область
                                // (поле) помечается как измененная
                                // для ее немедленного обновления
```

Результаты работы программы показаны на рис. 9.26.

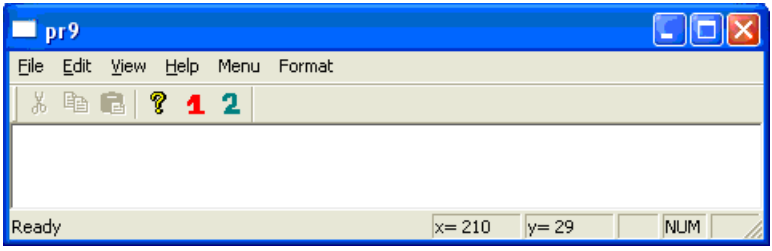


Рис. 9.26. Отображение текущих координат мыши в строке статуса

9.1.8. Изменение положения и цвета строки статуса

Перенесем строку состояния наверх и сделаем ее зеленого цвета. Изменения в файле приведены в листинге 9.30.

Листинг 9.30. Изменения в файле MainFrm.cpp для изменения цвета и положения строки состояния

```
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
   // ...
   if(//!m_wndStatusBar.Create(this) ||
      !m_wndStatusBar.Create(this, WS_CHILD | WS_VISIBLE | CBRS_TOP) ||
      !m_wndStatusBar.SetIndicators(indicators,
                                    sizeof(indicators)/sizeof(UINT)))
   {
      TRACE0("Failed to create status bar\n");
      return -1;      // fail to create
   }
   // ...
   // Меняем цвет строки статуса
   // Получить ссылку на объект CStatusBarCtrl
   CStatusBarCtrl &sbc = m_wndStatusBar.GetStatusBarCtrl();
   sbc.SetBkColor(RGB(0,255,0));          // Установить зеленый цвет

   return 0;
}
```

Функция получения ссылки на объект `CStatusBarCtrl`, дающий больше возможностей по управлению строкой статуса, следующая:

```
CStatusBarCtrl& CStatusBar::GetStatusBarCtrl() const; // Ссылка на объект
                                                      // CStatusBarCtrl
```

Цвет строки состояния изменяется с помощью функции:

```
COLORREF CStatusBarCtrl::SetBkColor(    // Предыдущее значение цвета
   COLORREF cr);                        // Устанавливаемое значение цвета
```

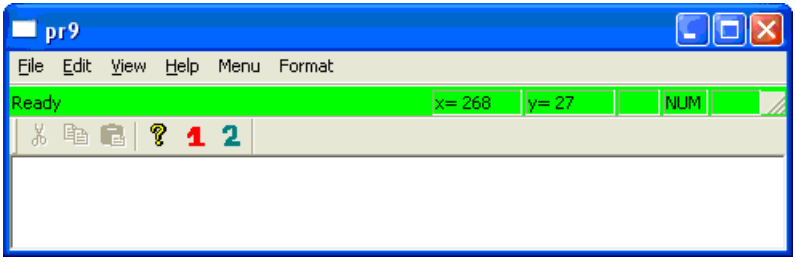


Рис. 9.27. Изменение положения и цвета строки статуса

Тип COLORREF был описан в *разд. 2.1.2*. Если в качестве параметра cr задать CLR_DEFAULT, то будет установлен цвет по умолчанию.

Результаты работы программы показаны на рис. 9.27.

9.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 9.31. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define ID_INDICATOR_X                  101
#define ID_INDICATOR_Y                  102
#define IDR_MAINFRAME                   128
#define IDR_pr9TYPE                     129
#define IDR_TOOLBAR1                    130
#define ID_MENU_MENU1                   32771
#define ID_MENU_MENU2                   32772
#define ID_FORMAT_VIEWBUT1              32775
#define ID_FORMAT_ADD2BUT2              32776
#define ID_FORMAT_ADDTB2                32778
// ...
```

Листинг 9.32. Файл MainFrm.h (объявление класса фрейма)

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:  // control bar embedded members
   // CStatusBar  m_wndStatusBar;
   CToolBar    m_wndToolBar;
   CChildView    m_wndView;
```

```
public:
   CStatusBar  m_wndStatusBar;
// ...
public:
   afx_msg void OnMenuMenu1();
public:
   afx_msg void OnMenuMenu2();
public:
   afx_msg void OnFormatViewbut1();
public:
   CMenu *pmenu,              // Меню приложения
         *psubm;              // Выпадающее меню Format | ViewBut1
public:
   afx_msg void OnFormatAdd2but2();
   TBBUTTON tb2;              // Информация о кнопке 2
public:
   afx_msg void OnFormatAddtb2();
   CToolBar myToolBar;        // Новая панель инструментов
};
```

Листинг 9.33. Файл MainFrm.cpp (определение класса фрейма)

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   // ...
   ON_COMMAND(ID_MENU_MENU1, &CMainFrame::OnMenuMenu1)
   ON_COMMAND(ID_MENU_MENU2, &CMainFrame::OnMenuMenu2)
   ON_COMMAND(ID_FORMAT_VIEWBUT1, &CMainFrame::OnFormatViewbut1)
   ON_COMMAND(ID_FORMAT_ADD2BUT2, &CMainFrame::OnFormatAdd2but2)
   ON_COMMAND(ID_FORMAT_ADDTB2, &CMainFrame::OnFormatAddtb2)
END_MESSAGE_MAP()

static UINT indicators[] =
{
   ID_SEPARATOR,           // status line indicator
```

```
   ID_INDICATOR_X,
   ID_INDICATOR_Y,
   ID_INDICATOR_CAPS,
   ID_INDICATOR_NUM,
   ID_INDICATOR_SCRL,
};
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
   return -1;
   // create a view to occupy the client area of the frame
   if(!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, CRect(0,0,0,0),
                        this, AFX_IDW_PANE_FIRST, NULL))
   {
      TRACE0("Failed to create view window\n");
      return -1;
   }

   if(!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
                             CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
                             CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
      !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
   {
      TRACE0("Failed to create toolbar\n");
      return -1;      // fail to create
   }

   if(//!m_wndStatusBar.Create(this) ||
      !m_wndStatusBar.Create(this, WS_CHILD | WS_VISIBLE | CBRS_TOP) ||
      !m_wndStatusBar.SetIndicators(indicators,
                                    sizeof(indicators)/sizeof(UINT)))
   {
      TRACE0("Failed to create status bar\n");
      return -1;      // fail to create
   }
```

```
// TODO: Delete these three lines if you don't want the toolbar to be
// dockable
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);

// Получить ссылку на встроенный объект класса CToolBarCtrl
CToolBarCtrl &tbc = this->m_wndToolBar.GetToolBarCtrl();
// Получить данные о кнопке2
tbc.GetButton(tbc.CommandToIndex(ID_MENU_MENU2),&tb2);

// Создание новой "невидимой" панели инструментов у левой стороны
// фреймового окна
if(!myToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | CBRS_LEFT |
                       CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
                       CBRS_SIZE_DYNAMIC) ||
   !myToolBar.LoadToolBar(IDR_TOOLBAR1))
{
   TRACE0("Failed to create toolbar\n");
   return -1;
}
myToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&myToolBar);

// Задание стилей двух новых полей строки статуса
m_wndStatusBar.SetPaneInfo(
                       m_wndStatusBar.CommandToIndex(ID_INDICATOR_X),
                       ID_INDICATOR_X, SBPS_NORMAL, 50);
m_wndStatusBar.SetPaneInfo(
                       m_wndStatusBar.CommandToIndex(ID_INDICATOR_Y),
                       ID_INDICATOR_Y, SBPS_NORMAL, 50);
// Меняем цвет строки статуса
// Получить ссылку на объект CStatusBarCtrl
```

```
    CStatusBarCtrl &sbc = m_wndStatusBar.GetStatusBarCtrl();
    sbc.SetBkColor(RGB(0,255,0));              // Установить зеленый цвет

    return 0;
}
// ...
void CMainFrame::OnMenuMenu1()
{
    // TODO: Add your command handler code here
    AfxMessageBox("Menu1");
}

void CMainFrame::OnMenuMenu2()
{
    // TODO: Add your command handler code here
    AfxMessageBox("Menu2");
}

void CMainFrame::OnFormatViewbut1()
{
    // TODO: Add your command handler code here
    static int ch=1;                  // Вначале меню помечено галочкой
    pmenu = this->GetMenu();          // Получить указатель меню приложения
    // Получить указатель на выпадающее меню "Format"
    psubm = pmenu->GetSubMenu(5);
    // Получить ссылку на встроенный объект класса CToolBarCtrl
    CToolBarCtrl &tbc = this->m_wndToolBar.GetToolBarCtrl();

    if(!ch)                           // Если меню не было помечено
    {
        // Пометить меню
        psubm->CheckMenuItem(ID_FORMAT_VIEWBUT1, MF_CHECKED);
        ch = 1;
        // Показать кнопку 1
        tbc.SetState(ID_MENU_MENU1,TBSTATE_ENABLED);
```

```
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
   else                                  // Если меню было помечено
   {
      // Снять пометку
      psubm->CheckMenuItem( ID_FORMAT_VIEWBUT1,MF_UNCHECKED);
      ch = 0;
      // Скрыть кнопку1
      tbc.SetState(ID_MENU_MENU1, TBSTATE_HIDDEN);
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
}

void CMainFrame::OnFormatAdd2but2()
{
   // TODO: Add your command handler code here
   static int ch=0;                  // Вначале меню не помечено галочкой
   pmenu = this->GetMenu();          // Получить указатель меню приложения
   // Получить указатель на выпадающее меню "Format"
   psubm = pmenu->GetSubMenu(5);
   // Получить ссылку на встроенный объект класса CToolBarCtrl
   CToolBarCtrl &tbc = this->m_wndToolBar.GetToolBarCtrl();

   if(!ch)                           // Если меню не было помечено
   {
      // Пометить меню
      psubm->CheckMenuItem(ID_FORMAT_ADD2BUT2, MF_CHECKED);
      ch = 1;
      // Установить для кнопки 2 "серый" стиль
      tb2.fsState  = TBSTATE_INDETERMINATE;
      tbc.InsertButton(7,&tb2);         // Вставить кнопку 2 в конец
      tbc.InsertButton(0,&tb2);         // Вставить кнопку 2 в начало
```

```
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
   else
   {
      // Снять пометку
      psubm->CheckMenuItem(ID_FORMAT_ADD2BUT2, MF_UNCHECKED);
      ch = 0;
      // Удалить кнопки 2 с конца и из начала
      tbc.DeleteButton(8);
      tbc.DeleteButton(0);
      // Послать сообщение об изменении размера фрейма
      this->SendMessageA(WM_SIZE);
   }
}

void CMainFrame::OnFormatAddtb2()
{
   // TODO: Add your command handler code here
   static int ch=0;                  // Вначале меню не помечено галочкой
   pmenu = this->GetMenu();          // Получить указатель меню приложения
   // Получить указатель на выпадающее меню "Format"
   psubm = pmenu->GetSubMenu(5);
   if(!ch)                           // Если меню не было помечено
   {
      // Пометить меню
      psubm->CheckMenuItem( ID_FORMAT_ADDTB2, MF_CHECKED);
      ch = 1;
      // Показать новую панель
      this->ShowControlBar(&myToolBar, 1, 0 );
   }
   else
   {
```

```
      // Снять пометку
      psubm->CheckMenuItem(ID_FORMAT_ADDTB2, MF_UNCHECKED);
      ch = 0;
      // Скрыть новую панель
      this->ShowControlBar(&myToolBar, 0, 0);
   }
}
```

Листинг 9.34. Файл ChildView.h (объявление класса представления)

```
// ...
class CChildView : public CWnd
{
// ...
public:
   afx_msg void OnMouseMove(UINT nFlags, CPoint point);
};
```

Листинг 9.35. Файл ChildView.cpp (определение класса представления)

```
// ...
#include "ChildView.h"
#include "MainFrm.h"
// ...
BEGIN_MESSAGE_MAP(CChildView, CWnd)
   ON_WM_PAINT()
   ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()
// ...
void CChildView::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   // Получить указатель на фреймовое окно
   CMainFrame* pFrm = (CMainFrame* )theApp.m_pMainWnd;
```

```
    CString s;
    s.Format("x= %d",point.x);          // Записать в строку координату X
    pFrm->m_wndStatusBar.SetPaneText(   // Вывести ее в строку статуса
                pFrm->m_wndStatusBar.CommandToIndex(ID_INDICATOR_X), s);
    s.Format("y= %d",point.y);          // Записать в строку координату Y
    pFrm->m_wndStatusBar.SetPaneText(   // Вывести ее в строку статуса
                pFrm->m_wndStatusBar.CommandToIndex(ID_INDICATOR_Y), s);

    CWnd::OnMouseMove(nFlags, point);
}
```

Глава 10

Архитектура документ/представление

Создадим новый проект, на примере которого рассмотрим принципы работы с документами.

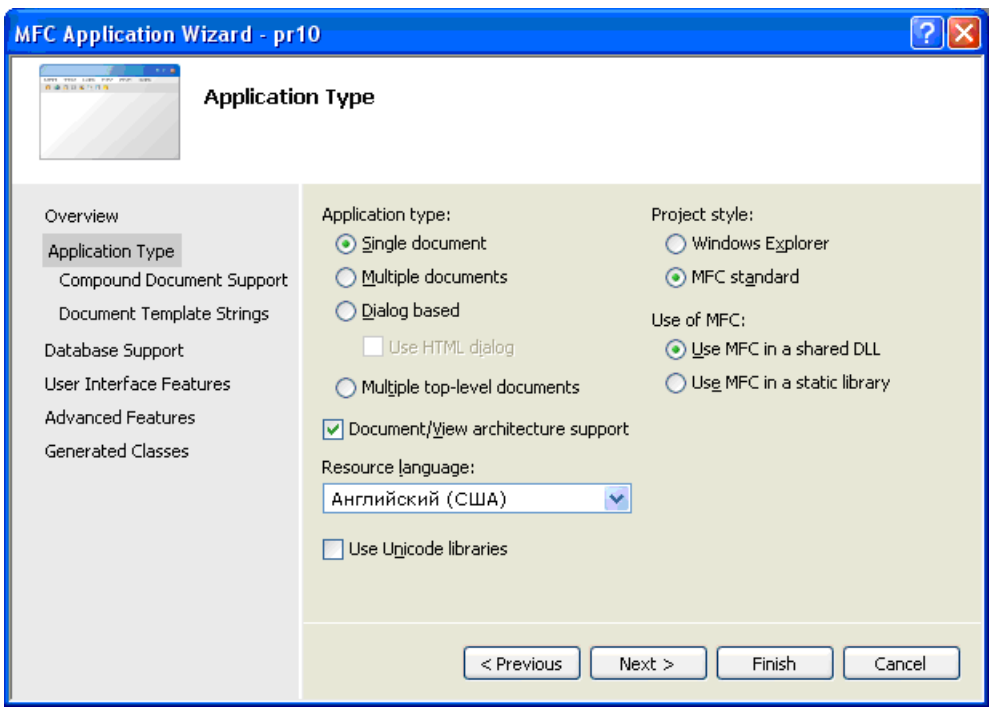


Рис. 10.1. Использование архитектуры документ/представление

Создайте проект **pr10** аналогично проекту **pr9**, *не снимая* флажка **Document/View architecture support** (Поддержка архитектуры документ/представление) и *убрав* флажок **Printing and print preview** (Печать и предварительный просмотр). Нужно изменить следующие опции относительно изначально предложенных мастером (рис. 10.1 и 10.2):

- ❒ на вкладке **Application Type**:
  - SDI-документ;
  - без Unicode;
- ❒ на вкладке **Advanced Features**:
  - без печати и предварительного просмотра (снять флажок **Printing and print preview**).

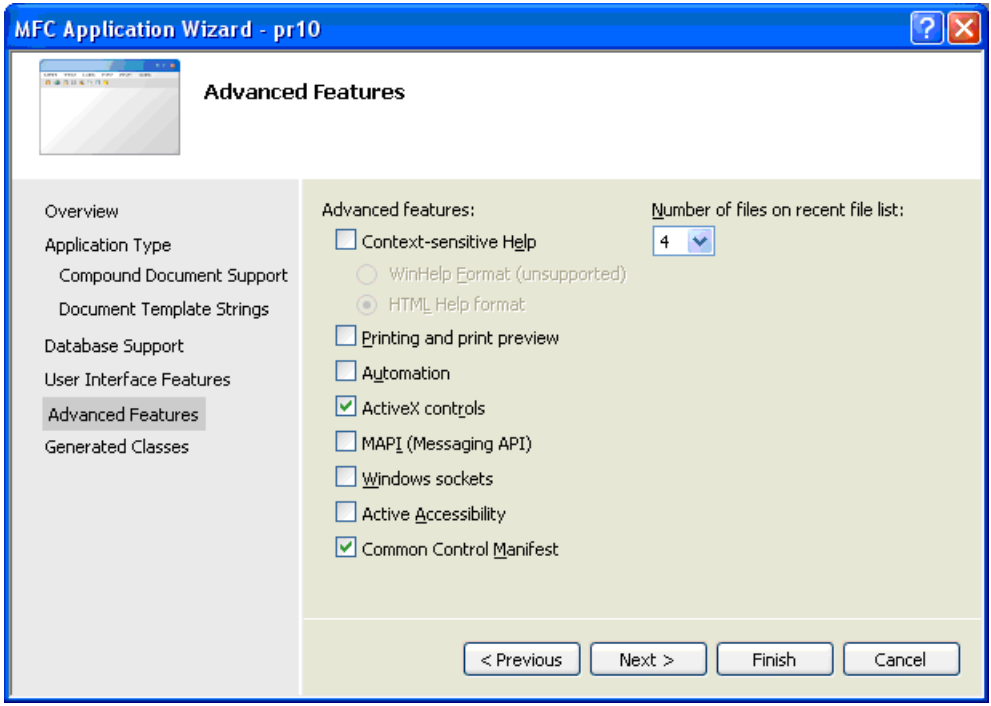


Рис. 10.2. Отключение возможности печати и предварительного просмотра

10.1. Описание программы

В основе архитектуры документ/представление лежат три основных понятия — фрейм, представление и документ.

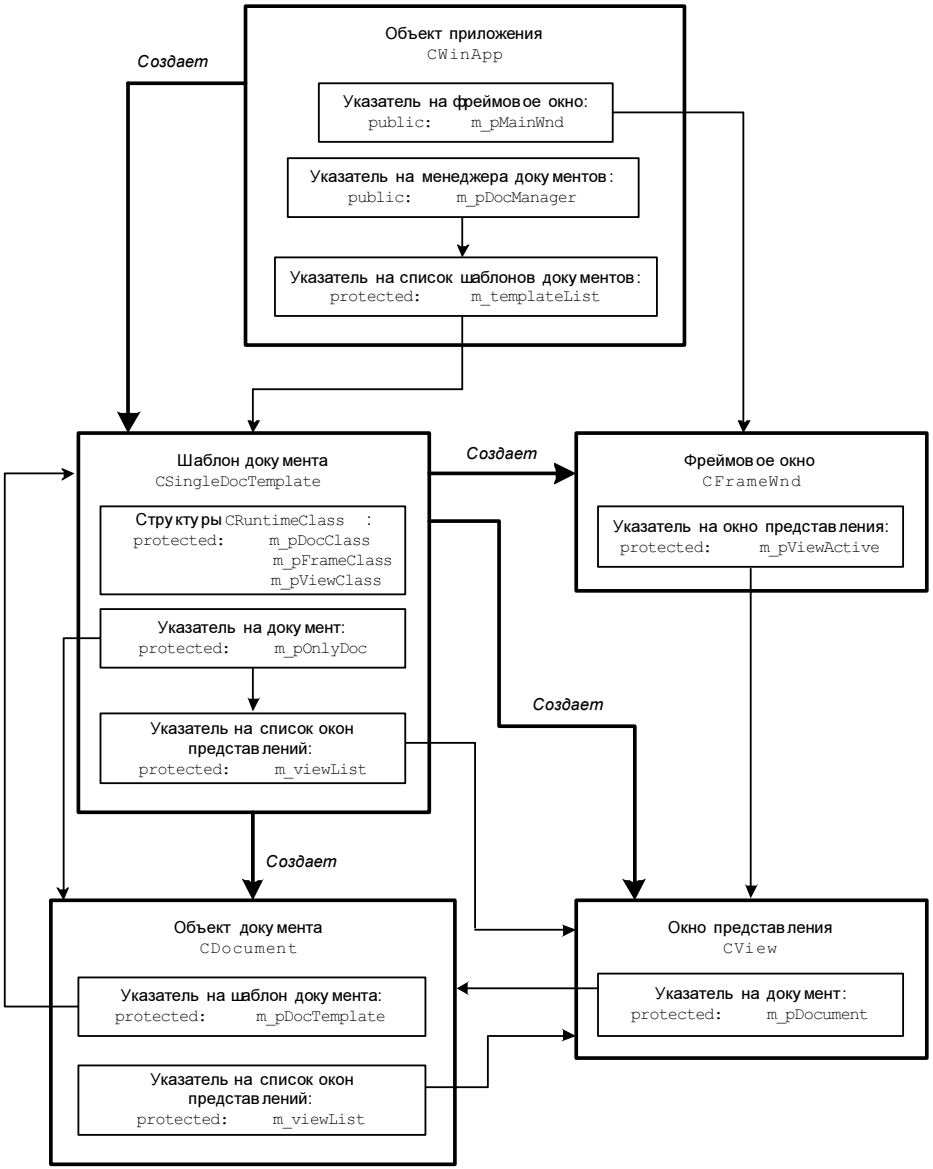


Рис. 10.3. Архитектура документ/представление

Документ отвечает за работу с данными (текст, графическое изображение и т. д.). *Окно представления* взаимодействует с документом и отображает данные. Вид архитектуры показан на рис. 10.3.

10.1.1. Класс приложения

У класса приложения в функции `Cpr10App::InitInstance()` теперь нет создания окна фрейма (`pFrame->LoadFrame(IDR_MAINFRAME, WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, NULL, NULL)`). Все окна (фреймовое и представления) создаются с использованием шаблона документа. Изменения в файле приведены в листинге 10.1.

Листинг 10.1. Изменения в файле pr10.cpp для работы с различными документами

```
#include "stdafx.h"
#include "pr10.h"
#include "MainFrm.h"

#include "pr10Doc.h"
#include "pr10View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Cpr10App

BEGIN_MESSAGE_MAP(Cpr10App, CWinApp)
   ON_COMMAND(ID_APP_ABOUT, &Cpr10App::OnAppAbout)
   // Standard file based document commands
   ON_COMMAND(ID_FILE_NEW, &CWinApp::OnFileNew)
   ON_COMMAND(ID_FILE_OPEN, &CWinApp::OnFileOpen)
END_MESSAGE_MAP()

// Cpr10App initialization
BOOL Cpr1App::InitInstance()
```

```
{
    // ...
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(4);  // Load standard INI file options
                                // (including MRU)
    // Register the application's document templates.  Document templates
    // serve as the connection between documents, frame windows and views
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
                        IDR_MAINFRAME,
                        RUNTIME_CLASS(Cpr10Doc),
                        RUNTIME_CLASS(CMainFrame), // main SDI frame window
                        RUNTIME_CLASS(Cpr10View));
    if (!pDocTemplate)
        return FALSE;
    AddDocTemplate(pDocTemplate);

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // Dispatch commands specified on the command line.  Will return FALSE
    // if app was launched with /RegServer, /Register, /Unregserver
    // or /Unregister.
    if(!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    // call DragAcceptFiles only if there's a suffix
    //  In an SDI app, this should occur after ProcessShellCommand
```

```
   // Enable drag/drop open
   m_pMainWnd->DragAcceptFiles();
   return TRUE;
}
```

Функция загрузки списка последних используемых файлов (MRU — most recently used, недавно использованные) выглядит следующим образом:

```
void CWinApp::LoadStdProfileSettings(
   UINT nMaxMRU = _AFX_MRU_COUNT);          // Количество файлов в списке
```

В этом проекте `nMaxMRU = 4`, т. к. мы не меняли стандартные настройки проекта (поле **Number of files on recent file list** (Количество файлов в списке недавно используемых) задано равным 4, см. рис. 10.2). Если `nMaxMRU = 0`, то список файлов не поддерживается.

Класс, определяющий шаблон документа для однодокументного (SDI) приложения, определен как:

```
class CSingleDocTemplate : public CDocTemplate
class CDocTemplate : public CCmdTarget
class CCmdTarget : public CObject
```

Конструктор выглядит следующим образом:

```
CSingleDocTemplate::CSingleDocTemplate(
   UINT nIDResource,            // Идентификатор ресурсов (меню,
                                // акселераторы)
   CRuntimeClass* pDocClass,    // Указатель на объект CRunTimeClass,
                                // отвечающий за данные документа
   CRuntimeClass* pFrameClass,  // Указатель на объект CRunTimeClass,
                                // отвечающий за фрейм документа
   CRuntimeClass* pViewClass);  // Указатель на объект CRunTimeClass,
                                // отвечающий за представление документа
```

Получить указатель на объект `CRunTimeClass` можно с помощью макроса:

```
RUNTIME_CLASS(
   class_name)                  // Имя класса
```

В качестве аргумента в `RUNTIME_CLASS` используется не объект класса или указатель на него, а имя класса. Это позволяет создавать объекты данных классов в качестве внутренних переменных класса шаблона документа. Именно такое объединение классов внутри класса шаблона документа и по-

зволяет установить между ними соответствие. Чтобы макрос нормально работал, надо, чтобы в объявлении класса (class_name.h) один из конструкторов объявлялся с использованием макроса `DECLARE_DYNCREATE`, а в файле определения класса (class_name.cpp) должен быть макрос `IMPLEMENT_DYNCREATE`.

Макрос, обеспечивающий динамическое создание объектов, произведенных от класса `CObject`, в ходе выполнения программы:

```
DECLARE_DYNCREATE(
   class_name)                    // Имя класса
```

Макрос, который является парой для `DECLARE_DYNCREATE` и может использоваться исключительно с ним:

```
IMPLEMENT_DYNCREATE(
   class_name,                    // Имя класса
   base_class_name)               // Имя базового класса
```

Добавление шаблона документа в список доступных шаблонов выполняется с помощью функции:

```
void CWinApp::AddDocTemplate(
   CDocTemplate* pTemplate);      // Добавляемый шаблон
```

Функция, разрешающая открывать файлы двойным щелчком (используется вместе с функцией `RegisterShellFileTypes()`), выглядит так:

```
void CWinApp::EnableShellOpen();
```

Регистрация всех типов документов приложения выполняется функцией:

```
void CWinApp::RegisterShellFileTypes(
   BOOL bCompat = FALSE);         // Возможность использования команд
                                  // Print и Print to
```

Класс, отвечающий за обработку командной строки при запуске приложения, определен следующим образом:

```
class CCommandLineInfo : public CObject
```

Для проверки командной строки и заполнения `CCommandLineInfo` используется функция:

```
void CWinApp::ParseCommandLine(
   CCommandLineInfo& rCmdInfo);   // Параметры командной строки
```

Функция обработки командной строки:

```
BOOL CWinApp::ProcessShellCommand(
   CCommandLineInfo& rCmdInfo);   // Параметры командной строки
```

Эта функция поддерживает следующие параметры в командной строке:

- **app.exe** — запускает приложение и открывает новый файл;
- **app.exe имя_файла** — запускает приложение и открывает указанный файл;
- **app.exe /p имя_файла** — запускает приложение и распечатывает указанный файл на принтере, используемом по умолчанию;
- **app.exe /pt имя_файла принтер драйвер порт** — запускает приложение и распечатывает указанный файл на указанном принтере;
- **app.exe /dde** — запускает приложение и ждет команды DDE (динамического обмена данными);
- **app.exe /Automation** — запускает приложение в качестве автоматного сервера OLE;
- **app.exe /Embedding** — запускает приложение в режиме редактирования внедренного объекта OLE.

Функция, разрешающая поддержку drag/drop (перенести/оставить), обеспечивает возможность "перетаскивания" файлов:

```
void CWnd::DragAcceptFiles(
   BOOL bAccept = TRUE);          // TRUE - разрешить, FALSE - запретить
```

10.1.2. Класс фрейма

У фреймового окна теперь нет объекта `CChildView m_wndView`, и в функции `CMainFrame::OnCreate()` нет явного создания окна представления (`m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, CRect(0,0,0,0), this, AFX_IDW_PANE_FIRST, NULL)`). Добавлены макросы для динамического создания объектов. Нет обработки сообщения `ON_WM_SETFOCUS()`. Изменения в файлах приведены в листингах 10.2 и 10.3.

Листинг 10.2. Изменения в файле MainFrm.h для работы с документами

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
```

```
protected: // create from serialization only
   CMainFrame();
   DECLARE_DYNCREATE(CMainFrame)
};
```

Листинг 10.3. Изменения в файле MainFrm.cpp для работы с документами

```
// ...
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_CREATE()
END_MESSAGE_MAP()
// ...
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if(CFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;

   if(!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
                             CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS |
                             CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
      !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
   {
      TRACE0("Failed to create toolbar\n");
      return -1;      // fail to create
   }

   if(!m_wndStatusBar.Create(this) ||
      !m_wndStatusBar.SetIndicators(indicators,
                                    sizeof(indicators)/sizeof(UINT)))
   {
      TRACE0("Failed to create status bar\n");
      return -1;      // fail to create
   }
```

```
    // TODO: Delete these three lines if you don't want the toolbar to be
    // dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}
```

10.1.3. Класс документа

Появился новый класс `Cpr10Doc`. Изменения в программе приведены в листингах 10.4 и 10.5.

Листинг 10.4. Файл pr10Doc.h (объявление класса для работы с документами)

```
#pragma once
class Cpr10Doc : public CDocument
{
protected: // create from serialization only
    Cpr10Doc();
    DECLARE_DYNCREATE(Cpr10Doc)

// Attributes
public:

// Operations
public:

// Overrides
public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Implementation
public:
```

```
    virtual ~Cpr10Doc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    DECLARE_MESSAGE_MAP()
};
```

Листинг 10.5. Файл pr10Doc.cpp (определение класса для работы с документами)

```
// pr10Doc.cpp : implementation of the Cpr10Doc class
//
#include "stdafx.h"
#include "pr10.h"
#include "pr10Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Cpr10Doc

IMPLEMENT_DYNCREATE(Cpr10Doc, CDocument)

BEGIN_MESSAGE_MAP(Cpr10Doc, CDocument)
END_MESSAGE_MAP()

// Cpr10Doc construction/destruction
Cpr10Doc::Cpr10Doc()
```

```
{
    // TODO: add one-time construction code here
}

Cpr10Doc::~Cpr10Doc()
{
}

BOOL Cpr10Doc::OnNewDocument()
{
    if(!CDocument::OnNewDocument())
        return FALSE;
    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)
    return TRUE;
}

// Cpr10Doc serialization
void Cpr10Doc::Serialize(CArchive& ar)
{
    if(ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

// Cpr10Doc diagnostics
#ifdef _DEBUG
void Cpr10Doc::AssertValid() const
{
    CDocument::AssertValid();
```

```
}
void Cpr10Doc::Dump(CDumpContext& dc) const
{
   CDocument::Dump(dc);
}
#endif //_DEBUG

// Cpr10Doc commands
```

Класс для работы с документом определен как:

```
class CDocument : public CCmdTarget
class CCmdTarget : public CObject
```

Функция создания нового документа выглядит следующим образом:

```
virtual BOOL CDocument::OnNewDocument();      // 0 - ошибка
```

Обмен данными между документом и представлением происходит через архив (`class CArchive`). Функция, осуществляющая чтение объекта из архива и запись его в архив, используется такая:

```
virtual void CObject::Serialize(
   CArchive& ar);                             // Ссылка на архив
```

Функция определения, был ли архив открыт для сохранения данных в архиве или для загрузки данных из архива:

```
BOOL CArchive:: IsStoring() const;   // 1 - для сохранения, 0 - загрузки
```

10.1.4. Класс представления

У класса окна представления добавилась новая функция `Cpr10View::GetDocument()` и для возможности рисования вместо функции `CChildView::OnPaint()` появилась `Cpr10View::OnDraw()`. Добавлены макросы для динамического создания объектов. Изменения в файлах приведены в листингах 10.6 и 10.7.

Листинг 10.6. Изменения в файле pr10View.h для работы с документами

```
// ...
class Cpr10View : public CView
{
```

```
protected: // create from serialization only
   Cpr10View();
   DECLARE_DYNCREATE(Cpr10View)

// Attributes
public:
   Cpr10Doc* GetDocument() const;
// ...
// Overrides
public:
   virtual void OnDraw(CDC* pDC);  // overridden to draw this view
// ...
};
#ifndef _DEBUG  // debug version in pr10View.cpp
inline Cpr10Doc* Cpr10View::GetDocument() const
{ return reinterpret_cast<Cpr10Doc*>(m_pDocument); }
#endif
```

Листинг 10.7. Изменения в файле pr10View.cpp для работы с документами

```
// ...
#include "pr10Doc.h"
// ...
IMPLEMENT_DYNCREATE(Cpr10View, CView)

BEGIN_MESSAGE_MAP(Cpr10View, CView)
END_MESSAGE_MAP()

void Cpr10View::OnDraw(CDC* /*pDC*/)
{
   Cpr10Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if(!pDoc)
      return;

   // TODO: add draw code for native data here
}
```

```
Cpr10Doc* Cpr10View::GetDocument() const // non-debug version is inline
{
   ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(Cpr10Doc)));
   return (Cpr10Doc*)m_pDocument;
}
```

Чисто виртуальная функция для отображения документа определена так:

```
virtual void CView::OnDraw(
   CDC* pDC) = 0;           // Указатель на контекст устройства, который
                            // надо использовать для представления образа
                            // документа
```

Следующая функция проверяет существование объекта `CRunTimeClass`:

```
BOOL CObject::IsKindOf(                  // 0 – объект не существует
   const CRuntimeClass* pClass) const;   // Указатель на объект
                                         // CRunTimeClass
```

10.1.5. Доступ к классам приложения

Здесь будут рассмотрены возможные способы доступа и общения между классами. Добавим в каждый класс (фрейма, документа и представления) функцию `f_mes()`, которая будет выводить сообщение своего класса. Изменения в файлах приведены в листингах 10.8—10.13.

Листинг 10.8. Изменения в файле MainFrm.h для выдачи сообщения класса фреймового окна

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   // Сообщение фрейма
   void f_mes(void);
};
```

Листинг 10.9. Изменения в файле MainFrm.cpp для выдачи сообщения класса фреймового окна

```
// ...
// Сообщение фрейма
void CMainFrame::f_mes(void)
{
   ::AfxMessageBox("Сообщение фрейма");
}
```

Листинг 10.10. Изменения в файле pr10Doc.h для выдачи сообщения класса документа

```
// ...
class Cpr10Doc : public CDocument
{
// ...
public:
   // Сообщение документа
   void f_mes(void);
};
```

Листинг 10.11. Изменения в файле pr10Doc.cpp для выдачи сообщения класса документа

```
// ...
// Сообщение документа
void Cpr10Doc::f_mes(void)
{
   ::AfxMessageBox("Сообщение документа");
}
```

Листинг 10.12. Изменения в файле pr10View.h для выдачи сообщения класса окна представления

```
// ...
class Cpr10View : public CView
```

```
{
// ...
public:
   // Сообщение представления
   void f_mes(void);
};
```

Листинг 10.13. Изменения в файле pr10View.cpp для выдачи сообщения класса окна представления

```
// ...
// Сообщение представления
void Cpr10View::f_mes(void)
{
   ::AfxMessageBox("Сообщение представления");
}
```

Примечание

Чтобы было удобнее набирать на клавиатуре функцию `AfxMessageBox()`, надо напечатать `"::"`, и тогда появится всплывающая подсказка, где можно выбрать название функции (рис. 10.4).

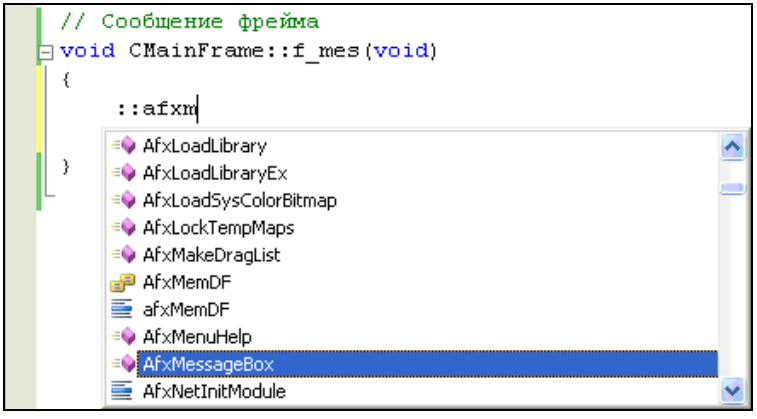


Рис. 10.4. Вызов подсказки для функций, которые не относятся к классу

Теперь добавим новый пункт главного меню **M** и его подменю **Frame**, **Doc**, **View** (рис. 10.5) и их обработку:

- **Frame** — в класс `CMainFrame` (рис. 10.6);
- **Doc** — в класс `Cpr10Doc`;
- **View** — в класс `Cpr10View`.

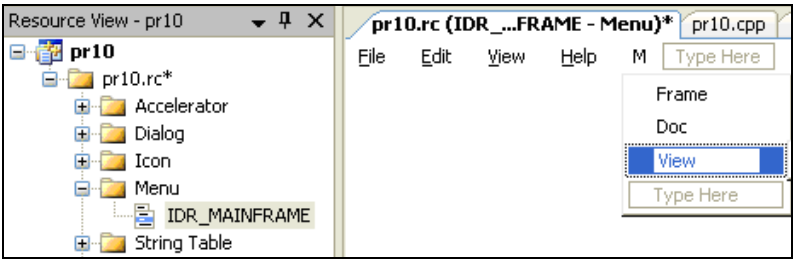
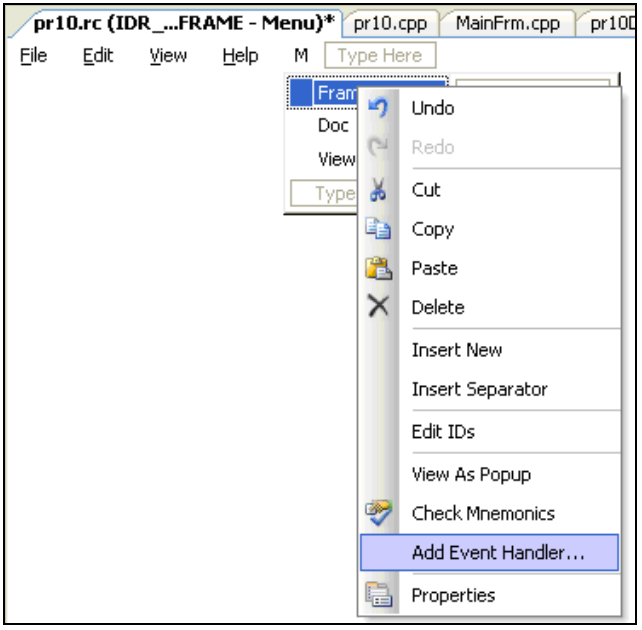
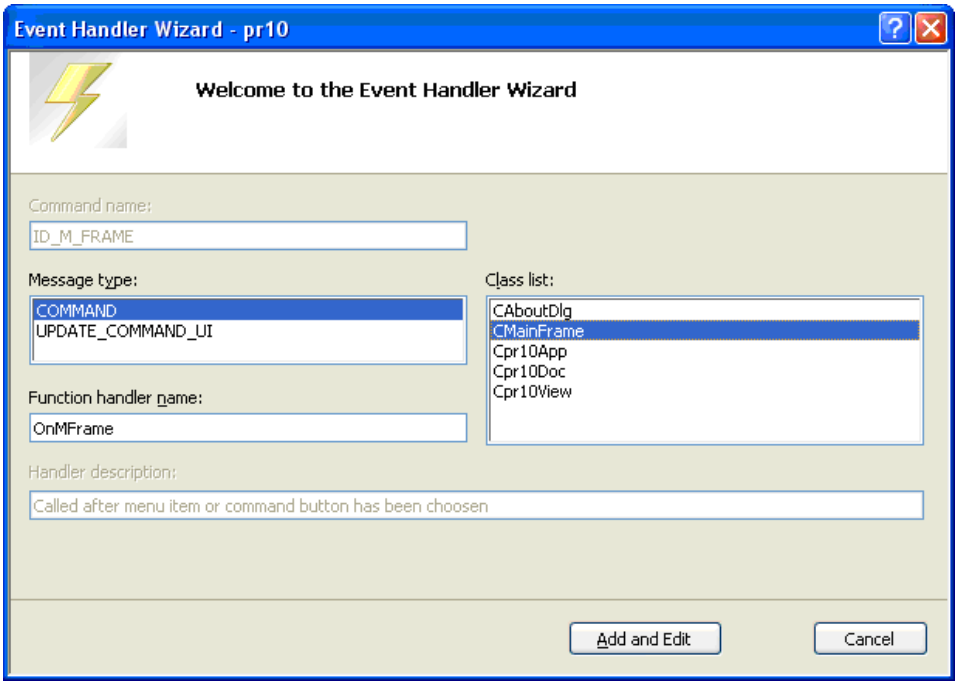


Рис. 10.5. Добавление меню
для реализации доступа к классам



а

Рис. 10.6. Добавление обработки сообщения меню (*а*)



б

Рис. 10.6. Функция обработки сообщения меню **M | Frame** (*б*)

В функции обработки просто вызовем свою функцию `f_mes()` для каждого класса. Изменения в файлах приведены в листингах 10.14—10.20.

Листинг 10.14. Изменения в файле Resource.h для обработки меню вызова сообщения каждого класса

```
// ...
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDR_MAINFRAME                   128
#define IDR_pr10TYPE                    129
#define ID_M_FRAME                      32771
#define ID_M_DOC                        32772
#define ID_M_VIEW                       32773
// ...
```

Листинг 10.15. Изменения в файле MainFrm.h для обработки меню вызова сообщения класса фреймового окна

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   afx_msg void OnMFrame();
};
```

Листинг 10.16. Изменения в файле MainFrm.cpp для обработки меню вызова сообщения класса фреймового окна

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_CREATE()
   ON_COMMAND(ID_M_FRAME, &CMainFrame::OnMFrame)
END_MESSAGE_MAP()

// ...
void CMainFrame::OnMFrame()
{
   // TODO: Add your command handler code here
   this->f_mes();
}
```

Листинг 10.17. Изменения в файле pr10Doc.h для обработки меню вызова сообщения класса документа

```
// ...
class Cpr10Doc : public CDocument
{
   // ...
public:
   afx_msg void OnMDoc();
};
```

Листинг 10.18. Изменения в файле pr10Doc.cpp для обработки меню вызова сообщения класса документа

```
// ...
BEGIN_MESSAGE_MAP(Cpr10Doc, CDocument)
   ON_COMMAND(ID_M_DOC, &Cpr10Doc::OnMDoc)
END_MESSAGE_MAP()
// ...
void Cpr10Doc::OnMDoc()
{
   // TODO: Add your command handler code here
   this->f_mes();
}
```

Листинг 10.19. Изменения в файле pr10View.h для обработки меню вызова сообщения класса окна представления

```
// ...
class Cpr10View : public CView
{
// ...
public:
   afx_msg void OnMView();
};
```

Листинг 10.20. Изменения в файле pr10View.cpp для обработки меню вызова сообщения класса окна представления

```
// ...
BEGIN_MESSAGE_MAP(Cpr10View, CView)
   ON_COMMAND(ID_M_VIEW, &Cpr10View::OnMView)
END_MESSAGE_MAP()
// ...
void Cpr10View::OnMView()
{
   // TODO: Add your command handler code here
   this->f_mes();
}
```

Результаты работы программы показаны на рис. 10.7.

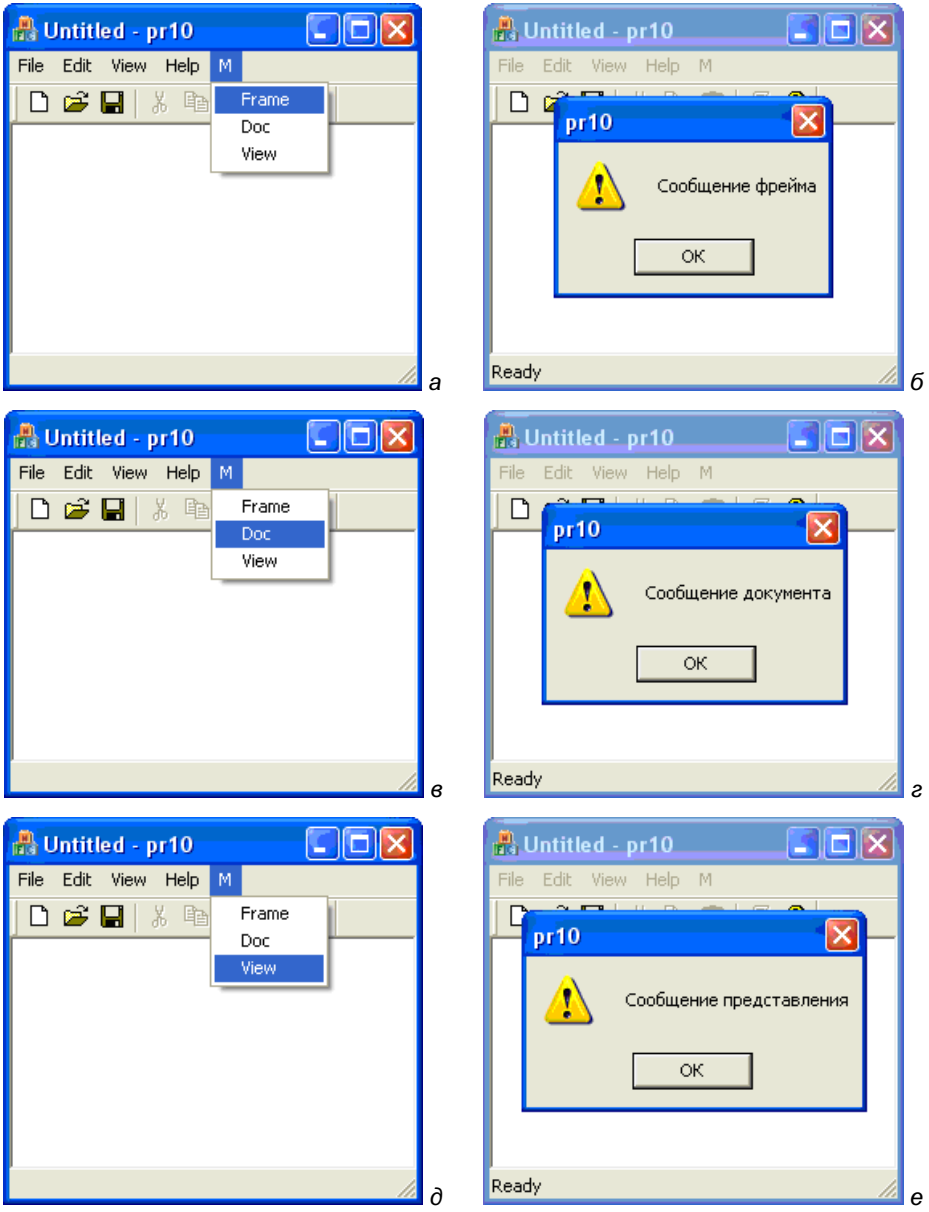


Рис. 10.7. Работа с меню **Frame** (*а*, *б*), с меню **Doc** (*в*, *г*) и с меню **View** (*д*, *е*)

Теперь сделаем так, чтобы при выборе пункта меню выдавалось не только свое сообщение, но и сообщения других классов, и сделаем это разными способами. Изменения в файлах приведены в листингах 10.21—10.23.

Листинг 10.21. Изменения в файле MainFrm.cpp для вызова сообщений класса документа и класса окна представления

```
// ...
#include "pr10Doc.h"
#include "pr10View.h"
// ...
void CMainFrame::OnMFrame()
{
   // TODO: Add your command handler code here
   this->f_mes();

   // Получить позицию первого шаблона документа
   POSITION post = theApp.GetFirstDocTemplatePosition();
   // Получить указатель на первый шаблон (по его позиции)
   CSingleDocTemplate *psdt =
                (CSingleDocTemplate *)theApp.GetNextDocTemplate(post);
   // Получить позицию первого документа
   POSITION posd = psdt->GetFirstDocPosition();
   // Получить указатель на первый документ (по его позиции)
   Cpr10Doc *pdoc1 = (Cpr10Doc *) psdt->GetNextDoc(posd);
   pdoc1->f_mes();                    // Вывести сообщение документа

   // Получить позицию первого шаблона документа
   POSITION post1 =
                theApp.m_pDocManager->GetFirstDocTemplatePosition();
   // Получить указатель на первый шаблон (по его позиции)
   CSingleDocTemplate*psdt1 = (CSingleDocTemplate *)
                 theApp.m_pDocManager->GetNextDocTemplate(post1);
   // Получить позицию первого документа
   POSITION posd1 = psdt1->GetFirstDocPosition();
```

```
   // Получить указатель на первый документ (по его позиции)
   Cpr10Doc *pdoc2 = (Cpr10Doc *) psdt1->GetNextDoc(posd1);
   pdoc2->f_mes();                        // Вывести сообщение документа

   // Получить указатель на активное окно представления
   Cpr10View *pView1 = (Cpr10View *)this->GetActiveView();
   pView1->f_mes();                       // Вывести сообщение представления

   // Через представление получить указатель на документ
   Cpr10Doc *pdoc3 = pView1->GetDocument();
   pdoc3->f_mes();                        // Вывести сообщение документа

   // Получить указатель на документ
   Cpr10Doc *pdoc4 = (Cpr10Doc *)theApp.GetRuntimeClass();
   pdoc4->f_mes();                       // Вывести сообщение документа

   // Получить указатель представления
   Cpr10View *pView2 = (Cpr10View *)theApp.GetRuntimeClass();
   pView2->f_mes();                     // Вывести сообщение представления
}
```

Листинг 10.22. Изменения в файле pr10Doc.cpp для вызова сообщений класса фреймового окна и класса окна представления

```
// ...
#include "MainFrm.h"
#include "pr10View.h"
// ...
void Cpr10Doc::OnMDoc()
{
   // TODO: Add your command handler code here
   this->f_mes();

   // Получить указатель на фрейм
   CMainFrame *pFrame = (CMainFrame *)theApp.m_pMainWnd;
   pFrame->f_mes();                    // Вывести сообщение фрейма
```

```
   // Получить указатель на фрейм
   CMainFrame *pFrame1 = (CMainFrame *)theApp.GetRuntimeClass();
   pFrame1->f_mes();                  // Вывести сообщение фрейма

   // Получить указатель на фрейм
   CMainFrame *pFrame2 =
                  (CMainFrame *)theApp.m_pDocManager->GetRuntimeClass();
   pFrame2->f_mes();                  // Вывести сообщение фрейма

   // Получить позицию первого представления в списке представлений
   POSITION pos = this->GetFirstViewPosition();
   // Получить указатель на первое представление (по его позиции)
   Cpr10View *pView = (Cpr10View *)this->GetNextView(pos);
   pView->f_mes();                    // Вывести сообщение представления

   // Получить указатель на представление
   Cpr10View *pView1 = (Cpr10View *)theApp.GetRuntimeClass();
   pView1->f_mes();                   // Вывести сообщение представления
}
```

Листинг 10.23. Изменения в файле pr10View.cpp для вызова сообщений класса документа и класса фреймового окна

```
// ...
#include "MainFrm.h"
// ...
void Cpr10View::OnMView()
{
   // TODO: Add your command handler code here
   this->f_mes();

   // Получить указатель на документ
   Cpr10Doc *pDoc = (Cpr10Doc *)this->m_pDocument;
   pDoc->f_mes();                          // Вывести сообщение документа
```

```
   // Получить указатель на документ
   Cpr10Doc *pDoc1 = this->GetDocument();
   pDoc1->f_mes();                          // Вывести сообщение документа

   // Получить указатель на фрейм
   CMainFrame *pFrame = (CMainFrame *)theApp.m_pMainWnd;
   pFrame->f_mes();                         // Вывести сообщение фрейма
}
```

Здесь `POSITION` — тип, используемый для получения позиции в списке.

Получить позицию первого шаблона документа элемента в списке шаблонов документов можно с помощью функции (функция возвращает позицию элемента или 0, если список пустой):

```
POSITION CWinApp::GetFirstDocTemplatePosition() const
```

Получить указатель на шаблон документа по его позиции можно функцией:

```
CDocTemplate* CWinApp::GetNextDocTemplate(   // Указатель на шаблон
                                             // документа
   POSITION& pos) const;                     // Позиция шаблона документа
                                             // в списке
```

Получить позицию первого документа в списке шаблонов документов можно следующей функцией (функция возвращает позицию элемента или 0, если список пустой):

```
virtual POSITION CDocTemplate::GetFirstDocPosition() const = 0;
```

Указатель на первый документ (по его позиции) можно получить с помощью функции:

```
virtual CDocument* CDocTemplate::GetNextDoc( // Указатель на документ
   POSITION& rPos) const = 0;                // Позиция документа в списке
```

Указатель на текущее окно представления:

```
CView* CFrameWnd::GetActiveView() const;     // NULL — если нет текущего
                                             // окна представления
```

Указатель на динамически созданный объект (функция возвращает `NULL`, если объект не существует):

```
virtual CRuntimeClass* CObject::GetRuntimeClass() const;
```

10.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 10.24. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDD_ABOUTBOX                100
#define IDP_OLE_INIT_FAILED         100
#define IDR_MAINFRAME               128
#define IDR_pr10TYPE                129
#define ID_M_FRAME                  32771
#define ID_M_DOC                    32772
#define ID_M_VIEW                   32773
// ...
```

Листинг 10.25. Файл MainFrm.h (объявление класса фрейма)

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
public:
   // Сообщение фрейма
   void f_mes(void);
public:
   afx_msg void OnMFrame();
};
```

Листинг 10.26. Файл MainFrm.cpp (определение класса фрейма)

```
// ...
#include "MainFrm.h"
#include "pr10Doc.h"
#include "pr10View.h"
```

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
   ON_WM_CREATE()
   ON_COMMAND(ID_M_FRAME, &CMainFrame::OnMFrame)
END_MESSAGE_MAP()
// ...
// Сообщение фрейма
void CMainFrame::f_mes(void)
{
   ::AfxMessageBox("Сообщение фрейма");
}
void CMainFrame::OnMFrame()
{
   // TODO: Add your command handler code here
   this->f_mes();

   // Получить позицию первого шаблона документа
   POSITION post = theApp.GetFirstDocTemplatePosition();
   // Получить указатель на первый шаблон (по его позиции)
   CSingleDocTemplate *psdt =
                (CSingleDocTemplate *)theApp.GetNextDocTemplate(post);
   // Получить позицию первого документа
   POSITION posd = psdt->GetFirstDocPosition();
   // Получить указатель на первый документ (по его позиции)
   Cpr10Doc *pdoc1 = (Cpr10Doc *) psdt->GetNextDoc(posd);
   pdoc1->f_mes();                        // Вывести сообщение документа

   // Получить позицию первого шаблона документа
   POSITION post1 =
                 theApp.m_pDocManager->GetFirstDocTemplatePosition();
   // Получить указатель на первый шаблон (по его позиции)
   CSingleDocTemplate*psdt1 = (CSingleDocTemplate *)
                  theApp.m_pDocManager->GetNextDocTemplate(post1);
   // Получить позицию первого документа
   POSITION posd1 = psdt1->GetFirstDocPosition();
```

```
   // Получить указатель на первый документ (по его позиции)
   Cpr10Doc *pdoc2 = (Cpr10Doc *) psdt1->GetNextDoc(posd1);
   pdoc2->f_mes();                        // Вывести сообщение документа

   // Получить указатель на активное окно представления
   Cpr10View *pView1 = (Cpr10View *)this->GetActiveView();
   pView1->f_mes();                       // Вывести сообщение представления

   // Через представление получить указатель на документ
   Cpr10Doc *pdoc3 = pView1->GetDocument();
   pdoc3->f_mes();                        // Вывести сообщение документа

   // Получить указатель на документ
   Cpr10Doc *pdoc4 = (Cpr10Doc *)theApp.GetRuntimeClass();
   pdoc4->f_mes();                       // Вывести сообщение документа

   // Получить указатель представления
   Cpr10View *pView2 = (Cpr10View *)theApp.GetRuntimeClass();
   pView2->f_mes();                     // Вывести сообщение представления
}
```

Листинг 10.27. Файл pr10Doc.h (объявление класса документа)

```
// ...
class Cpr10Doc : public CDocument
{
// ...
public:
   // Сообщение документа
   void f_mes(void);
public:
   afx_msg void OnMDoc();
};
```

Листинг 10.28. Файл pr10Doc.cpp (определение класса документа)

```
// ...
#include "pr10Doc.h"
#include "MainFrm.h"
#include "pr10View.h"
// ...
BEGIN_MESSAGE_MAP(Cpr10Doc, CDocument)
   ON_COMMAND(ID_M_DOC, &Cpr10Doc::OnMDoc)
END_MESSAGE_MAP()
// ...
// Сообщение документа
void Cpr10Doc::f_mes(void)
{
   ::AfxMessageBox("Сообщение документа");
}
void Cpr10Doc::OnMDoc()
{
   // TODO: Add your command handler code here
   this->f_mes();

   // Получить указатель на фрейм
   CMainFrame *pFrame = (CMainFrame *)theApp.m_pMainWnd;
   pFrame->f_mes();                  // Вывести сообщение фрейма

   // Получить указатель на фрейм
   CMainFrame *pFrame1 = (CMainFrame *)theApp.GetRuntimeClass();
   pFrame1->f_mes();                 // Вывести сообщение фрейма

   // Получить указатель на фрейм
   CMainFrame *pFrame2 =
                 (CMainFrame *)theApp.m_pDocManager->GetRuntimeClass();
   pFrame2->f_mes();                 // Вывести сообщение фрейма

   // Получить позицию первого представления в списке представлений
   POSITION pos = this->GetFirstViewPosition();
```

```
   // Получить указатель на первое представление (по его позиции)
   Cpr10View *pView = (Cpr10View *)this->GetNextView(pos);
   pView->f_mes();                  // Вывести сообщение представления

   // Получить указатель на представление
   Cpr10View *pView1 = (Cpr10View *)theApp.GetRuntimeClass();
   pView1->f_mes();                 // Вывести сообщение представления
}
```

Листинг 10.29. Файл pr10View.h (объявление класса представления)

```
// ...
class Cpr10View : public CView
{
// ...
public:
   // Сообщение представления
   void f_mes(void);
public:
   afx_msg void OnMView();
};
// ...
```

Листинг 10.30. Файл pr10View.cpp (определение класса представления)

```
// ...
#include "pr10Doc.h"
#include "pr10View.h"
#include "MainFrm.h"
// ...
BEGIN_MESSAGE_MAP(Cpr10View, CView)
   ON_COMMAND(ID_M_VIEW, &Cpr10View::OnMView)
END_MESSAGE_MAP()
// ...
// Сообщение представления
```

```
void Cpr10View::f_mes(void)
{
   ::AfxMessageBox("Сообщение представления");
}
void Cpr10View::OnMView()
{
   // TODO: Add your command handler code here
   this->f_mes();

   // Получить указатель на документ
   Cpr10Doc *pDoc = (Cpr10Doc *)this->m_pDocument;
   pDoc->f_mes();                          // Вывести сообщение документа

   // Получить указатель на документ
   Cpr10Doc *pDoc1 = this->GetDocument();
   pDoc1->f_mes();                         // Вывести сообщение документа

   // Получить указатель на фрейм
   CMainFrame *pFrame = (CMainFrame *)theApp.m_pMainWnd;
   pFrame->f_mes();                        // Вывести сообщение фрейма
}
```

Глава 11

Работа с графическими данными с помощью метафайла

Создадим новый проект, на примере которого рассмотрим возможности отображения графических данных в окне представления с помощью метафайла.

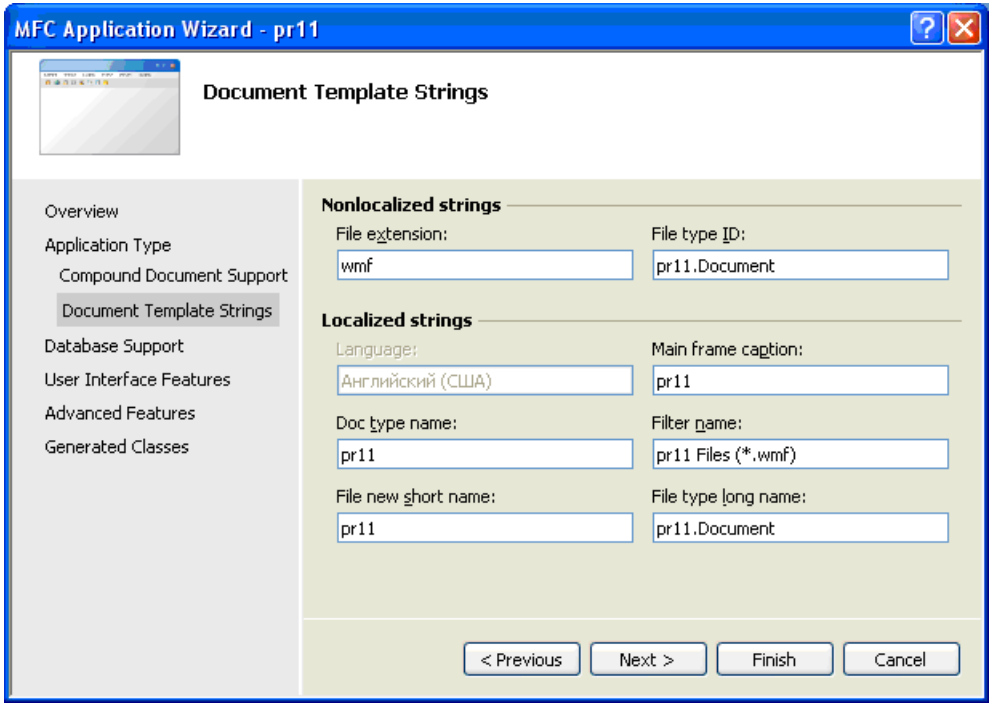


Рис. 11.1. Задание типа расширения файлов в настройках проекта

Создайте проект **pr11** аналогично проекту **pr10**, задав в свойствах проекта на вкладке **Document Template Strings** (Строки шаблона документа) в поле **File extension** (Расширение файла) расширение **wmf**, которое будет автоматически добавляться к именам файлов (это стандартное расширение Windows для метафайлов). Надо изменить следующие опции относительно изначально предложенных мастером:

- ❒ на вкладке **Application Type**:
  - SDI-документ;
  - без Unicode;
- ❒ на вкладке **Advanced Features**:
  - без печати и предварительного просмотра (снять флажок **Printing and print preview**);
- ❒ на вкладке **Document Template Strings:**
  - в поле **File extension** ввести **wmf** (рис. 11.1).

11.1. Описание программы

Если вы забыли задать расширение файлов в настройках проекта, то можно прописать нужное расширение вручную, заменив в ресурсах **String Table** (рис. 11.2) строку **IDR_MAINFRAME**:

```
pr11\n\npr11\n\n\npr11.Document\npr11.Document
```

на:

```
pr11\n\npr11\npr11 Files (*.wmf)\n.wmf\npr11.Document\npr11.Document
```

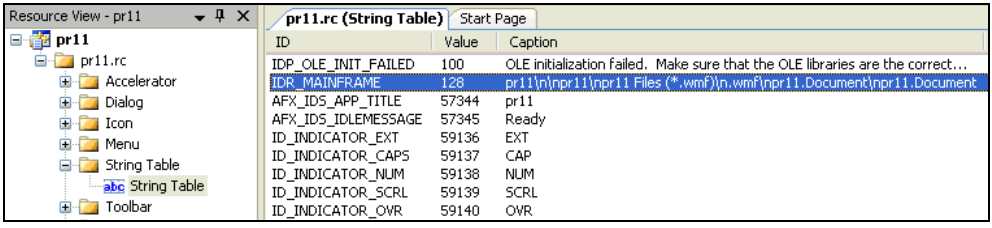


Рис. 11.2. Задание типа расширения файлов в таблице строк

11.1.1. Рисование графических изображений

Добавим код программы, который при удержании левой кнопки мыши будет рисовать прямую линию (от точки, где кнопку нажали, до точки, где кнопку отпустили). Для этого понадобится в класс представления добавить два обработчика левой кнопки мыши: нажатие (`WM_LBUTTONDOWN`) и отпускание (`WM_LBUTTONUP`) (рис. 11.3):

- в окне **Class View** вызвать контекстное меню для класса `Cpr11View` и выполнить команду **Properties**;
- в окне **Properties** выбрать вкладку **Messages**;

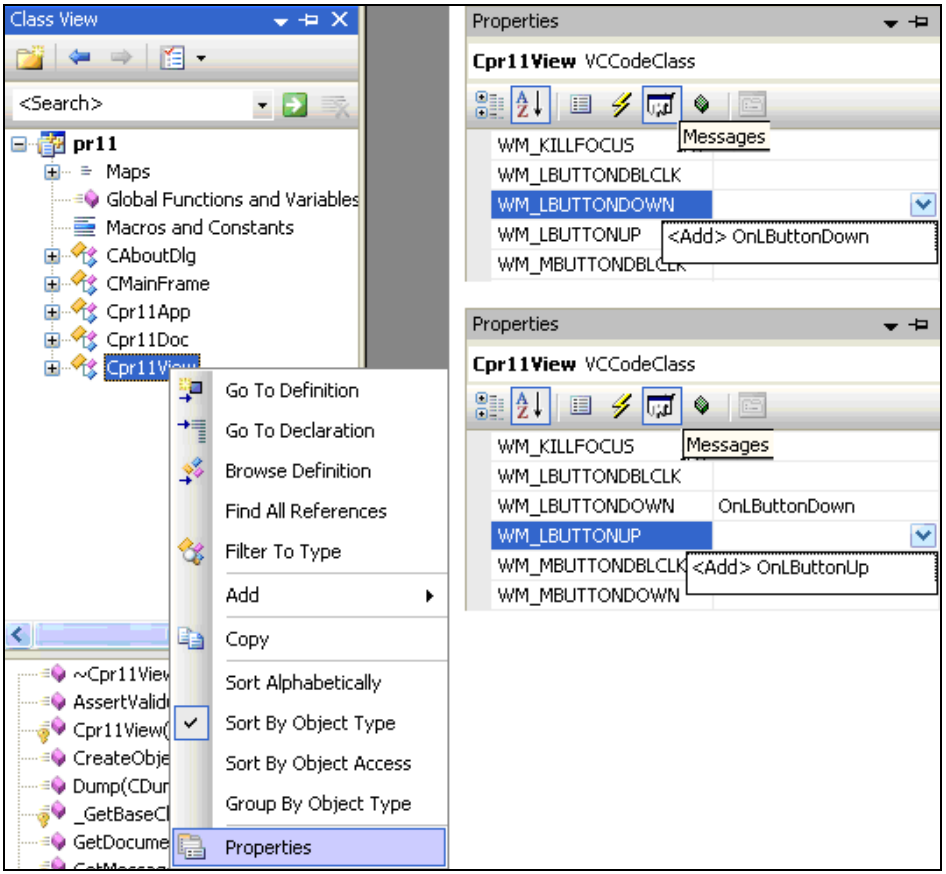


Рис. 11.3. Добавление обработки сообщений левой кнопки мыши

- в списке сообщений найти WM_LBUTTONDOWN и выбрать **<Add> OnLButtonDown**;
- в списке сообщений найти WM_LBUTTONUP и выбрать **<Add> OnLButtonUp**.

Изменения в файлах приведены в листингах 11.1 и 11.2.

Листинг 11.1. Изменения в файле pr11View.h для обработки нажатия левой кнопки мыши

```
// ...
class Cpr11View : public CView
{
// ...
public:
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
public:
   afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
};
```

Листинг 11.2. Изменения в файле pr11View.cpp для обработки нажатия левой кнопки мыши

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   ON_WM_LBUTTONDOWN()
   ON_WM_LBUTTONUP()
END_MESSAGE_MAP()
// ...
void Cpr11View::OnLButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default

   CView::OnLButtonDown(nFlags, point);
}

void Cpr11View::OnLButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
```

```
    CView::OnLButtonUp(nFlags, point);
}
```

Функция для обработки сообщения о нажатии левой кнопки мыши определена так:

```
afx_msg void CWnd::OnLButtonDown(
   UINT nFlags,           // Состояние кнопок в момент сообщения мыши
   CPoint point);         // Координаты курсора в момент сообщения мыши
```

Значения флагов кнопок (`nFlags`) могут принимать следующие значения:

- `MK_CONTROL` — в момент нажатия кнопки мыши была нажата клавиша <Ctrl>;
- `MK_LBUTTON` — была нажата левая кнопка мыши;
- `MK_MBUTTON` — была нажата средняя кнопка мыши;
- `MK_RBUTTON` — была нажата правая кнопка мыши;
- `MK_SHIFT` — в момент нажатия кнопки мыши была нажата клавиша <Shift>.

Функция для обработки сообщения об отпускании левой кнопки мыши определена так:

```
afx_msg void CWnd::OnLButtonUp(
   UINT nFlags,           // Состояние кнопок в момент сообщения мыши
   CPoint point);         // Координаты курсора в момент сообщения мыши
```

Значения флагов `nFlags` такие же, как и для `OnLButtonDown()` (кроме `MK_LBUTTON`).

Теперь добавим код для рисования линии. Изменения в файлах приведены в листингах 11.3 и 11.4.

Листинг 11.3. Изменения в файле pr11View.h для рисования линии в окне представления с помощью левой кнопки мыши

```
// ...
class Cpr11View : public CView
{
// ...
```

```
public:
   CPoint begin,              // Координаты начала линии
          end;                // Координаты конца линии
};
```

Листинг 11.4. Изменения в файле pr11View.cpp для рисования линии в окне представления с помощью левой кнопки мыши

```
// ...
void Cpr11View::OnLButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   begin = point;

   CView::OnLButtonDown(nFlags, point);
}
void Cpr11View::OnLButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   end = point;
   // Получить контекст устройства
   CClientDC *pDC = new CClientDC(this);
   // Нарисовать линию
   pDC->MoveTo(begin.x, begin.y);
   pDC->LineTo(end.x, end.y);

   begin = end;

   CView::OnLButtonUp(nFlags, point);
}
```

Класс для получения контекста устройства определен как:

```
class CClientDC : public CDC
```

Конструктор класса выглядит следующим образом:

```
explicit CClientDC::CClientDC(
   CWnd* pWnd);      // Указатель на окно, к чьей клиентской области надо
                     // получить контекст
```

Теперь изменим форму курсора со стрелки на перекрестье. Изменения в файле приведены в листинге 11.5.

Листинг 11.5. Изменения в файле pr11View.cpp для изменения формы курсора

```
// ...
BOOL Cpr11View::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Modify the Window class or styles here by modifying
   // the CREATESTRUCT cs
   cs.lpszClass = AfxRegisterWndClass(NULL, LoadCursor(NULL, IDC_CROSS),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return CView::PreCreateWindow(cs);
}
```

Результат работы программы показан на рис. 11.4.

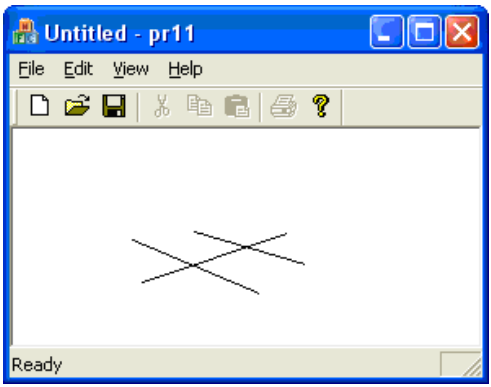


Рис. 11.4. Рисование линий в окне представления

Линия появляется на экране, только когда пользователь отпускает кнопку мыши (т. е. когда определены координаты конца линии). Чтобы можно было

видеть линию в процессе рисования, надо в класс представления добавить обработчик сообщения о перемещении мыши (`WM_MOUSEMOVE`) (рис. 11.5):

- в окне **Class View** вызвать контекстное меню для класса `Cpr11View` и выполнить команду **Properties**;
- в окне **Properties** выбрать вкладку **Messages**;
- в списке сообщений найти `WM_MOUSEMOVE` и выбрать **<Add> OnMouseMove**.

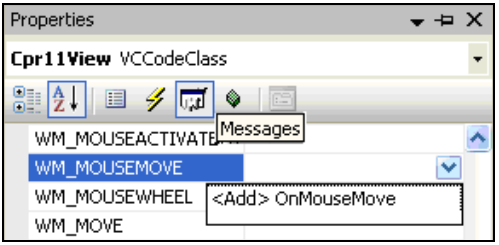


Рис. 11.5. Добавление обработки сообщения о перемещении мыши

Изменения в файлах приведены в листингах 11.6 и 11.7.

Листинг 11.6. Изменения в файле pr11View.h для обработки сообщения о перемещении мыши

```
// ...
class Cpr11View : public CView
{
// ...
public:
   afx_msg void OnMouseMove(UINT nFlags, CPoint point);
};
```

Листинг 11.7. Изменения в файле pr11View.cpp для обработки сообщения о перемещении мыши

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   // ...
```

```
    ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()
// ...
void Cpr11View::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    CView::OnMouseMove(nFlags, point);
}
```

Функция обработки сообщения о перемещении мыши определена как:

```
afx_msg void CWnd::OnMouseMove(
    UINT nFlags,            // Состояние кнопок в момент сообщения мыши
                            // (см. OnLButtonDown())
    CPoint point);          // Координаты курсора в момент сообщения мыши
```

Добавим обработку этого сообщения. Изменения в файле приведены в листингах 11.8 и 11.9.

Листинг 11.8. Изменения в файле pr11View.h для отображения перемещения мыши

```
// ...
class Cpr11View : public CView
{
// ...
public:
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    CPoint  oldmouse;           // Последние координаты перемещения мыши
};
```

Листинг 11.9. Изменения в файле pr11View.cpp для отображения перемещения мыши

```
// ...
void Cpr11View::OnLButtonDown(UINT nFlags, CPoint point)
{
```

```
   // TODO: Add your message handler code here and/or call default
   begin = point;
   oldmouse = point;

   CView::OnLButtonDown(nFlags, point);
}
// ...
void Cpr11View::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default

   int oldmode;                    // Старый бинарный растровый режим
   // Получить контекст устройства
   CClientDC *pDC = new CClientDC(this);
   // Если в момент перемещения нажата левая кнопка мыши
   if(nFlags && MK_LBUTTON)
   {
      oldmode = pDC->GetROP2();        // Сохранить старый растровый режим
      pDC->SetROP2(R2_NOT);            // Установить новый растровый режим
      // "Стирание" старой линии
      pDC->MoveTo(begin.x, begin.y);
      pDC->LineTo(oldmouse.x, oldmouse.y);
      // Рисование новой линии
      pDC->MoveTo(begin.x, begin.y);
      pDC->LineTo(point.x, point.y);

      oldmouse = point;
      pDC->SetROP2(oldmode);       // Восстановить старый растровый режим
   }
   CView::OnMouseMove(nFlags, point);
}
```

Функция получения текущего растрового режима выглядит следующим образом:

```
int CDC::GetROP2() const;
```

Функция установки растрового режима выглядит так:

```
int CDC::SetROP2(           // Предыдущий растровый режим
   int nDrawMode);          // Устанавливаемый растровый режим
```

Виды растровых режимов (`nDrawMode`) могут принимать следующие значения:

- `R2_BLACK` — пиксел всегда имеет черный цвет;
- `R2_WHITE` — пиксел всегда имеет белый цвет;
- `R2_NOP` — пиксел остается без изменений;
- `R2_NOT` — цвет пиксела является инвертированным цветом экрана;
- `R2_COPYPEN` — цвет пиксела совпадает с цветом пера;
- `R2_NOTCOPYPEN` — цвет пиксела является инвертированным цветом пера;
- `R2_MERGEPENNOT` — цвет пиксела = (`NOT` цвет экрана) `OR` цвет пера;
- `R2_MASKPENNOT` — цвет пиксела = (`NOT` цвет экрана) `AND` цвет пера;
- `R2_MERGENOTPEN` — цвет пиксела = (`NOT` цвет пера) `OR` цвет экрана;
- `R2_MASKNOTPEN` — цвет пиксела равен (`NOT` цвет пера) `AND` цвет экрана;
- `R2_MERGEPEN` — цвет пиксела = цвет пера `OR` цвет экрана;
- `R2_NOTMERGEPEN` — инвертирование `R2_MERGEPEN`. Цвет пиксела = `NOT` (цвет пера `OR` цвет экрана);
- `R2_MASKPEN` — цвет пиксела = цвет пера `AND` цвет экрана;
- `R2_NOTMASKPEN` — инвертирование `R2_MASKPEN`. Цвет пиксела = `NOT` (цвет пера `AND` цвет экрана);
- `R2_XORPEN` — цвет пиксела = цвет пера `XOR` цвет экрана;
- `R2_NOTXORPEN` — инвертирование `R2_XORPEN`. Цвет пиксела = `NOT` (цвет пера `XOR` цвет экрана).

В нашей программе выбран режим `R2_NOT`. Когда рисуется линия, цвет экрана в этом месте черный, поэтому в этом режиме мы поверх черной линии рисуем инвертированную — белую, т. е. как бы "стираем" старую линию. А новая линия рисуется черным на белом.

11.1.2. Рисование графических изображений с использованием метафайла

Метафайлом называется хранящийся в памяти объект, поддерживающий собственный контекст устройства. Любые операции, выполняемые с контек-

стом устройства, можно продублировать в метафайле. Чтобы отобразить хранимое в метафайле изображение, его надо воспроизвести (play). Изменения в файлах для работы с метафайлом приведены в листингах 11.10—11.12.

Листинг 11.10. Изменения в файле pr11Doc.h для работы с метафайлом

```
// ...
class Cpr11Doc : public CDocument
{
// ...
public:
   CMetaFileDC *pmetaDC;              // Указатель на метафайл
};
```

Листинг 11.11. Изменения в файле pr11Doc.cpp для работы с метафайлом

```
// ...
Cpr11Doc::Cpr11Doc()
{
   // TODO: add one-time construction code here
   pmetaDC = new CMetaFileDC;
   pmetaDC->Create();
}
```

Листинг 11.12. Изменения в файле pr11View.cpp для работы с метафайлом

```
// ...
void Cpr11View::OnDraw(CDC* pDC)
{
   Cpr11Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if(!pDoc)
      return;

   // TODO: add draw code for native data here
```

```
   // Закрыть контекст устройства и получить идентификатор метафайла
   HMETAFILE hmeta = pDoc->pmetaDC->Close();
   // Отобразить метафайл
   pDC->PlayMetaFile(hmeta);

   // Создать контекст устройства ("чистый" — для дальнейшей работы)
   pDoc->pmetaDC->Create();
   // Проиграть метафайл (чтобы контекст был не "чистый",
   // а содержал предыдущие данные)
   pDoc->pmetaDC->PlayMetaFile(hmeta);
   // Удалить идентификатор метафайла
   DeleteMetaFile(hmeta);
}
// ...
void Cpr11View::OnLButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   end = point;

   // Получить контекст устройства
   // CClientDC *pDC = new CClientDC(this);
   // Нарисовать линию
   // pDC->MoveTo(begin.x, begin.y);
   // pDC->LineTo(end.x, end.y);

   // Рисование в метафайле
   Cpr11Doc* pDoc = GetDocument();
   pDoc->pmetaDC->MoveTo(begin.x, begin.y);
   pDoc->pmetaDC->LineTo(end.x, end.y);

   begin = end;

   CView::OnLButtonUp(nFlags, point);
}
```

Класс метафайла определен как:

```
class CMetaFileDC : public CDC
```

Создание метафайла выполняется с помощью фукнции:

```
BOOL CMetaFileDC::Create(                // 0 — ошибка
   LPCTSTR lpszFilename = NULL);         // Имя метафайла. Если NULL — то
                                         // метафайл создается в памяти
```

Для того чтобы отобразить (проиграть) метафайл на экран, его надо закрыть и получить его идентификатор.

Функция закрытия метафайла выглядит так:

```
HMETAFILE CMetaFileDC::Close();          // Идентификатор метафайла или
                                         // NULL — при ошибке
```

Функция проигрывания метафайла следующая:

```
BOOL CDC::PlayMetaFile(                  // 0 — ошибка
   HMETAFILE hMF);                       // Идентификатор метафайла
```

Удаление идентификатора метафайла выполняется функцией:

```
BOOL DeleteMetaFile(                // 0 — ошибка
   HMETAFILE hmf);                  // Идентификатор метафайла
```

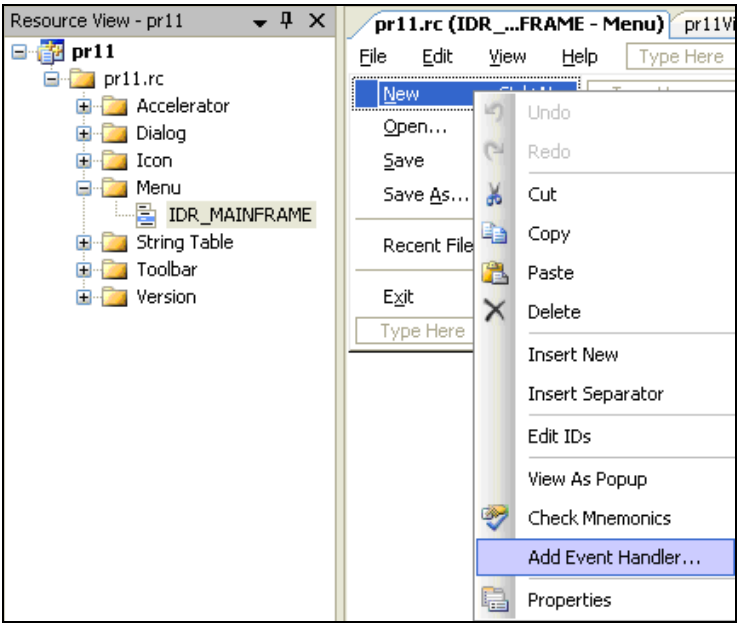
11.1.3. Сохранения и загрузка метафайла на диске

Создание нового файла

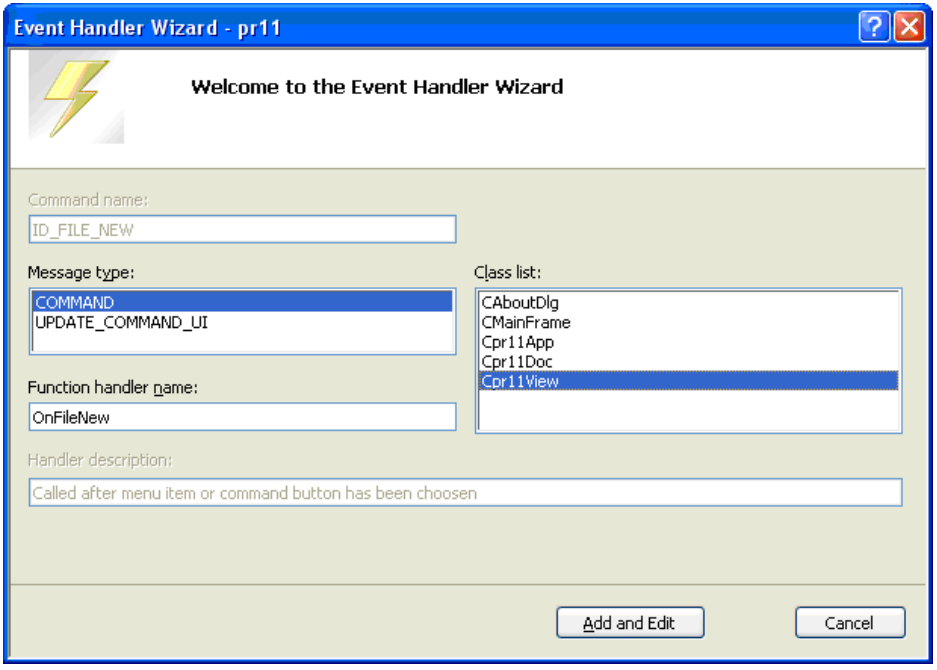
Добавим свою обработку пункта меню **File | New** (Файл | Создать) в класс представления `Cpr11View` (рис. 11.6):

- в окне **Resource View** выбрать ресурс меню `IDR_MAINFRAME`;
- в ресурсе меню вызвать контекстное меню для пункта **File | New** и выполнить команду **Add Event Handler**;
- в окне **Event Handler Wizard** в списке **Class list** выбрать класс `Cpr11View`.

Изменения в файлах приведены в листингах 11.13 и 11.14. Файл ресурсов Resource.h не изменился, т. к. мы воспользовались стандартным идентификатором этого пункта меню (`ID_FILE_NEW`).



а



б

Рис. 11.6. Добавление обработки сообщения меню (*а*)
и функция обработки сообщения меню **New** (*б*)

Листинг 11.13. Изменения в файле pr11View.h для добавления меню создания нового файла

```
// ...
class Cpr11View : public CView
{
// ...
public:
   afx_msg void OnFileNew();
};
```

Листинг 11.14. Изменения в файле pr11View.cpp для добавления меню создания нового файла

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   // ...
   ON_COMMAND(ID_FILE_NEW, &Cpr11View::OnFileNew)
END_MESSAGE_MAP()
// ...
void Cpr11View::OnFileNew()
{
   // TODO: Add your command handler code here
}
```

Теперь надо сделать так, чтобы при выборе этого меню содержимое окна очищалось, а в заголовке окна не было задано имя файла (как и при запуске программы —**Untitled - pr11**, см. рис. 11.4). Изменения в файлах приведены в листингах 11.15—11.18.

Листинг 11.15. Изменения в файле pr11.h для обработки меню создания нового файла

```
// ...
class Cpr11App : public CWinApp
{
// ...
```

```
public:
   CString deffile;      // Заголовок окна для файла без названия
                         // (Untitled — pr11)
};
```

Листинг 11.16. Изменения в файле pr10.cpp для обработки меню создания нового файла

```
// ...
BOOL Cpr10App::InitInstance()
{
   // ...
   // Получить заголовок нового (безымянного) файла
   m_pMainWnd->GetWindowTextA(deffile);

   return TRUE;
}
```

Листинг 11.17. Изменения в файле pr11View.h для обработки меню создания нового файла

```
// ...
class Cpr11View : public CView
{
// ...
public:
   afx_msg void OnFileNew();
   CString file;                    // Имя последнего загруженного
                                    // (или сохраненного) файла
};
```

Листинг 11.18. Изменения в файле pr11View.cpp для обработки меню создания нового файла

```
// ...
void Cpr11View::OnFileNew()
```

```
{
   // TODO: Add your command handler code here
   Cpr11Doc* pDoc = GetDocument();
   file = "";                  // Нет последнего загруженного или
                               // сохраненного файла
   // Заголовок окна — без имени файла (Untitled — pr11)
   theApp.m_pMainWnd->SetWindowTextA(theApp.deffile);
   // Закрыть старый метафайл (с данными)
   pDoc->pmetaDC->Close();
   // Создать новый метафайл (чистый)
   pDoc->pmetaDC->Create();
   // Перерисовать окно представления
   this->Invalidate();
}
```

Получить текст из окна (в данном случае — его заголовок) можно с помощью функции:

```
void CWnd::GetWindowText(
   CString& rString) const;              // Строка, куда будет записан
                                         // полученный текст
```

Задать текст окна (в данном случае — заголовок) можно функцией:

```
void CWnd::SetWindowText(
   LPCTSTR lpszString);                  // Строка с текстом
```

Открытие файла

Аналогичным образом *(см. разд. Создание нового файла)* добавим в класс представления обработку меню **File | Open**. Изменения в файлах приведены в листингах 11.19 и 11.20.

Листинг 11.19. Изменения в файле pr11View.h для добавления меню открытия файла

```
// ...
class Cpr11View : public CView
{
```

```
// ...
public:
   afx_msg void OnFileOpen();
};
```

Листинг 11.20. Изменения в файле pr11View.cpp для добавления меню открытия файла

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   // ...
   ON_COMMAND(ID_FILE_OPEN, &Cpr11View::OnFileOpen)
END_MESSAGE_MAP()
// ...
void Cpr11View::OnFileOpen()
{
   // TODO: Add your command handler code here
   Cpr11Doc* pDoc = GetDocument();
   // Создать объект стандартного диалога выбора файлов для открытия
   CFileDialog fd(true);
   // Выбранный пользователем файл должен существовать на диске
   fd.m_ofn.Flags |=OFN_FILEMUSTEXIST;
   // Новый (не стандартный) заголовок окна диалога
   fd.m_ofn.lpstrTitle = "Open WMF file";
   // Фильтр — только файлы *.wmf
   fd.m_ofn.lpstrFilter = "Grafic files( *.wmf )\0*.wmf";
   // В окне диалога отразить текущий каталог
   fd.m_ofn.lpstrInitialDir = NULL;
   // Запустить диалог
   if(fd.DoModal()== IDCANCEL)  // Если пользователь отказался от выбора
      return;
   file = fd.m_ofn.lpstrFile;  // Получить полное (с путем) имя
                               // выбранного файла
   // Получить короткое (без пути) имя файла (для заголовка окна)
   CString shortfile = fd.m_ofn.lpstrFileTitle;
```

```
    shortfile += " — ";
    shortfile += theApp.m_pszAppName;        // Добавить имя приложения
    // Установить заголовок окна (имя_файла.wmf — pr11)
    theApp.m_pMainWnd->SetWindowTextA(shortfile);

    // Получить дескриптор выбранного метафайла
    HMETAFILE hmeta = GetMetaFile(file);
    pDoc->pmetaDC->Close();              // Закрыть старый метафайл
    pDoc->pmetaDC->Create();             // Создать новый
    // Связать его с полученным дескриптором
    pDoc->pmetaDC->PlayMetaFile(hmeta);
    DeleteMetaFile(hmeta);               // Удалить дескриптор
    this->Invalidate();                 // Перерисовать окно представления
}
```

Класс диалога для выбора файла определен как:

```
class CFileDialog : public CCommonDialog
class CCommonDialog : public CDialog
```

Конструктор класса выглядит следующим образом:

```
explicit CFileDialog::CFileDialog(
    BOOL bOpenFileDialog,           // Тип окна диалога (TRUE — Open,
                                    // FALSE — Save As)
    LPCTSTR lpszDefExt = NULL,      // Умолчание для расширения файла
    LPCTSTR lpszFileName = NULL,    // Имя файла
    DWORD dwFlags = OFN_HIDEREADONLY |   // Режимы выбора файла
                    OFN_OVERWRITEPROMPT,
    LPCTSTR lpszFilter = NULL,      // Строка для задания фильтров выбора
    CWnd* pParentWnd = NULL,        // Указатель на родительское окно
    DWORD dwSize = 0);              // Размер структуры OPENFILENAME
```

Величина размера структуры `OPENFILENAME` зависит от версии операционной системы. Если задать `dwSize = 0`, то MFC самостоятельно определит размер.

Структура с настройками вида окна диалога выбора файла определена так:

```
typedef struct tagOFN
{
    DWORD lStructSize;       // Размер структуры в байтах
```

```
HWND hwndOwner;          // Дескриптор родительского окна (или NUUL)
HINSTANCE hInstance;     // Идентификатор блока памяти
LPCTSTR lpstrFilter;      //Указатель на буфер с одной (или больше)
                          // парой строк (первая строка — название
                          // фильтра, вторая —  параметры
                          // для фильтрации
LPTSTR lpstrCustomFilter; // Буфер для сохранения выбранного
                          // пользователем фильтра для последующих
                          // запусков окна диалога
                          // (если NULL — не сохранять)
DWORD nMaxCustFilter;     // Размер буфера lpstrCustomFilter
DWORD nFilterIndex;       // Индекс (начиная с 1) текущего
                          // выбранного фильтра (если nFilterIndex = 0
                          // и lpstrCustomFilter = 0, то
                          // используется первый фильтр из буфера
                          // lpstrFilter)
LPTSTR lpstrFile;         // Полное имя файла (с путем)
DWORD nMaxFile;           // Размер lpstrFile
LPTSTR lpstrFileTitle;    // Имя файла (без пути)
DWORD nMaxFileTitle;      // Размер lpstrFileTitle
LPCTSTR lpstrInitialDir;  // Строка с начальным каталогом.
                          // По умолчанию берется каталог
                          // "Мои документы". Если задать NULL,
                          // то будет текущий каталог (но только при
                          // условии, что там есть файлы, указанные
                          // в фильтре. Если таких файлов там нет, то
                          // будет каталог "Мои документы")
LPCTSTR lpstrTitle;       // Заголовок окна диалога. Если NULL,
                          // то будет умолчание (Open или Save As)
DWORD Flags;              // Режимы выбора файлов
WORD nFileOffset;         // Смещение первого символа имени файла
                          // относительно начала lpstrFile
WORD nFileExtension;      // Смещение первого символа расширения файла
                          // относительно lpstrDefExt
LPCTSTR lpstrDefExt;      // Расширения имени файла, используемое по
                          // умолчанию
```

```
   LPARAM lCustData;          // Данные, которые будут передаваться
                              // функции фильтра lpfnHook
   LPOFNHOOKPROC lpfnHook;    // Указатель на функцию фильтра
   LPCTSTR lpTemplateName;    // Имя ресурса шаблона диалога,
                              // определенного в hInstance
   #if(_WIN32_WINNT >= 0x0500)
      void *  pvReserved;
      DWORD  dwReserved;
      DWORD  FlagsEx;
   #endif    // (_WIN32_WINNT >= 0x0500)
} OPENFILENAME, *LPOPENFILENAME;
```

Значения флагов выбора файлов (`Flags`) могут быть такими:

- `OFN_ALLOWMULTISELECT` — разрешает выбор нескольких файлов одновременно. При выборе пользователем более одного файла `lpstrFile` будет содержать путь к текущему каталогу и имена файлов. Параметр `nFileOffset` будет содержать смещение до первого имени файла в `lpstrFile`;
- `OFN_CREATEPROMPT` — если пользователь задает несуществующий файл, то выводится блок диалога с предложением создать файл;
- `OFN_DONTADDTORECENT` — не позволяет добавлять связь к выбранному файлу в файловом системном каталоге;
- `OFN_ENABLEHOOK` — разрешает использовать функцию фильтра, адрес которой указан в `lpfnHook`;
- `OFN_ENABLEINCLUDENOTIFY` — заставляет диалоговое окно посылать сообщение `CDN_INCLUDEITEM` в вашу процедуру захвата `OFNHookProc`, когда пользователь открывает папку;
- `OFN_ENABLESIZING` — позволяет менять размеры диалогового окна с помощью мыши или клавиатуры.
- `OFN_ENABLETEMPLATE` — указывает на то, что поле `lpTemplateName` содержит указатель на имя ресурса шаблона блока диалога в `hInstance`;
- `OFN_ENABLETEMPLATEHANDLE` — указывает на то, что поле `hInstance` используется для идентификации блока памяти, содержащего предварительно загруженный шаблон блока диалога. Содержимое `lpTemplateName` игнорируется;
- `OFN_EXPLORER` — при создании блоков диалогов **Open** и **Save As** используется новый стиль EXPLORER;

- `OFN_EXTENSIONDIFFERENT` — устанавливается после завершения диалога и указывает, что расширение выбранного файла отличается от того, что было задано в `lpstrDefExt`. Этот флаг не устанавливается, если файл не имеет расширения или `lpstrDefExt = NULL`;
- `OFN_FILEMUSTEXIST` — пользователь может вводить только имена существующих файлов, иначе будет выдаваться соответствующее предупреждение. Если этот флаг задан, то автоматически устанавливается флаг `OFN_PATHMUSTEXIST`;
- `OFN_FORCESHOWHIDDEN` — заставляет систему показывать скрытые файлы;
- `OFN_HIDEREADONLY` — предписывает убрать из блока диалога флаг **Read Only** (только чтение);
- `OFN_LONGNAMES` — предписывает блоку диалога отображать длинные имена файлов;
- `OFN_NOCHANGEDIR` — устанавливает первоначальный текущий каталог, если пользователь изменил его при поиске файлов;
- `OFN_NODEREFERENCELINKS` — предписывает блоку диалога вернуть путь и имя выбранного сокращения (файла с расширением *.lnk);
- `OFN_NOLONGNAMES` — определяет, что в поле имени файла не отображаются длинные имена файлов;
- `OFN_NONETWORKBUTTON` — скрывает и блокирует кнопку **Network** (Сетевое окружение);
- `OFN_NOREADONLYRETURN` — определяет, что выбранный файл не имеет атрибута "только чтение" и не располагается в защищенном от записи каталоге;
- `OFN_NOTESTFILECREATE` — определяет, что файл не создается перед закрытием диалога, а также определяет отсутствие проверки на защиту от записи, переполнение диска или сетевую защиту;
- `OFN_NOVALIDATE` — определяет, что стандартные блоки диалога допускают наличие запрещенных символов в именах возвращаемых файлов;
- `OFN_OVERWRITEPROMPT` — если файл уже существует, то при выборе его в **Save As** появляется запрос на перезапись файла;
- `OFN_PATHMUSTEXIST` — пользователь может выбирать только существующий путь к файлу, иначе выдается соответствующее сообщение;
- `OFN_READONLY` — устанавливает в блоке диалога флаг **Read Only**;

- OFN_SHAREAWARE — устанавливается после возвращения из функции OpenFile() и указывает, что произошла ошибка при совместном доступе к файлу в сети;
- OFN_SHOWHELP — в блоке диалога отображается кнопка **Help** (Справка).

Если в текущем каталоге еще нет файлов с заданным расширением, а надо сделать его по умолчанию, можно воспользоваться функцией _getcwd():

```
#include "direct.h"                  // Для _getcwd()
// ...
void Cpr11View::OnFileOpen()
{
   // ...
   char *buf;
   buf = _getcwd(NULL, 0);           // Получить текущий каталог
                                     // (например, c:\pr11)
   CString dir = buf;
   dir += "\\default.wmf";
   CFileDialog fd(true, "wmf", dir);
   // ...
}
```

Функция получения текущего каталога определена так:

```
char *_getcwd(         // Указатель на строку с текущим каталогом
                       // или 0 — при ошибке
   char *buffer,       // Указатель на строку с текущим каталогом
                       // (можно NULL)
   int maxlen);        // Максимальная длина буфера buffer (можно 0)
```

Член класса, дающий доступ к полям структуры OPENFILENAME:

```
OPENFILENAME CFileDialog::m_ofn
```

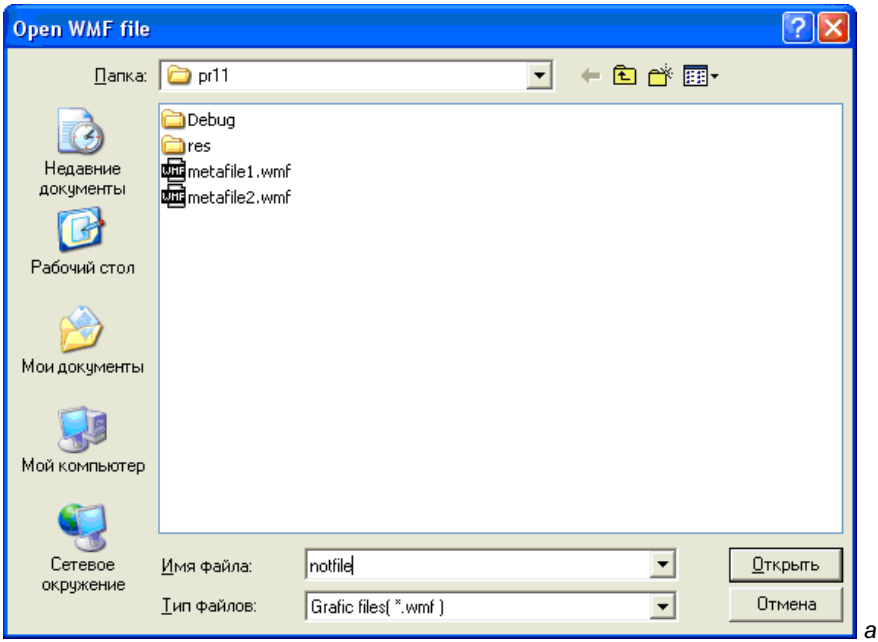
Член класса, содержащий имя приложения:

```
LPCTSTR CWinApp::m_pszAppName;
```

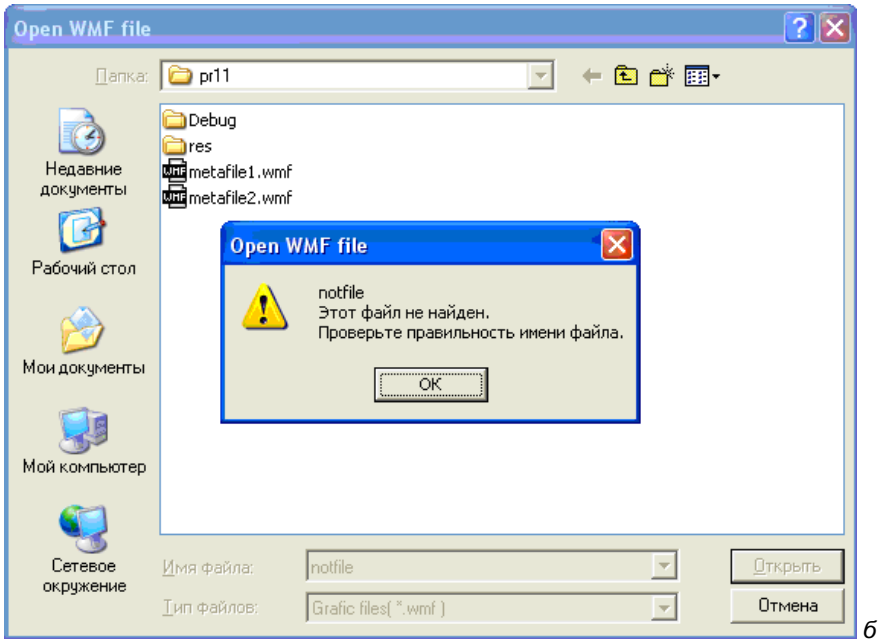
Получить дескриптор метафайла по его имени можно функцией:

```
HMETAFILE GetMetaFile(        // Дескриптор метафайла
   LPCTSTR lpszMetaFile);     // Имя метафайла
```

Результат работы программы показан на рис. 11.7.

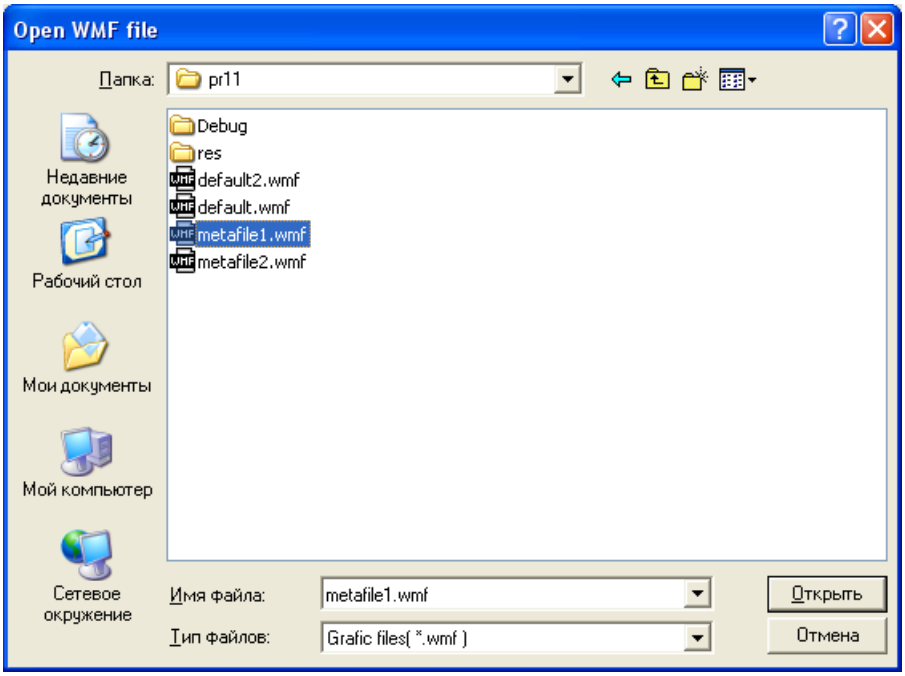


а

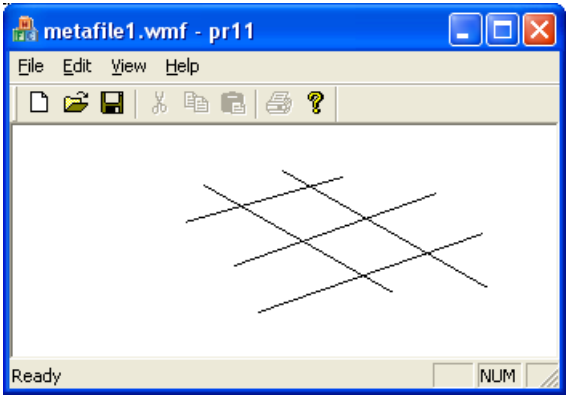


б

Рис. 11.7. Задание несуществующего файла для открытия (*а*), сообщение при попытке открытия несуществующего файла (*б*)



в



г

Рис. 11.7. Выбор файла (*в*),
открытие файла в окне представления (*г*)

Если надо установить несколько фильтров, это делается так (рис. 11.8):

```
fd.m_ofn.lpstrFilter = "Grafic files(*.wmf)\0*.wmf\0All(*.*)\0*.*";
```

Приведем также несколько полезных функций для получения имени файла, выбранного пользователем в блоке диалога.

Получить полное имя файла (включая путь) можно с помощью функции:

```
CString CFileDialog::GetPathName() const;
```

Получить имя файла с расширением:

```
CString CFileDialog::GetFileName() const;
```

Получить расширение имени файла:

```
CString CFileDialog::GetFileExt() const;
```

Получить имя файла без расширения:

```
CString CFileDialog::GetFileTitle() const;
```

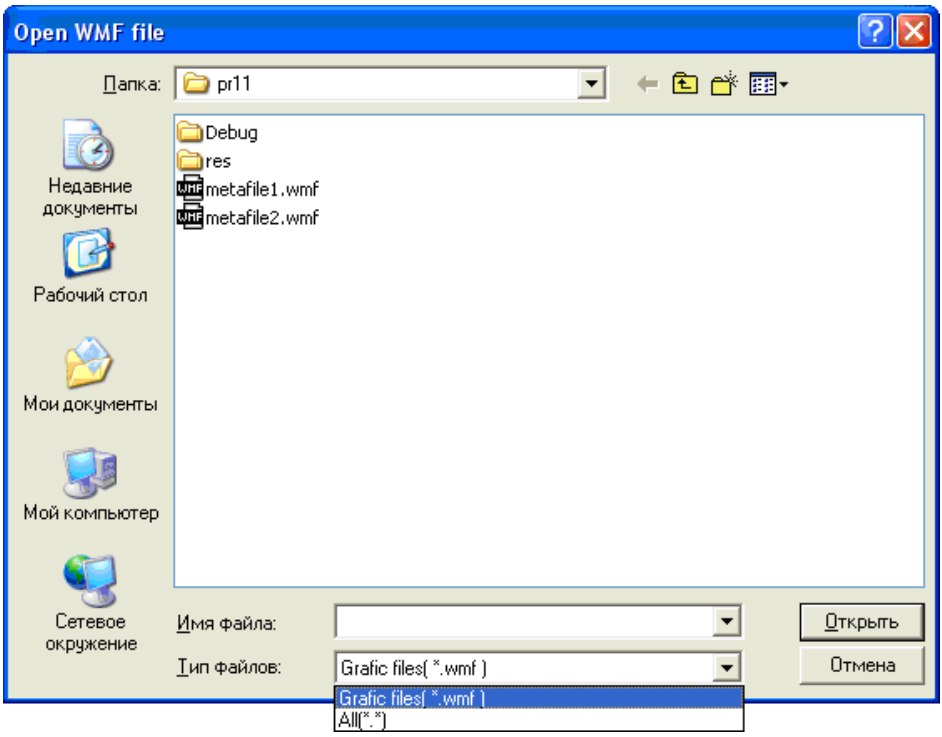
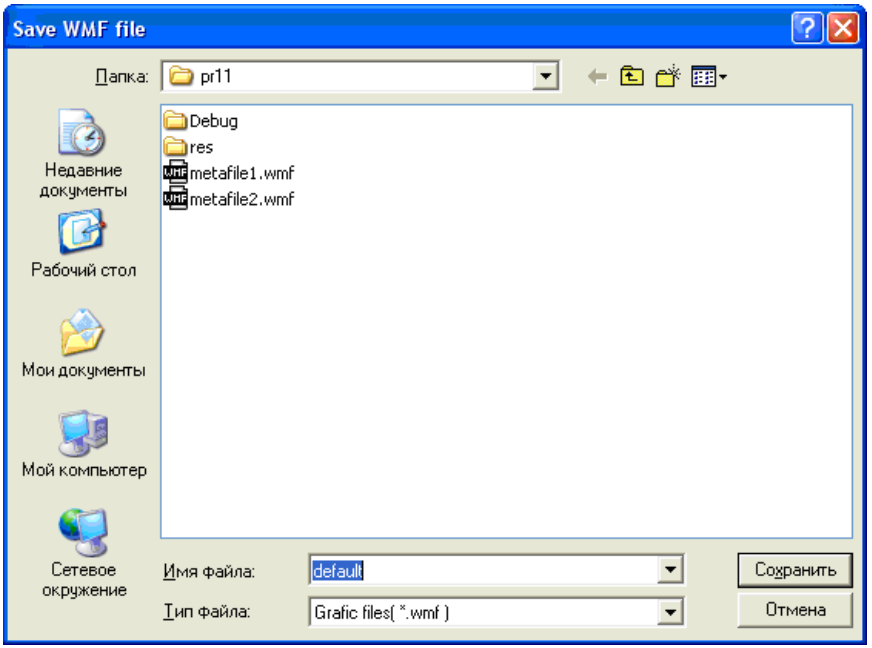


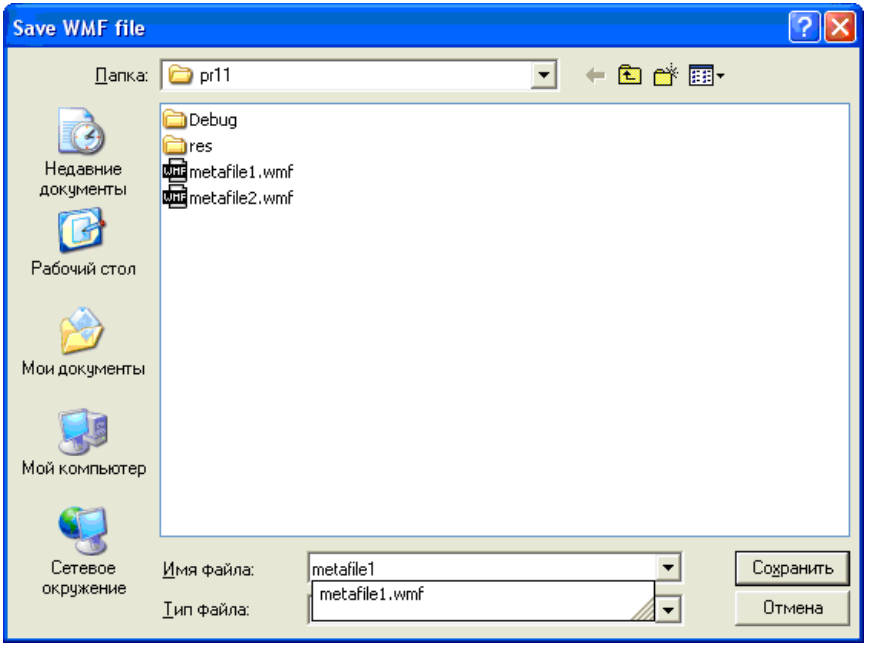
Рис. 11.8. Создание нескольких фильтров

Сохранение файла

Теперь добавим в класс представления обработку меню **File | Save** *(см. разд. Создание нового файла)*. Изменения в файлах приведены в листингах 11.21 и 11.22.

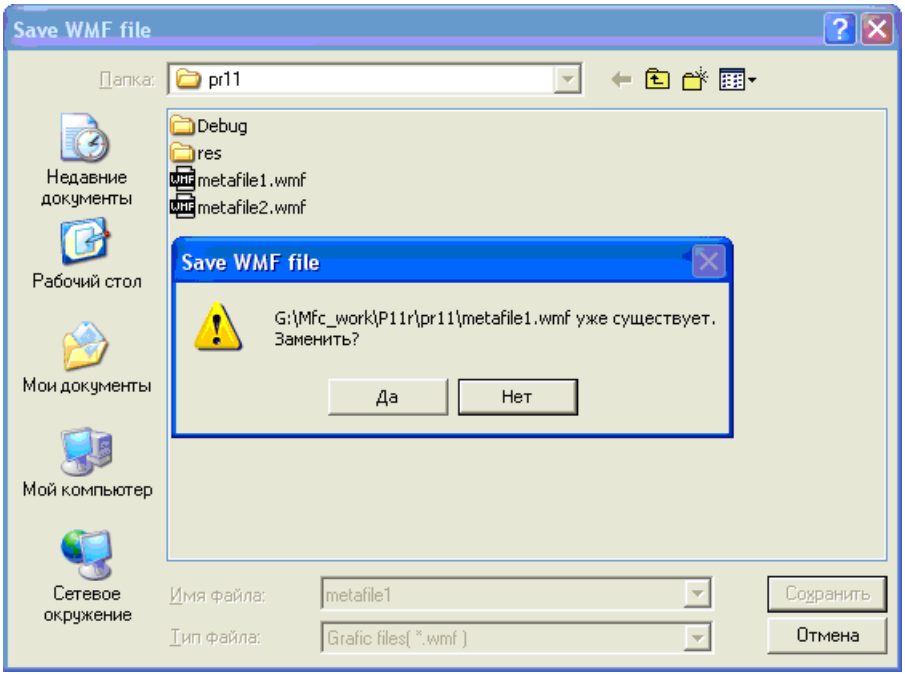


а

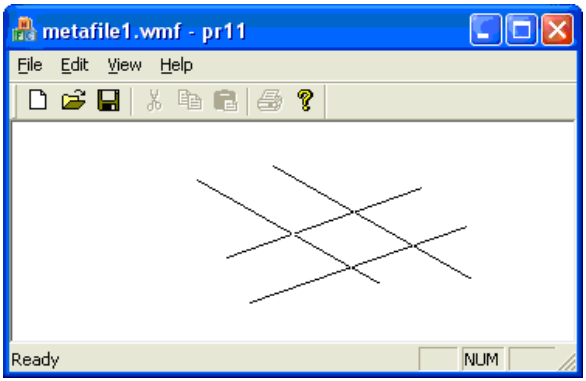


б

Рис. 11.9. Имя файла для записи по умолчанию (*а*), выбор для записи уже существующего файла (*б*)



в



г

Рис. 11.9. Сообщение при попытке записи в существующий файл (*в*) и новый файл, сохраненный на диске (*г*)

Листинг 11.21. Изменения в файле pr11View.h для добавления меню сохранения файла

```
// ...
class Cpr11View : public CView
```

```
{
// ...
public:
   afx_msg void OnFileSave();
};
```

Листинг 11.22. Изменения в файле pr11View.cpp для добавления меню сохранения файла

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   // ...
   ON_COMMAND(ID_FILE_SAVE, &Cpr11View::OnFileSave)
END_MESSAGE_MAP()
// ...
void Cpr11View::OnFileSave()
{
   // TODO: Add your command handler code here
   Cpr11Doc* pDoc = GetDocument();

   static CString oldfile = "?";  // Имя последнего сохраненного файла
   if(oldfile != file)            // Если открытый файл еще не сохраняли
   {
      CFileDialog fd(false, "wmf", "default");     // Диалог Save As
      fd.m_ofn.lpstrTitle = "Save WMF file";
      fd.m_ofn.lpstrFilter = "Grafic files(*.wmf)\0*.wmf";
      fd.m_ofn.lpstrInitialDir = NULL;
      if(fd.DoModal()== IDCANCEL)
         return;
      file = oldfile = fd.m_ofn.lpstrFile;        // Получить имя файла
      // Формирование заголовка окна
      CString shortfile = fd.m_ofn.lpstrFileTitle;
      shortfile += " — ";
      shortfile += theApp.m_pszAppName;
```

```
        theApp.m_pMainWnd->SetWindowTextA(shortfile);
    }

    HMETAFILE hmeta = pDoc->pmetaDC->Close();  // Закрыть метафайл
    // Скопировать метафайл в файл на диске
    hmeta = CopyMetaFile(hmeta, file);
    pDoc->pmetaDC->Create();                   // Создать новый метафайл
    // Связать его с дескриптором старого (чтобы новый не был чистым)
    pDoc->pmetaDC->PlayMetaFile(hmeta);
    DeleteMetaFile(hmeta);          // Удалить дескриптор старого метафайла
}
```

Функция копирования метафайла на диск выглядит так:

```
HMETAFILE CopyMetaFile(     // Дескриптор метафайла
    HMETAFILE hmfSrc,       // Дескриптор метафайла
    LPCTSTR lpszFile);      // Имя файла на диске, куда будет записан
                            // метафайл
```

При работе этой функции "затирается" значение `hmfSrc`, поэтому для нормальной дальнейшей работы с метафайлом надо сохранить его дескриптор в качестве возвращаемого значения.

Результаты работы программы показаны на рис. 11.9. Был создан новый файл и выбрана команда **File | Save**.

Сохранение файла под другим именем

Теперь осталось только добавить в класс представления обработку меню **File | Save As** *(см. разд. Создание нового файла)*. Изменения в файлах приведены в листингах 11.23 и 11.24.

Листинг 11.23. Изменения в файле pr11View.h для добавления и обработки меню сохранения файла под другим именем

```
// ...
class Cpr11View : public CView
{
```

```
// ...
public:
   afx_msg void OnFileSaveAs();
};
```

Листинг 11.24. Изменения в файле pr11View.cpp для добавления и обработки меню сохранения файла под другим именем

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   // ...
   ON_COMMAND(ID_FILE_SAVE_AS, &Cpr11View::OnFileSaveAs)
END_MESSAGE_MAP()
// ...
void Cpr11View::OnFileSave()
{
   // TODO: Add your command handler code here
   file = "";
   OnFileSave();
}
```

11.1.4. Рисуем красиво

Изменим текст программы так, чтобы можно было рисовать не просто прямую линию, а со шлейфом. Изменения в файле приведены в листинге 11.25.

Листинг 11.25. Изменения в файле pr11View.cpp для более красивого рисования линий

```
// ...
void Cpr11View::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   int oldmode;                          // Старый бинарный растровый режим
   // Получить контекст устройства
```

```
    CClientDC *pDC = new CClientDC(this);
    Cpr11Doc *pDoc = this->GetDocument();

    // Если в момент перемещения нажата левая кнопка мыши
    if(nFlags && MK_LBUTTON)
    {
       oldmode = pDC->GetROP2();       // Сохранить старый растровый режим
       pDC->SetROP2(R2_NOT);           // Установить новый растровый режим
       // "Стирание" старой линии
       pDC->MoveTo(begin.x, begin.y);
       pDC->LineTo(oldmouse.x, oldmouse.y);
      pDoc->pmetaDC->MoveTo(begin.x, begin.y);
       pDoc->pmetaDC->LineTo(oldmouse.x, oldmouse.y);
       // Рисование новой линии
       pDC->MoveTo(begin.x, begin.y);
       pDC->LineTo(point.x, point.y);
      pDoc->pmetaDC->MoveTo(begin.x, begin.y);
       pDoc->pmetaDC->LineTo(point.x, point.y);

      //oldmouse = point;
       begin = point;

       pDC->SetROP2(oldmode);      // Восстановить старый растровый режим
    }

    CView::OnMouseMove(nFlags, point);
}
```

Теперь можно рисовать красивые фигуры. Но если пользователь в окне диалога **Open** открывает файл не с помощью кнопки **Открыть**, а с помощью двойного щелчка мыши по выбираемому файлу, то на экране появляется дополнительная линия (рис. 11.10). Чтобы этого избежать, добавим код, представленный в листингах 11.26 и 11.27. Результаты работы программы показаны на рис. 11.11.

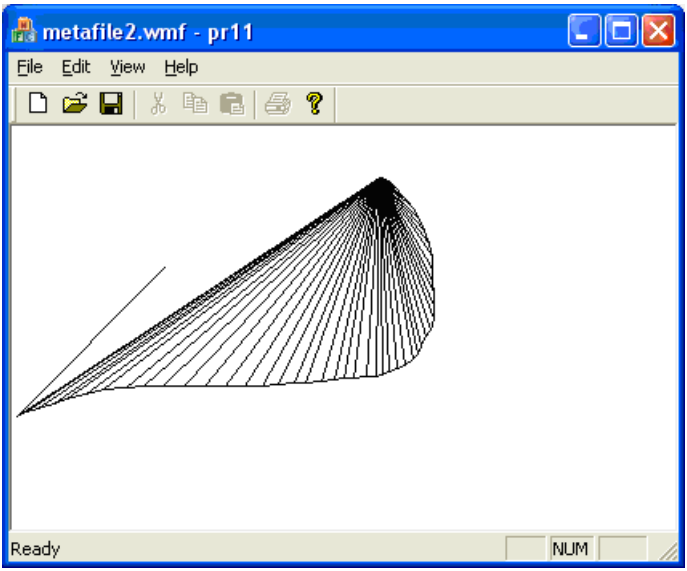


Рис. 11.10. Появление лишней линии при двойном щелчке мыши в окне выбора файлов

Листинг 11.26. Изменения в файле pr11View.h для устранения появления лишней линии при открытии файла

```
// ...
class Cpr11View : public CView
{
   // ...
public:
   afx_msg void OnFileSaveAs();
   int fmove;              // Флаг для обработки двойного щелчка мыши при
                           // выборе файла
};
```

Листинг 11.27. Изменения в файле pr11View.cpp для устранения появления лишней линии при открытии файла

```
// ...
Cpr11View::Cpr11View()
```

```
{
   // TODO: add construction code here
   fmove = 0;
}
// ...
void Cpr11View::OnMouseMove(UINT nFlags, CPoint point)
{
   // ...
   // Если в момент перемещения нажата левая кнопка мыши
   if(nFlags && MK_LBUTTON)
   {
      if(fmove == 1)
      {
         static int i;
         if(i == 2)
         {
            fmove=0;
            i=0;
         }
         i++;
         CView::OnMouseMove(nFlags, point);
         return;
      }

      oldmode = pDC->GetROP2();     // Сохранить старый растровый режим
      // ...
   }
   CView::OnMouseMove(nFlags, point);
}
// ...
void Cpr11View::OnFileOpen()
{
   // ...
   fmove = 1;
   this->Invalidate();              // Перерисовать окно представления
}
```

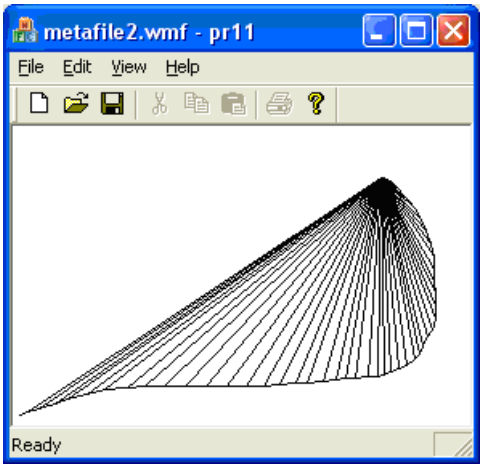


Рис. 11.11. Окончательный вариант работы программы

11.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 11.28. Файл pr11.h (объявление класса приложения)

```
// ...
class Cpr11App : public CWinApp
{
// ...
public:
   CString deffile;       // Заголовок окна для файла без названия
                          // (Untitled - pr11)
};
```

Листинг 11.29. Файл pr11.cpp (определение класса приложения)

```
// ...
BOOL Cpr11App::InitInstance()
```

```
{
   // ...
   // Получить заголовок нового (безымянного) файла
   m_pMainWnd->GetWindowTextA(deffile);

   return TRUE;
}
// ...
```

Листинг 11.30. Файл pr11View.h (объявление класса представления)

```
// ...
class Cpr11View : public CView
{
// ...
public:
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
public:
   afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
   CPoint begin,                    // Координаты начала линии
          end;                      // Координаты конца линии
public:
   afx_msg void OnMouseMove(UINT nFlags, CPoint point);
   CPoint oldmouse;                 // Последние координаты перемещения мыши
public:
   afx_msg void OnFileNew();
   CString file;                    // Имя последнего загруженного (или
                                    // сохраненного файла)
public:
   afx_msg void OnFileOpen();
public:
   afx_msg void OnFileSave();
public:
   afx_msg void OnFileSaveAs();
   int fmove;                   // Флаг для обработки двойного щелчка мыши
                                // при выборе файла
};
```

Листинг 11.31. Файл pr11View.cpp (определение класса представления)

```
// ...
BEGIN_MESSAGE_MAP(Cpr11View, CView)
   ON_WM_LBUTTONDOWN()
   ON_WM_LBUTTONUP()
   ON_WM_MOUSEMOVE()
   ON_COMMAND(ID_FILE_NEW, &Cpr11View::OnFileNew)
   ON_COMMAND(ID_FILE_OPEN, &Cpr11View::OnFileOpen)
   ON_COMMAND(ID_FILE_SAVE, &Cpr11View::OnFileSave)
   ON_COMMAND(ID_FILE_SAVE_AS, &Cpr11View::OnFileSaveAs)
END_MESSAGE_MAP()

// Cpr11View construction/destruction

Cpr11View::Cpr11View()
{
   // TODO: add construction code here
   fmove = 0;
}
// ...
BOOL Cpr11View::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Modify the Window class or styles here by modifying
   //  the CREATESTRUCT cs
   cs.lpszClass = AfxRegisterWndClass(NULL, LoadCursor(NULL, IDC_CROSS),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);
   return CView::PreCreateWindow(cs);
}

// Cpr11View drawing

void Cpr11View::OnDraw(CDC* pDC)
{
   Cpr11Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
```

```
    if(!pDoc)
        return;

    // TODO: add draw code for native data here

    // Закрыть контекст устройства и получить идентификатор метафайла
    HMETAFILE hmeta = pDoc->pmetaDC->Close();
    // Отобразить метафайл
    pDC->PlayMetaFile(hmeta);
    // Создать контекст устройства ("чистый" - для дальнейшей работы)
    pDoc->pmetaDC->Create();
    // Проиграть метафайл (чтобы контекст был не "чистый", а содержал
    // предыдущие данные)
    pDoc->pmetaDC->PlayMetaFile(hmeta);
    // Удалить идентификатор метафайла
    DeleteMetaFile(hmeta);
}
// ...
void Cpr11View::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    begin = point;
    oldmouse = point;

    CView::OnLButtonDown(nFlags, point);
}

void Cpr11View::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    end = point;

    // Рисование в метафайле
    Cpr11Doc* pDoc = GetDocument();
    pDoc->pmetaDC->MoveTo(begin.x, begin.y);
    pDoc->pmetaDC->LineTo(end.x, end.y);
```

```
    begin = end;

    CView::OnLButtonUp(nFlags, point);
}

void Cpr11View::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default

    int oldmode;                          // Старый бинарный растровый режим
    // Получить контекст устройства
    CClientDC *pDC = new CClientDC(this);
    Cpr11Doc *pDoc = this->GetDocument();

    // Если в момент перемещения нажата левая кнопка мыши
    if(nFlags && MK_LBUTTON)
    {
       if(fmove == 1)
       {
       static int i;
       if(i == 2)
       {
          fmove=0;
          i=0;
          }
          i++;
          CView::OnMouseMove(nFlags, point);
          return;
       }

       oldmode = pDC->GetROP2();     // Сохранить старый растровый режим
       pDC->SetROP2(R2_NOT);         // Установить новый растровый режим
       // "Стирание" старой линии
       pDC->MoveTo(begin.x, begin.y);
       pDC->LineTo(oldmouse.x, oldmouse.y);
```

```
        pDoc->pmetaDC->MoveTo( begin.x, begin.y);
        pDoc->pmetaDC->LineTo( oldmouse.x, oldmouse.y);
        // Рисование новой линии
        pDC->MoveTo(begin.x, begin.y);
        pDC->LineTo(point.x, point.y);
        pDoc->pmetaDC->MoveTo(begin.x, begin.y);
        pDoc->pmetaDC->LineTo(point.x, point.y);

        //oldmouse = point;
        begin = point;
        pDC->SetROP2(oldmode);      // Восстановить старый растровый режим
    }

    CView::OnMouseMove(nFlags, point);
}

void Cpr11View::OnFileNew()
{
    // TODO: Add your command handler code here
    Cpr11Doc* pDoc = GetDocument();
    file = "";       // Нет последнего загруженного или сохраненного файла
    // Заголовок окна - без имени файла (Untitled - pr11)
    theApp.m_pMainWnd->SetWindowTextA(theApp.deffile);
    // Закрыть старый метафайл (с данными)
    pDoc->pmetaDC->Close();
    // Создать новый метафайл (чистый)
    pDoc->pmetaDC->Create();
    // Перерисовать окно представления
    this->Invalidate();
}

void Cpr11View::OnFileOpen()
{
    // TODO: Add your command handler code here
    Cpr11Doc* pDoc = GetDocument();
```

```
    // Создать объект стандартного диалога выбора файлов для открытия
    CFileDialog fd(true);
    // Выбранный пользователем файл должен существовать на диске
    fd.m_ofn.Flags |=OFN_FILEMUSTEXIST;
    // Новый (не стандартный) заголовок окна диалога
    fd.m_ofn.lpstrTitle = "Open WMF file";
    // Фильтр - только файлы *.wmf
    fd.m_ofn.lpstrFilter = "Grafic files( *.wmf )\0*.wmf";
    // В окне диалога отразить текущий каталог
    fd.m_ofn.lpstrInitialDir = NULL;
    // Запустить диалог
    if(fd.DoModal()== IDCANCEL)    // Если пользователь отказался от выбора
       return;
    file = fd.m_ofn.lpstrFile;     // Получить полное (с путем) имя
                                   // выбранного файла
    // Получить короткое (без пути) имя файла (для заголовка окна)
    CString shortfile = fd.m_ofn.lpstrFileTitle;
    shortfile += " - ";
    shortfile += theApp.m_pszAppName;        // Добавить имя приложения
    // Установить заголовок окна ( имя_файла.wmf - pr11 )
    theApp.m_pMainWnd->SetWindowTextA(shortfile);

    // Получить дескриптор выбранного метафайла
    HMETAFILE hmeta = GetMetaFile(file);
    pDoc->pmetaDC->Close();                  // Закрыть старый метафайл
    pDoc->pmetaDC->Create();                 // Создать новый
    // Связать его с полученным дескриптором
    pDoc->pmetaDC->PlayMetaFile(hmeta);
    DeleteMetaFile(hmeta);                   // Удалить дескриптор
    fmove = 1;
    this->Invalidate();                  // Перерисовать окно представления
}

void Cpr11View::OnFileSave()
{
```

```
    // TODO: Add your command handler code here
    Cpr11Doc* pDoc = GetDocument();

    static CString oldfile = "?";  // Имя последнего сохраненного файла
    if(oldfile != file)            // Если открытый файл еще не сохраняли
    {
       CFileDialog fd(false, "wmf", "default");     // Диалог Save As
       fd.m_ofn.lpstrTitle = "Save WMF file";
       fd.m_ofn.lpstrFilter = "Grafic files( *.wmf )\0*.wmf";
       fd.m_ofn.lpstrInitialDir = NULL;
       if(fd.DoModal()== IDCANCEL)
          return;

       file = oldfile = fd.m_ofn.lpstrFile;     // Получить имя файла
       // Формирование заголовка окна
       CString shortfile = fd.m_ofn.lpstrFileTitle;
       shortfile += " - ";
       shortfile += theApp.m_pszAppName;
       theApp.m_pMainWnd->SetWindowTextA(shortfile);
    }

    HMETAFILE hmeta = pDoc->pmetaDC->Close();   // Закрыть метафайл
    // Скопировать метафайл в файл на диске
    hmeta = CopyMetaFile(hmeta, file);
    pDoc->pmetaDC->Create();                    // Создать новый метафайл
    // Связать его с дескриптором старого (чтобы новый не был чистым)
    pDoc->pmetaDC->PlayMetaFile(hmeta);
    DeleteMetaFile(hmeta);          // Удалить дескриптор старого метафайла
}

void Cpr11View::OnFileSaveAs()
{
    // TODO: Add your command handler code here
    file = "";
    OnFileSave();
}
```

Глава 12

Работа с графическими данными с использованием архива

Создадим новый проект, на примере которого рассмотрим возможности отображения графических данных в окне представления с помощью архива.

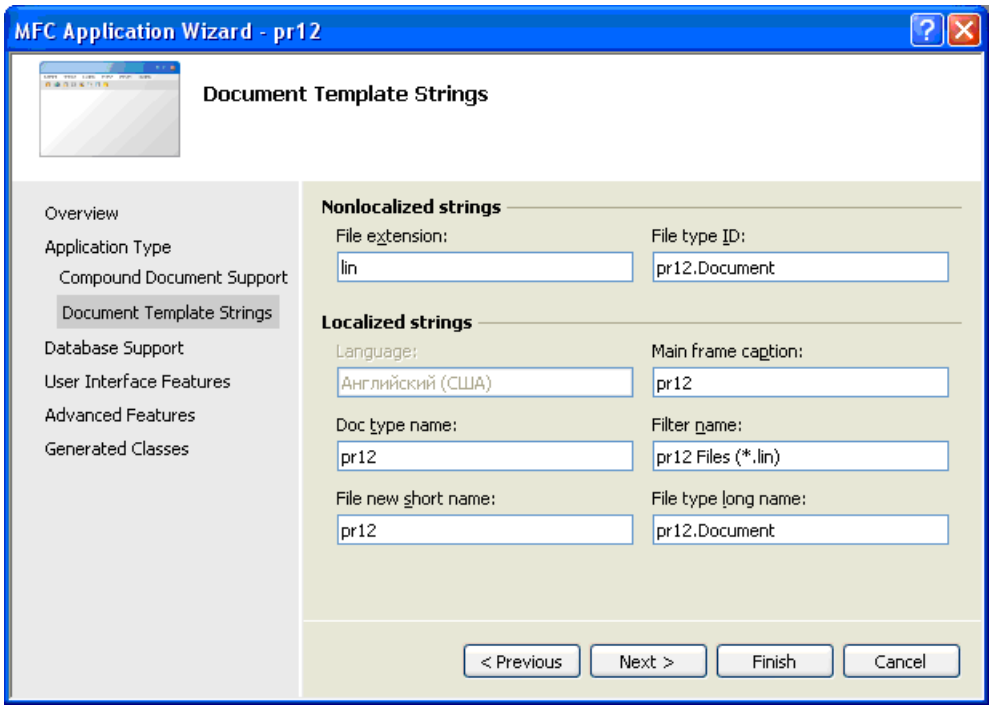


Рис. 12.1. Задание типа расширения файлов в настройках проекта

Создайте проект **pr12** аналогично проекту **pr11**, задав в свойствах проекта на вкладке **Document Template Strings** в поле **File extension** (Расширение файла) расширение **lin**, которое будет автоматически добавляться к именам файлов. Надо изменить опции относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ;
  - без Unicode;
- на вкладке **Advanced Features**:
  - без печати и предварительного просмотра;
- на вкладке **Document Template Strings**:
  - в поле **File extension** ввести **lin** (рис. 12.1).

12.1. Описание программы

12.1.1. Рисование графических изображений

Сделаем так, чтобы пользователь мог рисовать прямые линии в окне представления с помощью мыши (нажатие левой кнопки мыши — начало рисования линии, отпускание левой кнопки мыши — конец рисования). Для этого надо добавить в программу обработку сообщений о нажатии и отпускании левой кнопки мыши и о перемещении мыши.

Выполнить все действия как в *разд. 11.2.1*. Изменения в файлах приведены в листингах 12.1 и 12.2.

Листинг 12.1. Изменения в файле pr12View.h для рисования линий с помощью левой кнопки мыши

```
// ...
class Cpr12View : public CView
{
// ...
public:
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
```

```
public:
   afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
public:
   afx_msg void OnMouseMove(UINT nFlags, CPoint point);
   CPoint begin,          // Координаты начала линии
         end;             // Координаты конца линии
   CPoint oldmouse;       // Последние координаты перемещения мыши
};
```

Листинг 12.2. Изменения в файле pr12View.cpp для рисования линий с помощью левой кнопки мыши

```
// ...
BEGIN_MESSAGE_MAP(Cpr12View, CView)
   ON_WM_LBUTTONDOWN()
   ON_WM_LBUTTONUP()
   ON_WM_MOUSEMOVE()
END_MESSAGE_MAP()

BOOL Cpr12View::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Modify the Window class or styles here by modifying
   //  the CREATESTRUCT cs
   cs.lpszClass = AfxRegisterWndClass(NULL, LoadCursor(NULL, IDC_CROSS),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);

   return CView::PreCreateWindow(cs);
}
// ...
void Cpr12View::OnLButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   begin = point;
   oldmouse = point;
```

```
    CView::OnLButtonDown(nFlags, point);
}

void Cpr12View::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    end = point;
    // Получить контекст устройства
    CClientDC *pDC = new CClientDC(this);
    CDC *p = this->GetDC();
    // Нарисовать линию
    pDC->MoveTo(begin.x, begin.y);
    pDC->LineTo(end.x, end.y);

    begin = end;

    CView::OnLButtonUp(nFlags, point);
}

void Cpr12View::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    int oldmode;                    // Старый бинарный растровый режим
    // Получить контекст устройства
    CClientDC *pDC = new CClientDC(this);

    // Если в момент перемещения нажата левая кнопка мыши
    if(nFlags && MK_LBUTTON)
    {
        oldmode = pDC->GetROP2();   // Сохранить старый растровый режим
        pDC->SetROP2(R2_NOT);       // Установить новый растровый режим
        // "Стирание" старой линии
        pDC->MoveTo(begin.x, begin.y);
        pDC->LineTo(oldmouse.x, oldmouse.y);
```

```
        // Рисование новой линии
        pDC->MoveTo(begin.x, begin.y);
        pDC->LineTo(point.x, point.y);

        oldmouse = point;

        pDC->SetROP2(oldmode);        // Восстановить старый растровый режим
    }

    CView::OnMouseMove(nFlags, point);
}
```

12.1.2. Работа с архивом для чтения/записи данных на диск

Теперь добавим новый класс `CLine` (производный от `CObject`, для возможности работы с архивом и поддержки стандартных функций **New**, **Open**, **Save** и т. д.). В классе документа `Cpr12Doc` заведем массив линий `arline`. При рисовании очередной линии она будет заноситься в массив. Рисование линий теперь будет происходить в функции `Cpr12View::OnDraw()`. Для разрушения массива в класс документа `Cpr12Doc` надо добавить стандартную функцию удаления содержимого текущего документа `DeleteContents()` (рис. 12.2):

- в окне **Class View** вызвать контекстное меню для класса `Cpr12Doc` и выполнить команду **Properties**;
- в окне **Properties** выбрать вкладку **Overrides**;
- в списке функций найти `DeleteContents` и выбрать **<Add> DeleteContents**.

В файл определения класса документа добавить определение конструкторов и деструктора класса `CLine`. В класс линии (`CLine`) для поддержки работы с архивом добавить стандартную функцию сериализации *(см. разд. 10.1.3)* `Serialize()` (рис. 12.3):

- в окне **Class View** вызвать контекстное меню для класса `CLine` и выполнить команду **Properties**;
- в окне **Properties** выбрать вкладку **Overrides**;
- в списке функций найти `Serialize` и выбрать **<Add> Serialize**.

Изменения в файлах приведены в листингах 12.3 и 12.4.

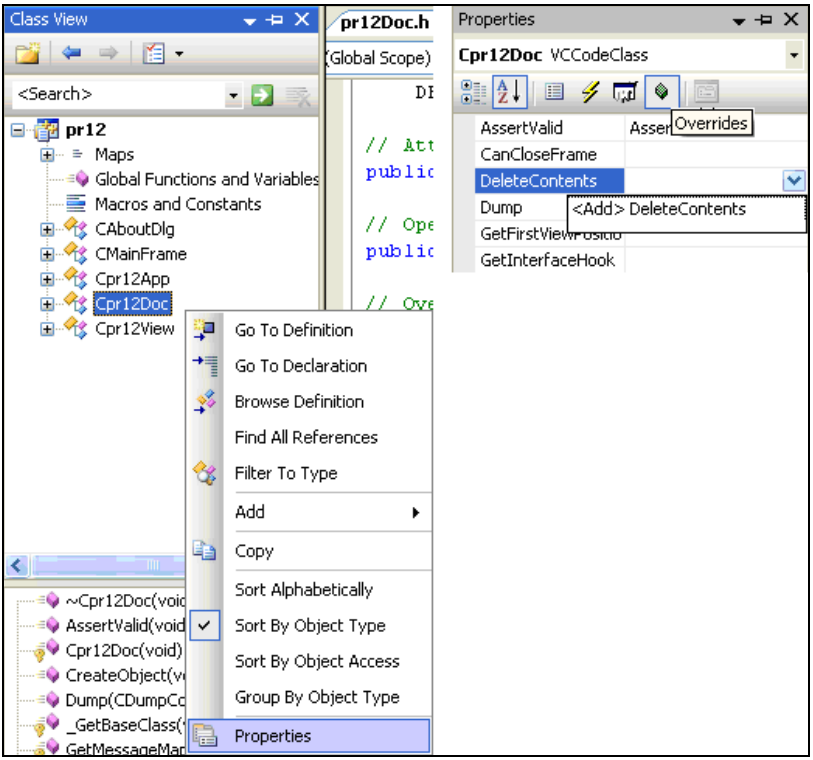


Рис. 12.2. Добавление функции удаления содержимого текущего документа

Листинг 12.3. Изменения в файле pr12Doc.h для обмена данными между документом и окном представления через архив

```
// pr12Doc.h : interface of the Cpr12Doc class
//
#pragma once

class CLine : public CObject
{
public:
   CPoint begin, end;                  // Координаты начала и конца линии

   DECLARE_SERIAL (CLine)              // Для работы с архивом
public:
   CLine(CPoint start, CPoint finish);
```

```
   virtual ~CLine();
   CLine(void);
public:
   virtual void Serialize(CArchive& ar);
};

class Cpr12Doc : public CDocument
{
// ...
public:
   CTypedPtrArray<CObArray, CLine*> arline;      // Массив линий

public:
   virtual void DeleteContents();
};
```

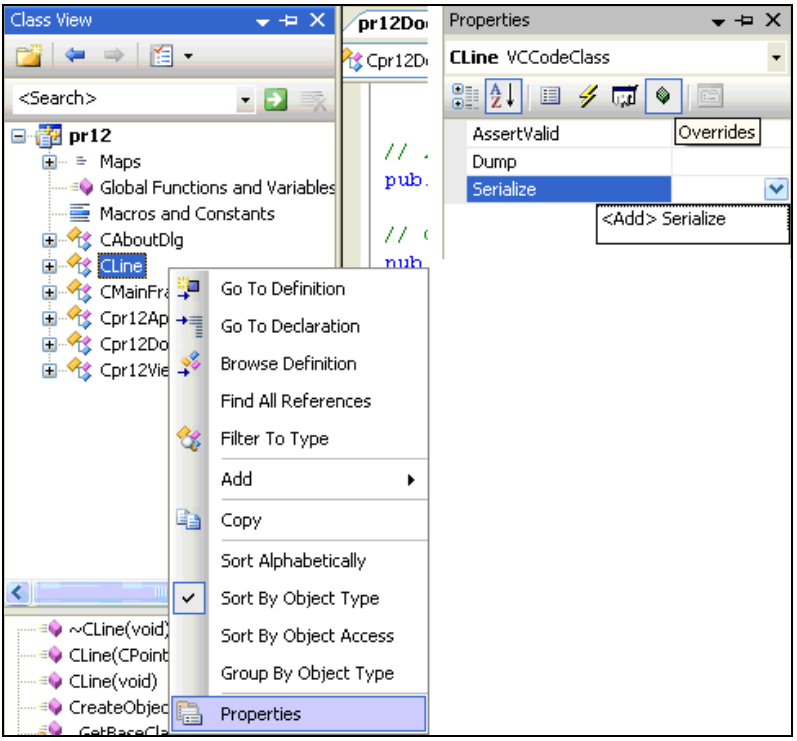


Рис. 12.3. Добавление в класс линии `CLine` функции сериализации

Листинг 12.4. Изменения в файле pr12Doc.cpp для обмена данными между документом и окном представления через архив

```
// ...
// Cpr12Doc serialization

void Cpr12Doc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {
      // TODO: add storing code here
      this->arline.Serialize(ar);
   }
   else
   {
      // TODO: add loading code here
      this->arline.Serialize(ar);
   }
}

// ...
// Cpr12Doc commands

void Cpr12Doc::DeleteContents()
{
   // TODO: Add your specialized code here and/or call the base class
   int sz = (int)this->arline.GetSize();
   for(int i = 0; i < sz; i++)
   {
      delete this->arline.GetAt(i);     // Удалить объект
   }
   this->arline.RemoveAll();            // Удалить все указатели на объект

   CDocument::DeleteContents();
}
```

```
IMPLEMENT_SERIAL(CLine, CObject, 1)        // Для работы с архивом
// Конструктор
CLine::CLine(CPoint start, CPoint finish):begin(start),end(finish)
{
}
CLine::~CLine()
{
}
CLine::CLine(void)
{
}
void CLine::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {   // storing code
      // Занесение данных в архив для записи в файл
      ar << begin.x << begin.y << end.x << end.y;
   }
   else
   {   // loading code
      // Получение данных из архива после чтения из файла
      ar >> begin.x >> begin.y >> end.x >> end.y;
   }
}
```

Для классов, производных от сериализуемого класса `CObject`, с целью обеспечения доступа к имени класса во время выполнения программы в объявление класса надо добавить макрос `DECLARE_SERIAL`, а в определение — `IMPLEMENT_SERIAL`:

```
DECLARE_SERIAL(
   class_name)                     // Имя класса
IMPLEMENT_SERIAL(
   class_name,                     // Имя класса
   base_class_name,                // Имя базового класса
   wSchema)                        // Номер версии программы
```

Номер версии программы (wSchema) сохраняется в записанном файле. Прочитатать такой файл может только программа, указавшая такой же номер. Нельзя задавать номер версии равным -1.

Библиотека MFC поддерживает две коллекции объектов CObject:

```
class CObArray : public CObject          // Массив указателей объектов
                                         // класса CObject
```

и

```
class CObList : public CObject           // Список указателей объектов
                                         // класса CObject
```

Для поддержки безопасности типов указателей используются классы шаблонов коллекций CTypedPtrArray (массивы), CTypedPtrList (списки) и CTypedPtrMap (изображения).

Шаблон коллекций массивов определен как:

```
template<class BASE_CLASS,     // Базовый класс (CObArray или CPtrArray)
         class TYPE>           // Тип указателя
class CTypedPtrArray : public BASE_CLASS
```

Тип указателя (class TYPE) может представлять собой:

1. Тип указателя на класс, производный от CObject, если базовым классом (BASE_CLASS) является CObArray.
2. Указатель на любой тип, если базовым классом (BASE_CLASS) является CPtrArray.

При создании нового документа (или при открытии документа) функция CWinApp::OnFileNew() или CWinApp::OnFileOpen() вызывает виртуальную функцию DeleteContents(), задача которой удалить содержимое текущего документа перед инициализацией нового.

Функция для удаления содержимого текущего документа (ее надо переопределять самому) выглядит следующим образом:

```
virtual void CDocument::DeleteContents();
```

Функция получения размера шаблонного массива определена как:

```
INT_PTR CObArray::GetSize() const;
```

Получение указателя на элемент шаблонного массива по его индексу выполняется функцией:

```
TYPE CTypedPtrArray::GetAt(        // Тип указателя на элемент массива
   INT_PTR nIndex) const;          // Индекс элемента в массиве
```

При уничтожении объекта (с помощью оператора `delete`) он прекращает свое существование и освобождает занимаемую им память. Поэтому после удаления объекта надо удалить все указатели на него для предотвращения их использования:

```
void CObArray::RemoveAll();
```

Конструктор класса `CLine` получает начальные (`start`) и конечные (`finish`) координаты создаваемой линии. Запись вида:

```
CLine::CLine(CPoint start, CPoint finish):begin(start),end(finish)
{
}
```

аналогична записи:

```
CLine::CLine(CPoint start, CPoint finish)
{
   begin  = start;
   end = finish;
}
```

Для переменных, которые представляют собой объекты определенного класса, вызывается функция-член `Serialize()` их класса, причем класс, объекты которого могут быть сериализированы, должен быть наследником класса `CObject`. Порядок записи/чтения данных из архива должен соответствовать друг другу.

Теперь добавим в класс представления необходимый код для создания отображения линий. Изменения в файлах приведены в листинге 12.5.

Листинг 12.5. Изменения в файле pr12View.cpp для отображения линий в окна представления

```
// ...
void Cpr12View::OnDraw(CDC* pDC)
{
   Cpr12Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if(!pDoc)
      return;

   // TODO: add draw code for native data here
   int sz = (int)pDoc->arline.GetSize(); // Получить размер массива линий
```

```
   for(int i = 0; i < sz; i++)
   {
      // Рисование текущей линии
      pDC->MoveTo(pDoc->arline.GetAt(i)->begin.x,
                  pDoc->arline.GetAt(i)->begin.y);
      pDC->LineTo(pDoc->arline.GetAt(i)->end.x,
                  pDoc->arline.GetAt(i)->end.y );
   }
}

void Cpr12View::OnLButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   end = point;
   // Получить контекст устройства
   //CClientDC *pDC = new CClientDC(this);
   //CDC *p = this->GetDC();
   // Нарисовать линию
   //pDC->MoveTo(begin.x, begin.y);
   //pDC->LineTo(end.x, end.y);

   Cpr12Doc* pDoc = GetDocument();
   CLine *pl = new CLine(begin, end);      // Создать новую линию
   pDoc->arline.Add(pl);                   // Добавить ее в массив
   pDoc->SetModifiedFlag();                // Установить флаг изменений

   begin = end;

   CView::OnLButtonUp(nFlags, point);
}
```

Добавление элемента в массив выполняется с помощью фукнции:

```
INT_PTR CTypedPtrArray::Add(          // Индекс добавленного элемента
   TYPE newElement);                  // Добавляемый элемент
```

При создании нового документа, при открытии документа или при закрытии приложения проверяется значение флага изменений. Этот флаг показывает, имеются ли в документе несохраненные данные, и, если да, выводится соответствующее предупреждение (рис. 12.4).

Функция установки флага изменений определена как:

```
virtual void CDocument::SetModifiedFlag(
   BOOL bModified = TRUE);        // TRUE — есть несохраненные данные,
                                  // FALSE — все данные сохранены
```

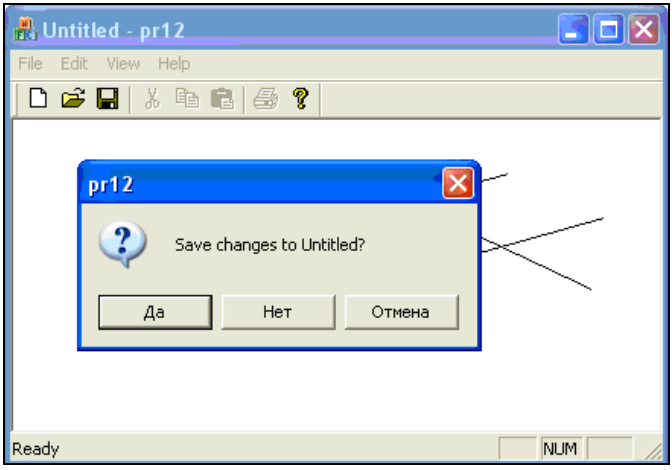
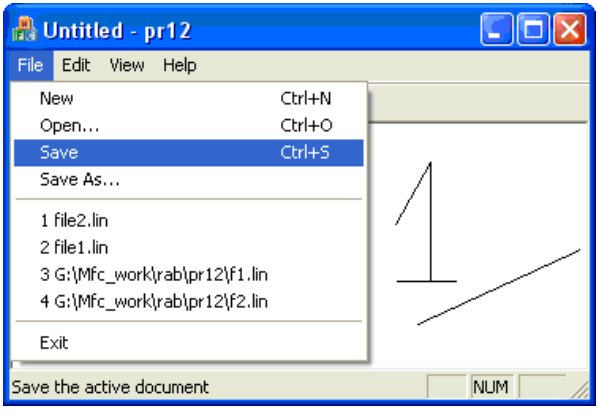
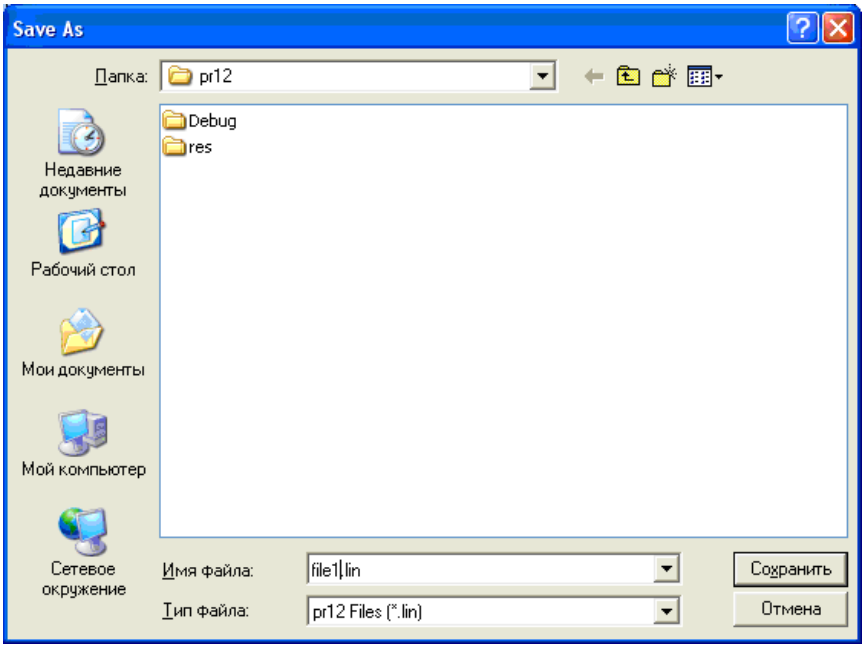


Рис. 12.4. Сообщение при попытке закрыть несохраненный документ

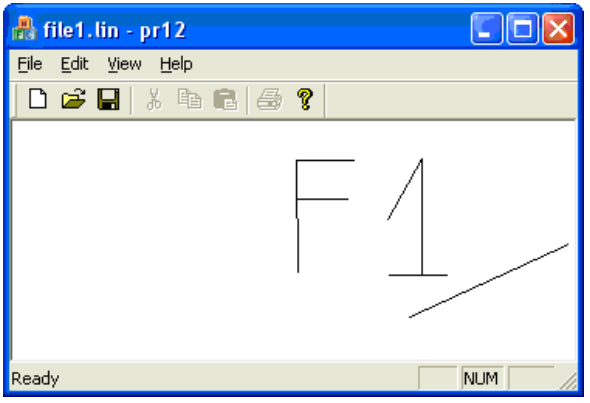


а

Рис. 12.5. Выбор сохранения нового документа (*а*)



б



в

Рис. 12.5. Указание имени файла для сохраняемого документа (*б*) и сохраненный в файле документ (*в*)

Получить текущее состояние флага изменений можно вызовом следующей функции (функция возвращает значение `TRUE`, если документ был изменен после его последнего сохранения, иначе — `FALSE`):

```
virtual BOOL void CDocument::IsModified();
```

Результат работы программы показан на рис. 12.5.

12.1.3. Дополнительные возможности работы с файлами

При работе с документом с использованием архива не надо дополнительно создавать обработку пунктов меню для открытия и записи файлов (как в *гл. 11*). Это делается стандартными функциями `CWinApp::OnFileNew()` и `CWinApp::OnFileOpen()`, заданными в карте сообщений приложения (см. листинг 10.1). Определения этих функций и функция записи файлов скрыты в библиотеке MFC. Но можно сделать и свою обработку для открытия/записи файлов (обычно это делается, если стандартная обработка не устраивает).

Добавим в проект новое меню **m** с тремя подменю (рис. 12.6):

- **openlin** — для своей обработки открытия графического файла;
- **savetxt** — для записи содержимого графического окна в текстовый файл. В файл будут записываться координаты линий текущего графического документа в формате `(Хначальное; Yначальное) - (Хконечное; Yконечное)`;
- **opentxt** — для открытия текстового файла с координатами в Блокноте Windows.

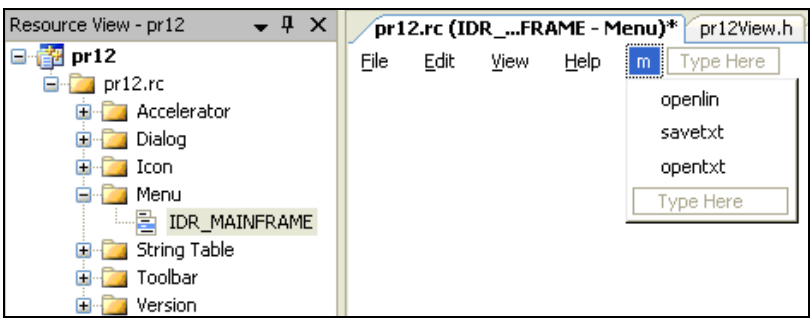
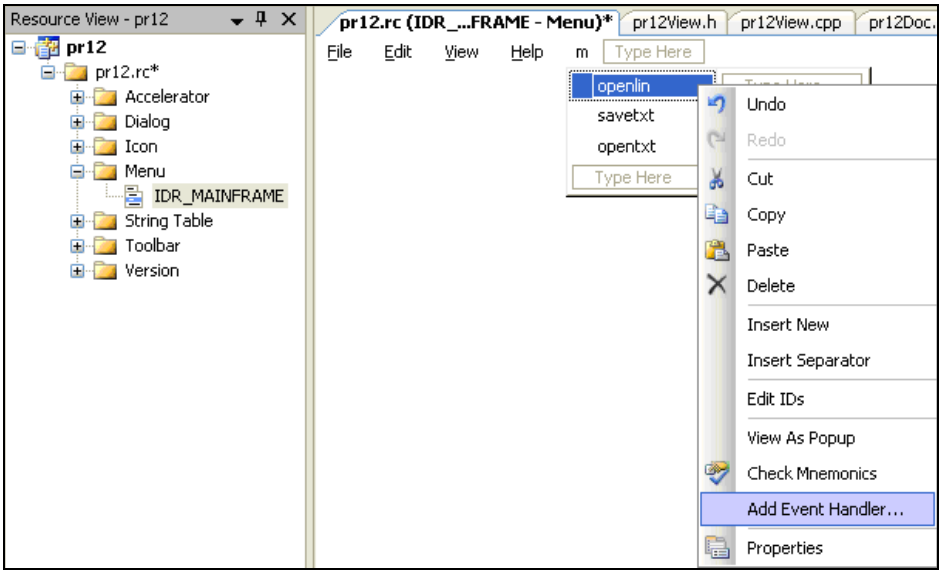


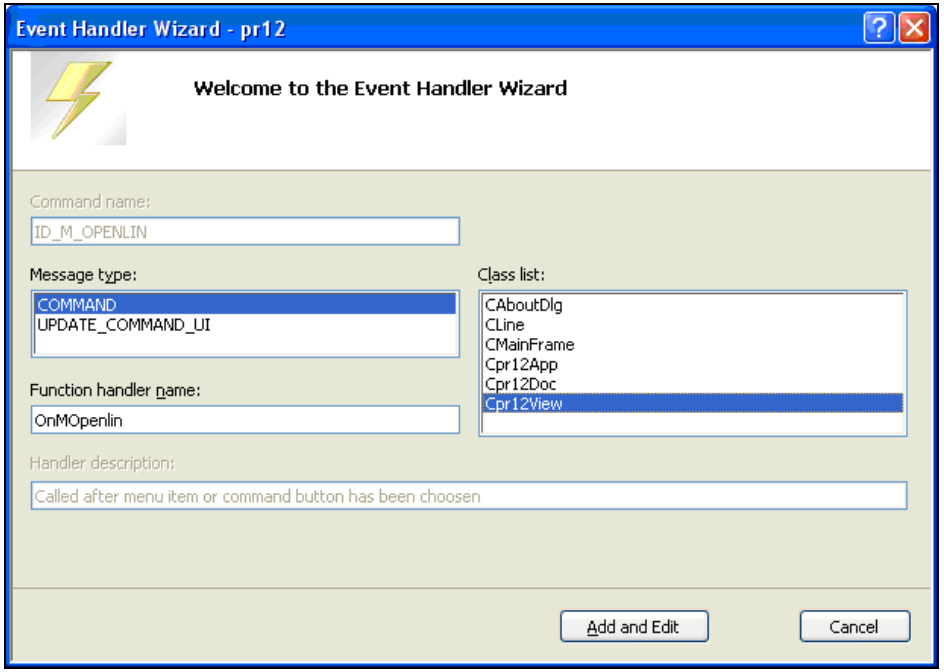
Рис. 12.6. Добавление новых меню для работы с файлами

Обработку этих меню сделаем в классе представления (рис. 12.7):

- для нужного пункта меню вызвать контекстное меню и выполнить команду **Add Event Handler**;
- в окне **Event Handler Wizard**, в поле **Class list** выбрать класс `Cpr12View`;
- нажать кнопку **Add and Edit**.



а



б

Рис. 12.7. Добавление обработки пункта меню (*а*)
и функции обработки пункта меню в класс окна представления (*б*)

Эти действия надо выполнить три раза — для каждого пункта добавленного меню.

Изменения в файлах приведены в листингах 12.6—12.8.

Листинг 12.6. Изменения в файле Resource.h для добавления меню открытия и записи файлов

```
// ...
#define ID_M_OPENLIN        32771
#define ID_M_SAVETXT        32772
#define ID_M_OPENTXT         32773
// ...
```

Листинг 12.7. Изменения в файле pr12View.h для добавления меню открытия и записи файлов

```
// ...
class Cpr12View : public CView
{
// ...
public:
   afx_msg void OnMOpenlin();
public:
   afx_msg void OnMSavetxt();
public:
   afx_msg void OnMOpentxt();
};
```

Листинг 12.8. Изменения в файле pr12View.cpp для добавления меню открытия и записи файлов

```
// ...
BEGIN_MESSAGE_MAP(Cpr12View, CView)
   // ...
   ON_COMMAND(ID_M_OPENLIN, &Cpr12View::OnMOpenlin)
   ON_COMMAND(ID_M_SAVETXT, &Cpr12View::OnMSavetxt)
```

```
   ON_COMMAND(ID_M_OPENTXT, &Cpr12View::OnMOpentxt)
END_MESSAGE_MAP()
// ...
void Cpr12View::OnMOpenlin()
{
   // TODO: Add your command handler code here
}
void Cpr12View::OnMSavetxt()
{
   // TODO: Add your command handler code here
}
void Cpr12View::OnMOpentxt()
{
   // TODO: Add your command handler code here
}
```

Своя обработка открытия графического файла

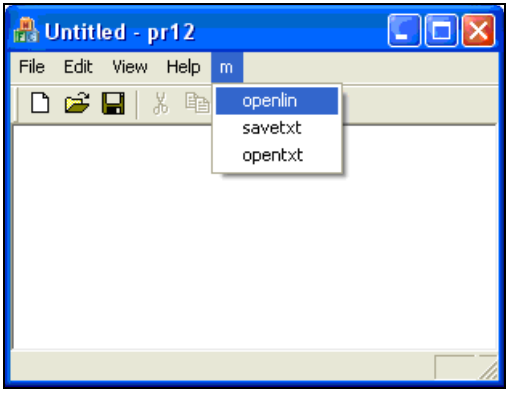
Изменения в файле приведены в листинге 12.9.

Листинг 12.9. Изменения в файле pr12View.cpp для обработки открытия графического файла

```
// ...
void Cpr12View::OnMOpenlin()
{
   // TODO: Add your command handler code here

   CString file;                  // Полное имя файла (с путем)
   CString shortfile;             // Короткое имя файла (без пути)
   CFileDialog fd(true);          // Диалог открытия файла
   // Выбранный пользователем файл должен существовать на диске
   fd.m_ofn.Flags |= OFN_FILEMUSTEXIST;
   // Новый (не стандартный) заголовок окна диалога
   fd.m_ofn.lpstrTitle = "Open LIN file";
```

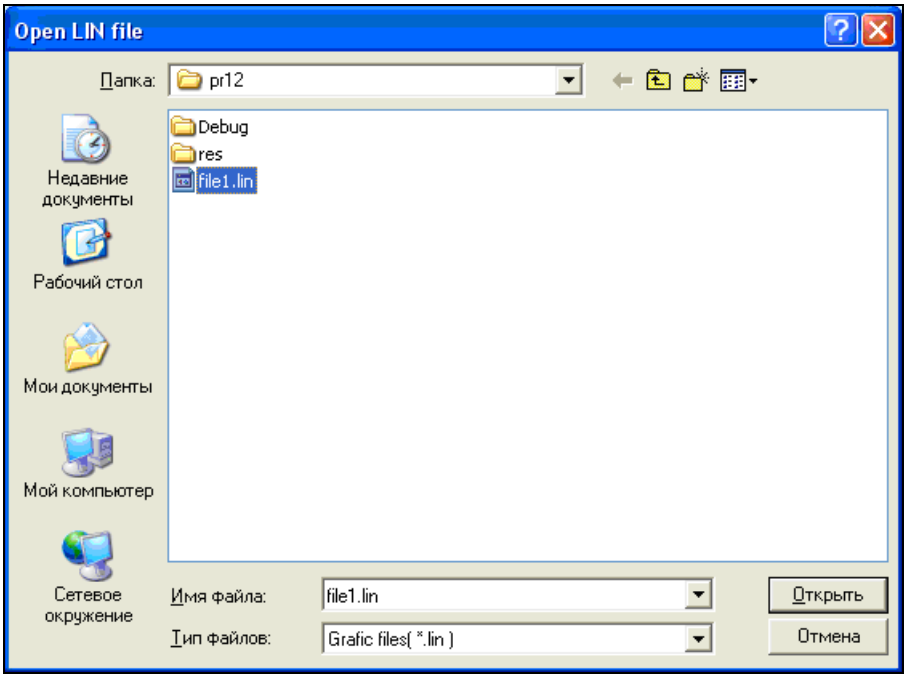
```
    // Фильтр - только файлы *.lin
    fd.m_ofn.lpstrFilter = "Grafic files( *.lin )\0*.lin";
    // В окне диалога отразить текущий каталог
    fd.m_ofn.lpstrInitialDir = NULL;
    // Запустить диалог
    if(fd.DoModal()== IDCANCEL)    // Если пользователь отказался от выбора
       return;

    file = fd.m_ofn.lpstrFile;      // Получить полное имя выбранного файла

    // Формирование заголовка окна ( имя_файла.lin - pr12 )
    shortfile = fd.m_ofn.lpstrFileTitle;    // Получить короткое имя файла
    shortfile += " - ";
    shortfile += theApp.m_pszAppName;       // Добавить имя приложения
    // Установить заголовок окна
    theApp.m_pMainWnd->SetWindowTextA(shortfile);
    // Открыть выбраннй файл в окне представления
    theApp.m_pDocManager->OpenDocumentFile(file);
}
```

Результат работы программы показан на рис. 12.8.

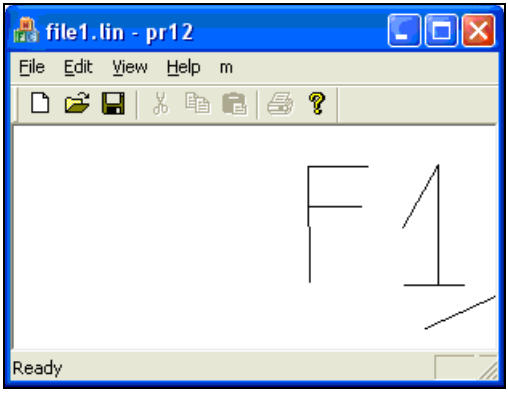


а

Рис. 12.8. Выбор меню
для загрузки графического файла (*а*)



б



в

Рис. 12.8. Выбор имени открываемого графического файла (*б*) и открытый графический файл (*в*)

Сохранение графических координат в текстовом файле

Изменения в файле приведены в листинге 12.10.

Листинг 12.10. Изменения в файле pr12View.cpp для сохранения координат графического рисунка в текстовом файле

```
// ...
void Cpr12View::OnMSavetxt()
{
   // TODO: Add your command handler code here

   CString file;                    // Полное имя файла
   CFileDialog fd(false, "txt");    // Диалог для записи файла
   // Новый (не стандартный) заголовок окна диалога
   fd.m_ofn.lpstrTitle = "Save TXT file";
   // Фильтр - только файлы *.txt
   fd.m_ofn.lpstrFilter = "Text files( *.txt)\0*.txt";
   // В окне диалога отразить текущий каталог
   fd.m_ofn.lpstrInitialDir = NULL;
   // Запустить диалог
   if(fd.DoModal()== IDCANCEL)    // Если пользователь отказался от выбора
      return;

   file = fd.m_ofn.lpstrFile;      // Получить полное имя выбранного файла
   CStdioFile f;                   // Класс для работы с файлами
   // Открыть выбранный файл для записи
   f.Open(file, CFile::modeCreate | CFile::modeWrite | CFile::typeText);
   CString s;                      // Строка с координатами текущей линии
   Cpr12Doc* pDoc = GetDocument();     // Получить указатель на документ

   int sz = (int)pDoc->arline.GetSize(); // Получить размер массива линий
   for(int i = 0; i < sz; i++)
   {
      // Сформировать строку с координатами
      s.Format("( %d; %d ) - ( %d; %d )\n",
               pDoc->arline.GetAt(i)->begin.x,
               pDoc->arline.GetAt(i)->begin.y,
               pDoc->arline.GetAt(i)->end.x,
```

```
                pDoc->arline.GetAt(i)->end.y);
      // Записать ее в файл
      f.WriteString(s);
   }
   f.Close();                              // Закрыть файл
}
```

Классы для работы с файлами перечислены далее.

Работа с двоичными файлами:

```
class CFile : public CObject
```

Работа с двоичными или с *текстовыми* файлами:

```
class CStdioFile : public CFile
```

Конструктор класса `CStdioFile` определен как:

```
CStdioFile::CStdioFile();
```

или

```
CStdioFile::CStdioFile(FILE* pOpenStream);    // Указатель на файл,
                                              // используемый функцией Си
                                              // fopen()
```

или

```
CStdioFile::CStdioFile(LPCTSTR lpszFileName, // Указатель на имя файла
   UINT nOpenFlags);                          // Режим открытия (может
                                              // комбинироваться с '|')
```

Режимы открытия файла (`nOpenFlags`) могут быть такие:

- `CFile::modeCreate` — создание нового файла. Если данный файл существует, он усекается до нулевого размера;
- `CFile::modeNoTruncate` — значение может быть объединено с `modeCreate`. Если файл уже существует, то он не усекается до 0;
- `CFile::modeRead` — файл открывается только для чтения;
- `CFile::modeReadWrite` — файл открывается для чтения и записи;
- `CFile::modeWrite` — файл открывается только для записи;
- `CFile::modeNoInherit` — предотвращает наследование файла дочерними процессами;

- `CFile::shareDenyNone` — открывает файл, не запрещая другим процессам чтение или запись в него;
- `CFile::shareDenyRead` — открывает файл и запрещает другим процессам доступ к нему по чтению;
- `CFile::shareDenyWrite` — открывает файл и запрещает другим процессам доступ к нему для записи;
- `CFile::shareExclusive` — открывает файл в монопольном режиме (запрещает другим процессам доступ к нему для чтения и записи);
- `CFile::typeText` — открывает файл в текстовом режиме;
- `CFile::typeBinary` — открывает файл в двоичном режиме.

Открытие файла выполняется с помощью функции:

```
virtual BOOL CFile::Open(           // 0 - ошибка
   LPCTSTR lpszFileName,            // Указатель на имя файла
   UINT nOpenFlags,                 // Режим открытия (см. выше)
   CFileException* pError = NULL);  // Исключение при ошибке открытия
```

Для того чтобы не просто проверить, открылся файл или нет, а узнать точную причину возникновения ошибки открытия, надо воспользоваться параметром `pError`. Например, следующая обработка попытки открытия несуществующего файла в режиме чтения:

```
CStdioFile ff;
CFileException ex;
if(!ff.Open( "notf.txt", CFile::modeRead, &ex))
{
   ex.ReportError();
}
```

приведет к автоматической выдаче окна сообщения с текстом "notf.txt was not found".

Класс обработки исключений при работе с файлами:

```
class CFileException : public CException
```

Базовый класс для обработки исключений MFC:

```
class AFX_NOVTABLE CException : public CObject
```

Функция выдачи сообщения текста ошибки в окне сообщения определена как:

```
virtual int CException::ReportError( // 0 - ошибка (не хватило памяти
                                     // для выдачи окна сообщения)
```

```
   UINT nType = MB_OK,              // Стиль окна сообщения
                                    // (см. разд. 1.2.2)
   UINT nMessageID = 0);            // Идентификатор ресурса сообщения
                                    // в таблице строк (String Table)
```

Варианты возвращаемого значения зависят от того, какую кнопку нажал пользователь в окне сообщения (см. `AfxMessageBox()` в *разд. 1.6.1*).

Классы для обработки исключений могут быть следующими:

- `CSimpleException` — базовый класс для обработки критических исключений ресурсов MFC;
- `CInvalidArgException` — неверные аргументы;
- `CMemoryException` — работа с памятью;
- `CNotSupportedException` — неподдерживаемые действия;
- `CArchiveException` — работа с архивом;
- `CFileException` — работа с файлами;
- `CResourceException` — не создан или не обнаружен ресурс окна;
- `COleException` — OLE исключения;
- `CDBException` — работа с базой данных;
- `CUserException` — не найден ресурс;
- `CInternetException` — работа с Интернетом.

Запись строки в файл выполняется функцией:

```
virtual void CStdioFile::WriteString(
   LPCTSTR lpsz);                    // Указатель на записываемую строку
```

Чтение строки из файла выполняется функцией:

```
virtual BOOL CStdioFile::ReadString(
   CString& rString);               // Читаемая строка
```

или

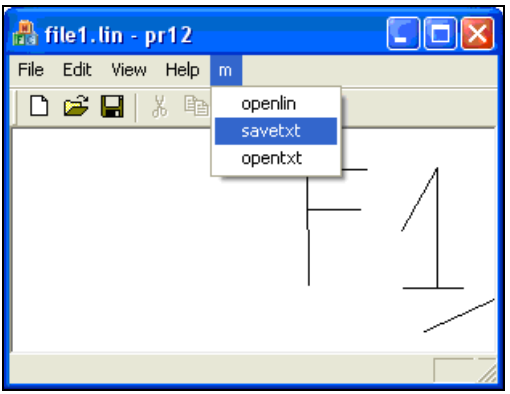
```
virtual LPTSTR CStdioFile:: ReadString(
   LPTSTR lpsz,                     // Указатель на читаемую строку
   UINT nMax);                      // Количество читаемых байтов
```

Буфер строки должен завершаться нуль-символом (`'\0'`). Чтение строки прекращается, как только встретится символ новой строки, независимо от значения `nMax`. Если будет прочитано символов меньше, чем `nMax-1`, то в буфер автоматически будет добавлен символ новой строки.

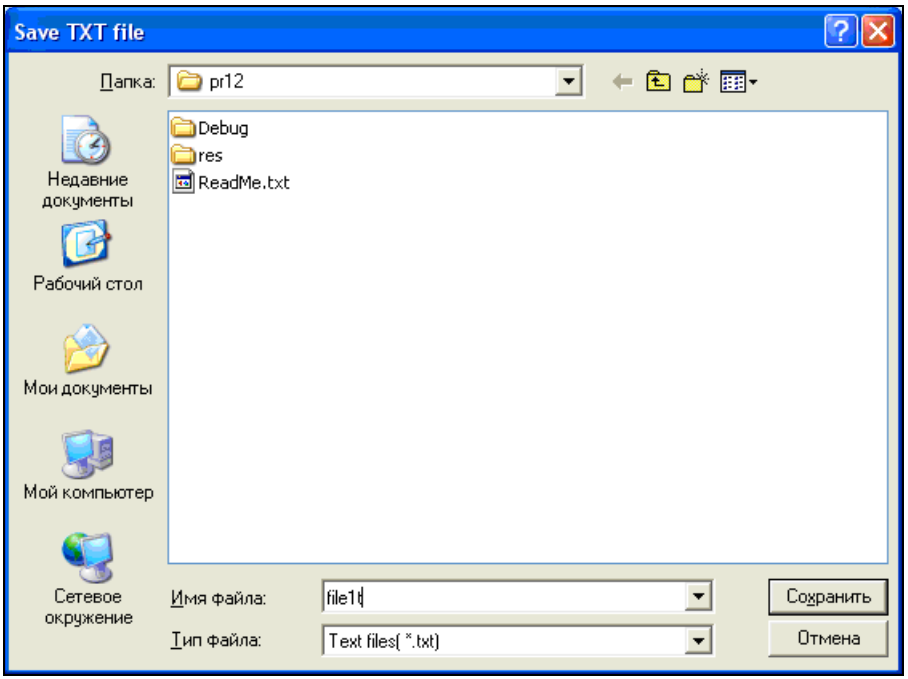
Закрытие файла выполняется с помощью функции:

```
virtual void CFile::Close();
```

Результат работы программы показан на рис. 12.9.



а



б

Рис. 12.9. Сохранение графического изображения в текстовом файле (*а*) и диалоговое окно выбора имени текстового файла (*б*)

Загрузка текстового файла

Так как наше приложение однодокументное (рассчитанное на работу с графическими файлами), то для загрузки текстового файла можно воспользоваться вызовом стандартного блокнота и загрузить в него выбранный текстовый файл. Изменения в файле приведены в листинге 12.11.

Листинг 12.11. Изменения в файле pr12View.cpp для открытия файла с помощью Блокнота

```
// ...
void Cpr12View::OnMOpentxt()
{
   // TODO: Add your command handler code here
   CString file,                    // Имя файла без пути
           dir;                     // Путь к файлу
   CFileDialog fd(true);            // Диалог открытия файла
   // Выбранный пользователем файл должен существовать на диске
   fd.m_ofn.Flags |=OFN_FILEMUSTEXIST;
   // Новый (не стандартный) заголовок окна диалога
   fd.m_ofn.lpstrTitle = "Open TXT file";
   // Фильтр - только файлы *.txt
   fd.m_ofn.lpstrFilter = "Text files( *.txt)\0*.txt";
   // В окне диалога отразить текущий каталог
   fd.m_ofn.lpstrInitialDir = NULL;
   // Запустить диалог
   if(fd.DoModal()== IDCANCEL)    // Если пользователь отказался от выбора
      return;

   dir = fd.m_ofn.lpstrFile;     // Получить полное имя файла (с путем)
   int i;
   int sz = dir.GetLength();     // Длина полного имени файла
   char ch;
   for(i=sz-1; i >= 0; i--)      // Просмотр полного имени файла с конца
   {
      ch = dir.GetAt(i);
```

```
      // Нашли первый с конца символ '\', который отделяет имя файла от
      // пути
      if(ch  == '\\')
         break;
   }
   dir.Delete(i+1, i);    // Удалили все после '\' (остался только путь)
   file = fd.GetFileName();      // Получили просто имя выбранного файла
   // Вызвали стандартную программу для работы с текстовыми файлами
   /// (блокнот)
   ShellExecute(theApp.m_pMainWnd->operator HWND(), "edit", file, 0, dir,
                SW_SHOWNORMAL);
}
```

Получение символа из строки выполняется функцией:

```
XCHAR CSimpleStringT::GetAt(        // Символ
   int iChar) const;                // Индекс символа
```

Удаление символов из строки выполняется функцией:

```
int CStringT::Delete(      // Длина измененной строки
   int iIndex,             // Индекс, с которого будут удаляться символы
   int nCount = 1);        // Количество удаляемых символов
```

Функция, выполняющая действия с определенным файлом, определена как:

```
HINSTANCE ShellExecute(     // Результат > 32, если все хорошо
                            // <= 32, при ошибке
   HWND hwnd,               // Дескриптор родительского окна
   LPCTSTR lpOperation,     // Указатель на строку с действием
   LPCTSTR lpFile,          // Имя файла
   LPCTSTR lpParameters,    // Дополнительные параметры для выполняемых
                            // файлов (ключи)
   LPCTSTR lpDirectory,     // Каталог с файлом
   INT nShowCmd);           // Режим отображения
```

Виды действий с файлом (`lpOperation`) могут быть такими:

- `edit` — запускает редактор и открывает файл для редактирования;
- `explore` — открывает и показывает папку с файлом;

- `find` — запускает поиск, начиная с определенного директория;
- `open` — открывает файл. Файл может быть выполняемым, документом или папкой;
- `print` — печатает файл (файл должен быть документом).

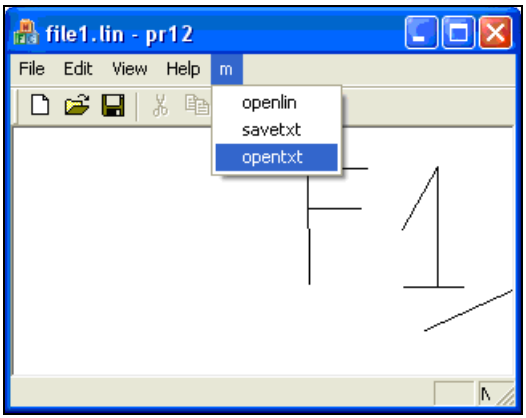
Режимы отображения (`nShowCmd`) могут принимать следующие значения (описание *см. разд. 1.6.1*):

- `SW_HIDE`;
- `SW_MAXIMIZE`;
- `SW_MINIMIZE`;
- `SW_RESTORE`;
- `SW_SHOW`;
- `SW_SHOWDEFAULT`;
- `SW_SHOWMAXIMIZED`;
- `SW_SHOWMINIMIZED`;
- `SW_SHOWMINNOACTIVE`;
- `SW_SHOWNA`;
- `SW_SHOWNOACTIVATE`;
- `SW_SHOWNORMAL`.

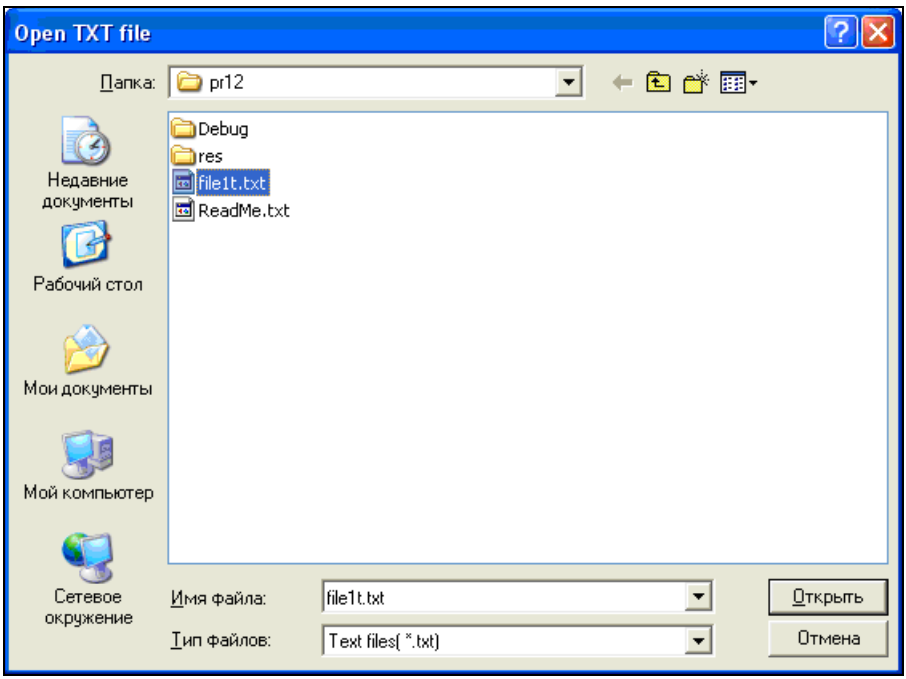
Значения, возвращаемые функцией при ошибке, могут быть такие:

- `0` — выход за пределы памяти или ресурсов системы;
- `ERROR_FILE_NOT_FOUND` (2L) — файл не найден;
- `ERROR_PATH_NOT_FOUND` (3L) — путь не найден;
- `ERROR_BAD_FORMAT` (11L) — выполняемый файл не может быть запущен;
- `SE_ERR_ACCESSDENIED` (5) — у операционной системы нет доступа к файлу;
- `SE_ERR_ASSOCINCOMPLETE` (27) — файловая ассоциация имени неполная или неправильная;
- `SE_ERR_DDEBUSY` (30) — не смог пройти динамический обмен данными DDE (Dynamic Data Exchange);
- `SE_ERR_DDEFAIL` (29) — обмен DDE произошел с ошибкой;

- `SE_ERR_DDETIMEOUT` (28) — обмен DDE не завершен из-за потери синхронизации;
- `SE_ERR_DLLNOTFOUND` (32) — не найдена динамическая библиотека;

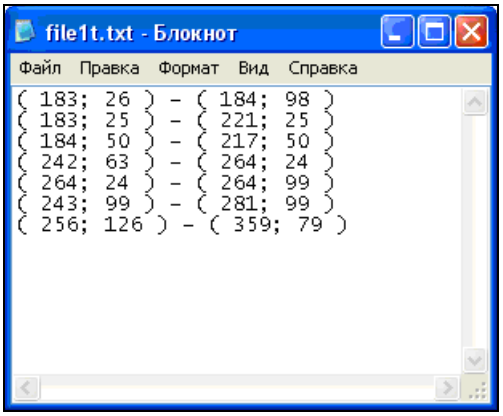


а



б

Рис. 12.10. Открытие текстового файла (*а*), выбор имени открываемого текстового файла (*б*)



в

Рис. 12.10. Выбранный текстовый файл открыт в Блокноте (*в*)

- `SE_ERR_FNF` (2) — файл не был найден;
- `SE_ERR_NOASSOC` (31) — нет приложения, которое связано с файлом с таким расширением;
- `SE_ERR_OOM` (8) — недостаточно памяти;
- `SE_ERR_PNF` (3) — путь не был найден.

Результаты работы программы показаны на рис. 12.10.

12.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 12.12. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define ID_M_OPENLIN                    32771
#define ID_M_SAVETXT                    32772
#define ID_M_OPENTXT                    32773
// ...
```

Листинг 12.13. Файл pr12Doc.h (объявление класса документа)

```
// pr12Doc.h : interface of the Cpr12Doc class
//
#pragma once

class CLine : public CObject
{
public:
   CPoint begin, end;              // Координаты начала и конца линии

   DECLARE_SERIAL (CLine)          // Для работы с архивом
public:
   CLine(CPoint start, CPoint finish);
   virtual ~CLine();
   CLine(void);
public:
   virtual void Serialize(CArchive& ar);
};

class Cpr12Doc : public CDocument
{
// ...
public:
   CTypedPtrArray<CObArray, CLine*> arline;       // Массив линий
public:
   virtual void DeleteContents();
};
```

Листинг 12.14. Файл pr12Doc.cpp (определение класса документа)

```
// ...
void Cpr12Doc::Serialize(CArchive& ar)
{
   if(ar.IsStoring())
```

```
    {
        // TODO: add storing code here
        this->arline.Serialize(ar);
    }
    else
    {
        // TODO: add loading code here
        this->arline.Serialize(ar);
    }
}
// ...
void Cpr12Doc::DeleteContents()
{
    // TODO: Add your specialized code here and/or call the base class
    int sz = (int)this->arline.GetSize();
    for(int i = 0; i < sz; i++)
    {
        delete this->arline.GetAt(i);     // Удалить объект
    }
    this->arline.RemoveAll();             // Удалить все указатели на объект

    CDocument::DeleteContents();
}

IMPLEMENT_SERIAL(CLine, CObject, 1)     // Для работы с архивом
// Конструктор
CLine::CLine(CPoint start, CPoint finish):begin(start),end(finish)
{
}
CLine::~CLine()
{
}
CLine::CLine(void)
{
}
```

```
void CLine::Serialize(CArchive& ar)
{
   if(ar.IsStoring())
   {  // storing code
      // Занесение данных в архив для записи в файл
      ar << begin.x << begin.y << end.x << end.y;
   }
   else
   {  // loading code
      // Получение данных из архива после чтения из файла
      ar >> begin.x >> begin.y >> end.x >> end.y;
   }
}
```

Листинг 12.15. Файл pr12View.h (объявление класса представления)

```
// ...
class Cpr12View : public CView
{
// ...
public:
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
public:
   afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
public:
   afx_msg void OnMouseMove(UINT nFlags, CPoint point);
   CPoint   begin,           // Координаты начала линии
            end;             // Координаты конца линии
   CPoint   oldmouse;        // Последние координаты перемещения мыши

public:
   afx_msg void OnMOpenlin();
public:
   afx_msg void OnMSavetxt();
```

```
public:
   afx_msg void OnMOpentxt();
};
```

Листинг 12.16. Файл pr12View.cpp (определение класса представления)

```
// ...
BEGIN_MESSAGE_MAP(Cpr12View, CView)
   ON_WM_LBUTTONDOWN()
   ON_WM_LBUTTONUP()
   ON_WM_MOUSEMOVE()
   ON_COMMAND(ID_M_OPENLIN, &Cpr12View::OnMOpenlin)
   ON_COMMAND(ID_M_SAVETXT, &Cpr12View::OnMSavetxt)
   ON_COMMAND(ID_M_OPENTXT, &Cpr12View::OnMOpentxt)
END_MESSAGE_MAP()
// ...
BOOL Cpr12View::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Modify the Window class or styles here by modifying
   //  the CREATESTRUCT cs
   cs.lpszClass = AfxRegisterWndClass(NULL, LoadCursor(NULL, IDC_CROSS ),
                        reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), NULL);

   return CView::PreCreateWindow(cs);
}
// ...
void Cpr12View::OnDraw(CDC* pDC)
{
   Cpr12Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if(!pDoc)
      return;

   // TODO: add draw code for native data here
   int sz = (int)pDoc->arline.GetSize(); // Получить размер массива линий
```

```
   for(int i = 0; i < sz; i++)
   {
      // Рисование текущей линии
      pDC->MoveTo(pDoc->arline.GetAt(i)->begin.x,
                  pDoc->arline.GetAt(i)->begin.y);
      pDC->LineTo(pDoc->arline.GetAt(i)->end.x,
                  pDoc->arline.GetAt(i)->end.y);
   }
}
// ...
void Cpr12View::OnLButtonDown(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   begin = point;
   oldmouse = point;

   CView::OnLButtonDown(nFlags, point);
}
void Cpr12View::OnLButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   end = point;

   Cpr12Doc* pDoc = GetDocument();
   CLine *pl = new CLine(begin, end);      // Создать новую линию
   pDoc->arline.Add(pl);                   // Добавить ее в массив
   pDoc->SetModifiedFlag();                // Установить флаг изменений

   begin = end;

   CView::OnLButtonUp(nFlags, point);
}
void Cpr12View::OnMouseMove(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
```

```
   int oldmode;                    // Старый бинарный растровый режим
   //  Получить контекст устройства
   CClientDC *pDC = new CClientDC(this);

   // Если в момент перемещения нажата левая кнопка мыши
   if(nFlags && MK_LBUTTON)
   {
      oldmode = pDC->GetROP2();    // Сохранить старый растровый режим
      pDC->SetROP2(R2_NOT);        // Установить новый растровый режим
      // "Стирание" старой линии
      pDC->MoveTo(begin.x, begin.y);
      pDC->LineTo(oldmouse.x, oldmouse.y);
      // Рисование новой линии
      pDC->MoveTo(begin.x, begin.y);
      pDC->LineTo(point.x, point.y);

      oldmouse = point;

      pDC->SetROP2(oldmode);       // Восстановить старый растровый режим
   }

   CView::OnMouseMove(nFlags, point);
}
void Cpr12View::OnMOpenlin()
{
   // TODO: Add your command handler code here
   CString file;                // Полное имя файла (с путем)
   CString shortfile;           // Короткое имя файла (без пути)
   CFileDialog fd(true);        // Диалог открытия файла
   // Выбранный пользователем файл должен существовать на диске
   fd.m_ofn.Flags |=OFN_FILEMUSTEXIST;
   // Новый (не стандартный) заголовой окна диалога
   fd.m_ofn.lpstrTitle = "Open LIN file";
   // Фильтр - только файлы *.lin
   fd.m_ofn.lpstrFilter = "Grafic files( *.lin )\0*.lin";
```

```
    // В окне диалога отразить текущий каталог
    fd.m_ofn.lpstrInitialDir = NULL;
    // Запустить диалог
    if(fd.DoModal()== IDCANCEL)  // Если пользователь отказался от выбора
       return;

    file = fd.m_ofn.lpstrFile;   // Получить полное имя выбранного файла
    // Формирование заголовка окна ( имя_файла.lin - pr12 )
    shortfile = fd.m_ofn.lpstrFileTitle;   // Получить короткое имя файла
    shortfile += " - ";
    shortfile += theApp.m_pszAppName;      // Добавить имя приложения
    // Установить заголовок окна
    theApp.m_pMainWnd->SetWindowTextA(shortfile);
    // Открыть выбраннй файл в окне представления
    theApp.m_pDocManager->OpenDocumentFile(file);
}
void Cpr12View::OnMSavetxt()
{
    // TODO: Add your command handler code here
    CString file;                          // Полное имя файла
    CFileDialog fd(false, "txt");          // Диалог записи файла
    // Новый (не стандартный) заголовой окна диалога
    fd.m_ofn.lpstrTitle = "Save TXT file";
    // Фильтр - только файлы *.txt
    fd.m_ofn.lpstrFilter = "Text files( *.txt)\0*.txt";
    // В окне диалога отразить текущий каталог
    fd.m_ofn.lpstrInitialDir = NULL;
    // Запустить диалог
    if(fd.DoModal()== IDCANCEL)   // Если пользователь отказался от выбора
       return;

    file = fd.m_ofn.lpstrFile;     // Получить полное имя выбранного файла

    CStdioFile f;                  // Класс для работы с файлами
```

```
   // Открыть выбранный файл для записи
   f.Open(file, CFile::modeCreate | CFile::modeWrite | CFile::typeText);
   CString s;                          // Строка с координатами текущей линии
   Cpr12Doc* pDoc = GetDocument(); // Получить указатель на документ

   int sz = (int)pDoc->arline.GetSize(); // Получить размер массива линий
   for(int i = 0; i < sz; i++)
   {
      // Сформировать строку с координатами
      s.Format("( %d; %d ) - ( %d; %d )\n",
               pDoc->arline.GetAt(i)->begin.x,
               pDoc->arline.GetAt(i)->begin.y,
               pDoc->arline.GetAt(i)->end.x,
               pDoc->arline.GetAt(i)->end.y);
      // Записать ее в файл
      f.WriteString(s);
   }
   f.Close();                                  // Закрыть файл
}
void Cpr12View::OnMOpentxt()
{
   // TODO: Add your command handler code here
   CString file,                       // Имя файла без пути
           dir;                        // Путь к файлу
   CFileDialog fd(true);               // Диалог открытия файла
   // Выбранный пользователем файл должен существовать на диске
   fd.m_ofn.Flags |=OFN_FILEMUSTEXIST;
   // Новый (не стандартный) заголовок окна диалога
   fd.m_ofn.lpstrTitle = "Open TXT file";
   // Фильтр - только файлы *.txt
   fd.m_ofn.lpstrFilter = "Text files( *.txt)\0*.txt";
   // В окне диалога отразить текущий каталог
   fd.m_ofn.lpstrInitialDir = NULL;
```

```
    // Запустить диалог
    if(fd.DoModal()== IDCANCEL)    // Если пользователь отказался от выбора
       return;
    dir = fd.m_ofn.lpstrFile;      // Получить полное имя файла (с путем)
    int i;
    int sz = dir.GetLength();      // Длина полного имени файла
    char ch;
    for(i=sz-1; i >= 0; i--)       // Просмотр полного имени файла с конца
    {
       ch = dir.GetAt(i);
       // Нашли первый с конца символ '\', который отделяет имя файла от
       // пути
       if(ch  == '\\')
          break;
    }
    dir.Delete(i+1, i);      // Удалили все после '\' (остался только путь)
    file = fd.GetFileName();  // Получили просто имя выбранного файла
    // Вызвали стандартную программу для работы с текстовыми файлами
    // (блокнот)
    ShellExecute(theApp.m_pMainWnd->operator HWND(), "edit", file, 0, dir,
                 SW_SHOWNORMAL);
}
```

ГЛАВА 13

Возможные виды окна представления

Создадим новый проект, на примере которого рассмотрим различные виды окна представления.

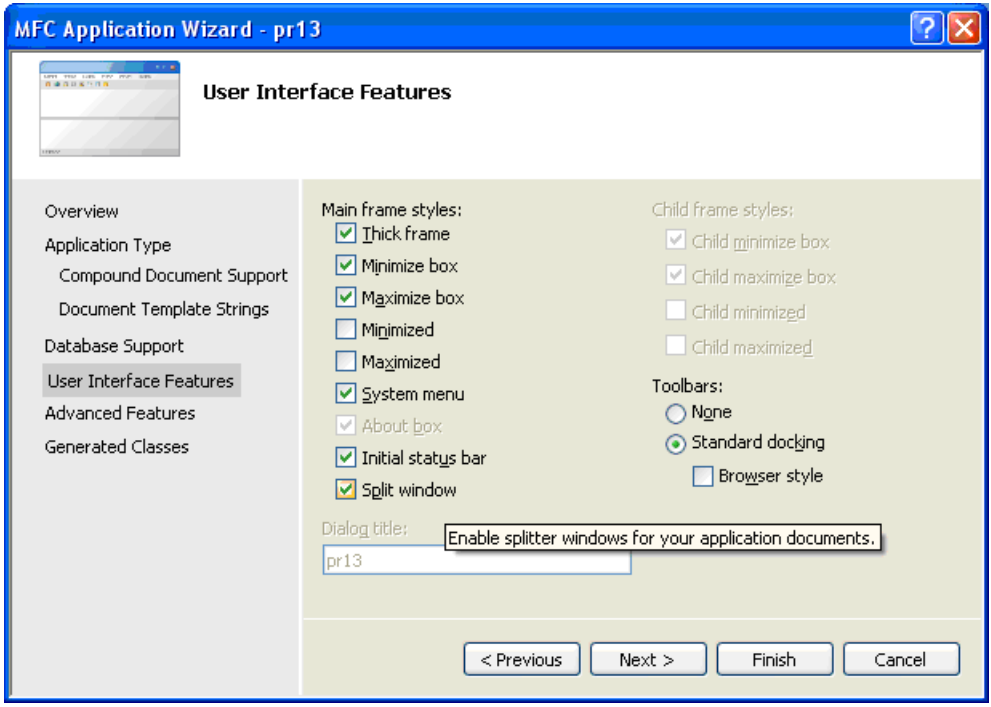


Рис. 13.1. Добавление опции разделения окна представления

Создайте проект **pr13** аналогично проекту **pr10**, *установив флажок* **Split window** (Разделение окон) на вкладке **User Interface Features** (рис. 13.1). Надо изменить следующие опции относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ;
  - без Unicode;
- на вкладке **User Interface Features**:
  - установить флажок разделения окон **Split window**;
- на вкладке **Advanced Features**:
  - без печати и предварительного просмотра.

13.1. Описание программы

13.1.1. Разделение окна представления

При таком создании проекта окно представления может быть разделено на 4 части. В листинг программы автоматически добавится код, приведенный в листингах 13.1 и 13.2.

Листинг 13.1. Изменения в файле MainFrm.h для автоматического разделения окна представления

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:
   CSplitterWnd m_wndSplitter;
public:
   virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs,
                               CCreateContext* pContext);
// ...
};
```

Листинг 13.2. Изменения в файле MainFrm.cpp для автоматического разделения окна представления

```
// ...
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT /*lpcs*/,
                                CCreateContext* pContext)
{
   return m_wndSplitter.Create(this,
            2, 2,            // TODO: adjust the number of rows, columns
            CSize(10, 10),  // TODO: adjust the minimum pane size
            pContext);
}
// ...
```

Для поддержки разделенных окон существует класс:

```
class CSplitterWnd : public CWnd
```

Каждая область представляет собой отдельное окно, которое управляется объектом `CSplitterWnd` и обычно является объектом, производным от `CView` (или от его производного класса). При этом класс, на основе которого создается область, должен использовать макросы `DECLARE_DYNCREATE` и `IMPLEMENT_DYNCREATE`.

Функция создания клиентских областей (вызывается из `CFrameWnd::OnCreate()`):

```
virtual BOOL CFrameWnd::OnCreateClient(        // 0 - ошибка
   LPCREATESTRUCT lpcs,          // Указатель на структуру CREATESTRUCT
                                 // (см. разд. 1.6.2)
   CCreateContext* pContext);    // Указатель на структуру CCreateContext
```

При создании фреймового окна и окон представлений, связанных с документами, используется структура `CCreateContext`.

При создании окна величины этой структуры обеспечивают доступ и связь компонентов. Структура `CCreateContext` содержит указатели на документ, на фреймовое окно, окно представления, шаблон документа и указатель на объект `CRuntimeClass`.

Основные элементы структуры `CCreateContext` следующие:

- `m_pNewViewClass` — указатель на объект класса `CRuntimeClass`, содержащий объект класса представления;

- `m_pCurrentDoc` — указатель на объект класса текущего документа, который надо связать с новым объектом класса представления;
- `m_pNewDocTemplate` — указатель на объект класса шаблона документа для связывания с новым окном фрейма мультидокументального (MDI) приложения;
- `m_pLastView` — указатель на следующее представление в списке представлений;
- `m_pCurrentFrame` — указатель на текущее фреймовое окно (для MDI-приложения).

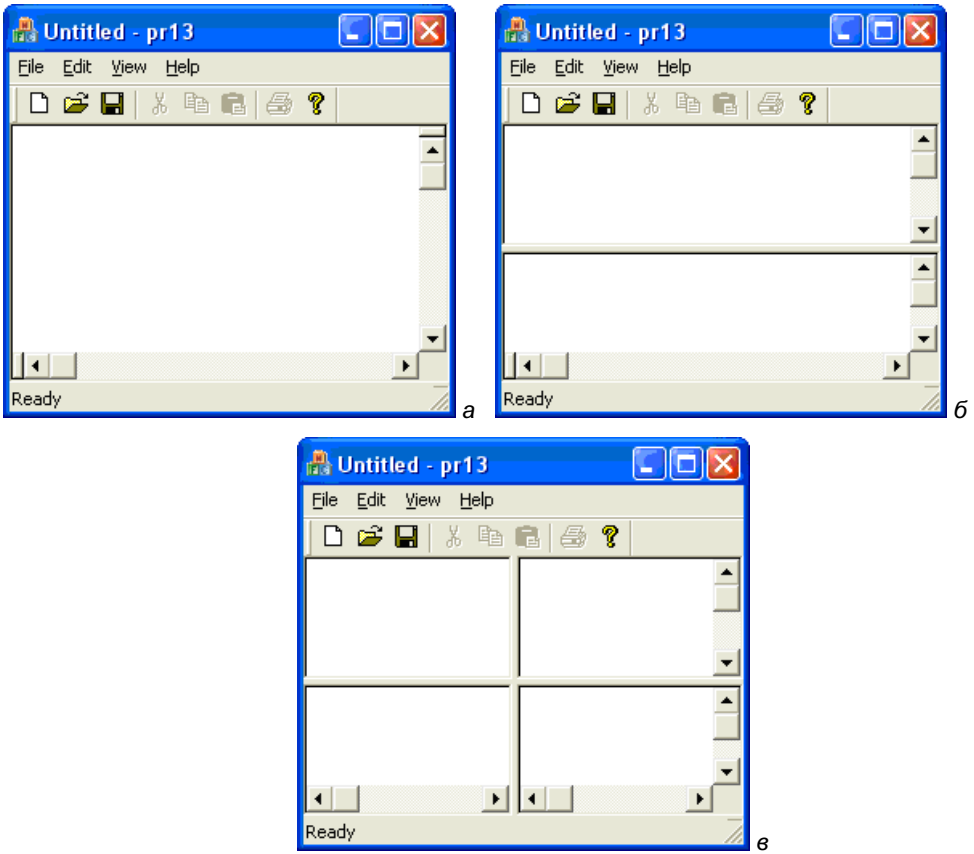


Рис. 13.2. Разделение окна представления: вид окна при запуске программы (*а*), при перемещении дополнительного движка на вертикальной (*б*) и горизонтальной (*б*) полосах прокрутки

Функция создания окна и присоединения его к объекту определена как:

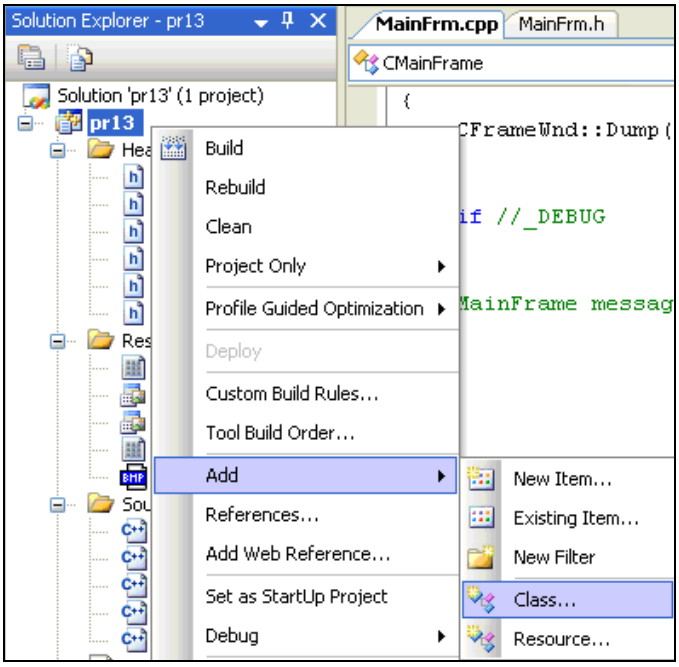
```
virtual BOOL CSplitterWnd::Create(  // 0 - ошибка
   CWnd* pParentWnd,                // Указатель на родительское окно
   int nMaxRows,                    // Максимальное число строк
                                    // (горизонтальных областей)
   int nMaxCols,                    // Максимальное число столбцов
                                   // (вертикальных областей)
   SIZE sizeMin,                   // Минимальная высота строки или
                                   // ширина столбца, при которой область
                                   // будет отображаться на экране
   CCreateContext* pContext,       // Указатель на структуру
                                   // CCreateContext
   DWORD dwStyle = WS_CHILD |      // Стиль окна (см. разд. 1.6.1)
           WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
           SPLS_DYNAMIC_SPLIT,     // Динамически разделяемое окно
   UINT nID = AFX_IDW_PANE_FIRST); // Идентификатор дочернего окна
```

При запуске программы вначале каждой линейки прокрутки есть дополнительный движок, "подцепив" который мышью можно менять размеры областей. Результаты работы программы показаны на рис. 13.2.

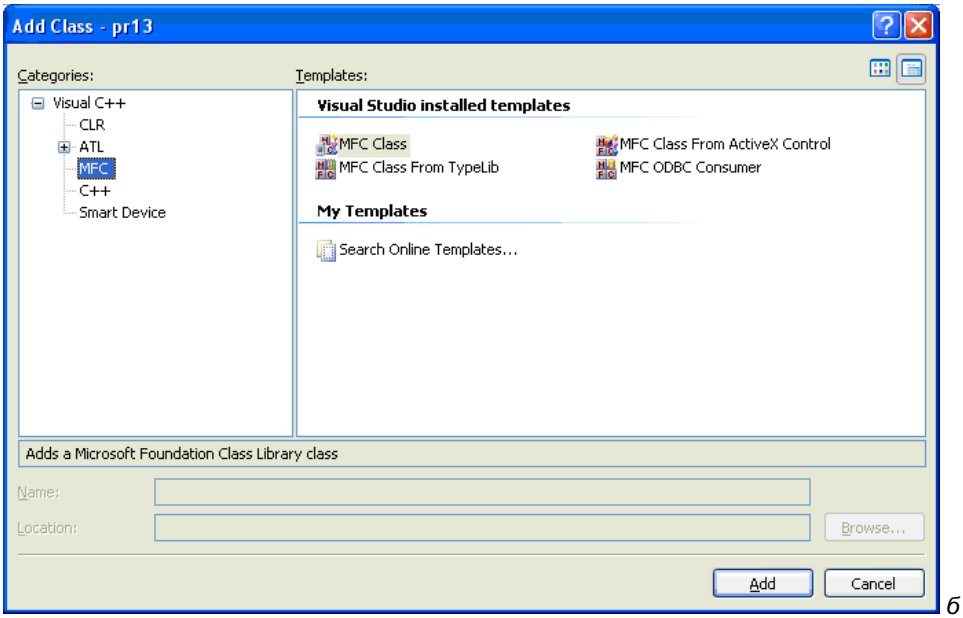
13.1.2. Добавление своих областей

Сделаем свое разделения окна представления так, чтобы в верхней половине было обычное окно представления (`Cpr13View`), где по нажатию правой кнопки мыши (в месте нажатия) появлялся бы символ `'X'`, а в нижней половине было окно редактирования, где отображаются координаты символа `'X'` в верхнем окне. Для этого надо добавить класс `CEditView2`, производный от `CEditView` (рис. 13.3):

- в окне **Solution Explorer** вызвать контекстное меню для проекта **pr13** и выполнить команду **Add | Class**;
- в окне **Add Class**:
  - выбрать в списке **Categories** узел **MFC**;
  - в списке **Templates** выбрать **MFC Class**;
  - нажать кнопку **Add**;

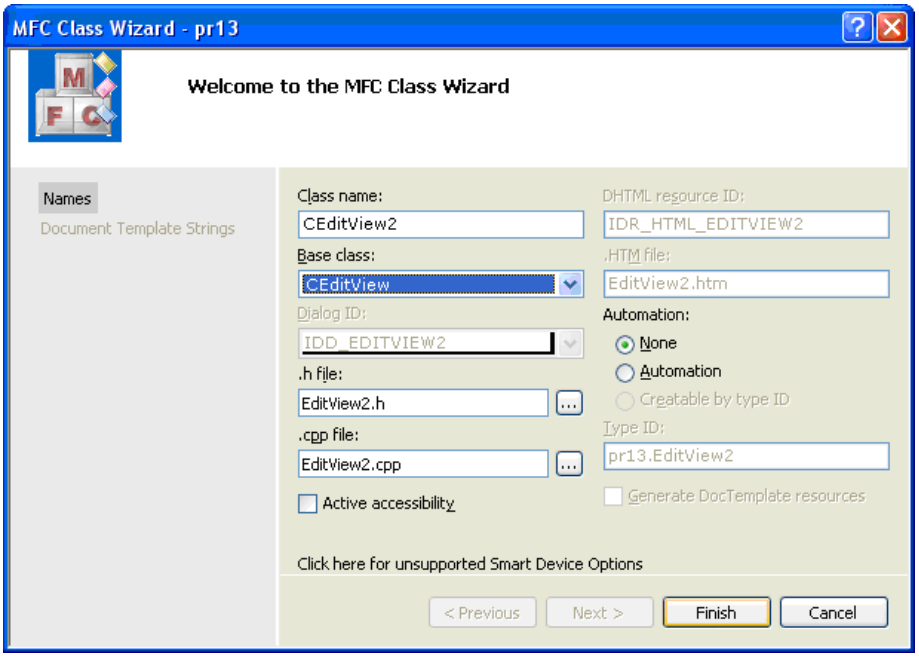


а



б

Рис. 13.3. Добавление нового класса в проект (*а*), выбор типа добавляемого класса (*б*)



в

Рис. 13.3. Задание названия класса
и типа базового класса (*в*)

- в окне **MFC Class Wizard**:
  - ввести в поле **Class name**: `CEditView2`;
  - в списке классов **Base class** выбрать **CEditView**;
  - нажать кнопку **Finish**.

Изменения в программе приведены в листингах 13.3 и 13.4.

Листинг 13.3. Файл EditView2.h (объявление класса нижнего текстового окна)

```
#pragma once

// CEditView2 view

class CEditView2 : public CEditView
{
   DECLARE_DYNCREATE(CEditView2)
```

```
protected:
    CEditView2();   // protected constructor used by dynamic creation
    virtual ~CEditView2();

public:
#ifdef _DEBUG
    virtual void AssertValid() const;
#ifndef _WIN32_WCE
    virtual void Dump(CDumpContext& dc) const;
#endif
#endif

protected:
    DECLARE_MESSAGE_MAP()
};
```

Листинг 13.4. Файл EditView2.cpp (объявление класса нижнего текстового окна)

```
// EditView2.cpp : implementation file
//
#include "stdafx.h"
#include "pr13.h"
#include "EditView2.h"

// CEditView2

IMPLEMENT_DYNCREATE(CEditView2, CEditView)

CEditView2::CEditView2()
{
}

CEditView2::~CEditView2()
{
}
```

```
BEGIN_MESSAGE_MAP(CEditView2, CEditView)
END_MESSAGE_MAP()

// CEditView2 diagnostics

#ifdef _DEBUG
void CEditView2::AssertValid() const
{
   CEditView::AssertValid();
}

#ifndef _WIN32_WCE
void CEditView2::Dump(CDumpContext& dc) const
{
   CEditView::Dump(dc);
}
#endif
#endif //_DEBUG

// CEditView2 message handlers
```

Создадим в окне представления две горизонтальные области. Изменения в файлах приведены в листингах 13.5 и 13.6.

Листинг 13.5. Изменения в файле MainFrm.h для разделения окна представления на две области

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:
   //CSplitterWnd m_wndSplitter;
public:
   CSplitterWnd m_wndSplitter;
   // ...
};
```

Листинг 13.6. Изменения в файле MainFrm.cpp для разделения окна представления на две области

```
// ...
#include "EditView2.h"
// ...
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                                CCreateContext* pContext)
{
   //return m_wndSplitter.Create(this,
   //      2, 2,       // TODO: adjust the number of rows, columns
   //      CSize(10, 10),   // TODO: adjust the minimum pane size
   //      pContext);

   // Создание статического разделенного окна из двух горизонтальных
   // областей (2 строки, 1 колонка)
   if(!this->m_wndSplitter.CreateStatic(this, 2,1))
      return FALSE;
   // Создание первой (верхней) области
   if(!this->m_wndSplitter.CreateView(0, 0, pContext->m_pNewViewClass,
                                      CSize(lpcs->cx, 200),pContext))
      return FALSE;
   // Создание второй (нижней) области
   if(!this->m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CEditView2),
                                      CSize(lpcs->cx, 200), pContext))
      return FALSE;

   // Сделать активной вторую область
   this->SetActiveView((CView *)m_wndSplitter.GetPane(1,0));

   return TRUE;
}
```

Создание статического разделенного окна, в котором пользователь может только изменять размер областей, но не их количество и порядок, выполняется функцией:

```
virtual BOOL CSplitterWnd::CreateStatic(   // 0 - ошибка
   CWnd* pParentWnd,                       // Указатель на родительское
                                           // окно
   int nRows,                              // Количество строк
   int nCols,                              // Количество столбцов
   DWORD dwStyle = WS_CHILD | WS_VISIBLE,  // Стиль окна
   UINT nID = AFX_IDW_PANE_FIRST);         // Идентификатор дочернего
                                           // окна
```

В отличие от функции `CSplitterWnd::Create()`, при создании статически разделяемого окна надо самостоятельно создавать все его области.

Создание области представления выполняется с помощью функции:

```
virtual BOOL CSplitterWnd::CreateView(                  // 0 - ошибка
   int row,                      // Строка, с которой будет размещаться
                                 // область
   int col,                      // Столбец, с которого будет
                                 // размещаться область
   CRuntimeClass* pViewClass,    // Указатель на структуру CRuntimeClass,
                                 // определяющую класс представления
                                 // области
   SIZE sizeInit,                // Начальный размер нового представления
   CCreateContext* pContext);    // Указатель на структуру CCreateContext
```

Сделать активным заданное окно представления можно функцией:

```
void CFrameWnd::SetActiveView(
   CView* pViewNew,          // Указатель на задаваемое окно представления
   BOOL bNotify = TRUE);     // Должно ли окно быть немедленно извещено
                             // об активизации (TRUE - да)
```

Получить указатель на окно, связанное с областью, можно функцией:

```
CWnd* CSplitterWnd::GetPane( // Указатель на окно, связанное с областью
   int row,                   // Индекс (с нуля) строки области
   int col) const;            // Индекс столбца области
```

Результат работы программы показан на рис. 13.4.

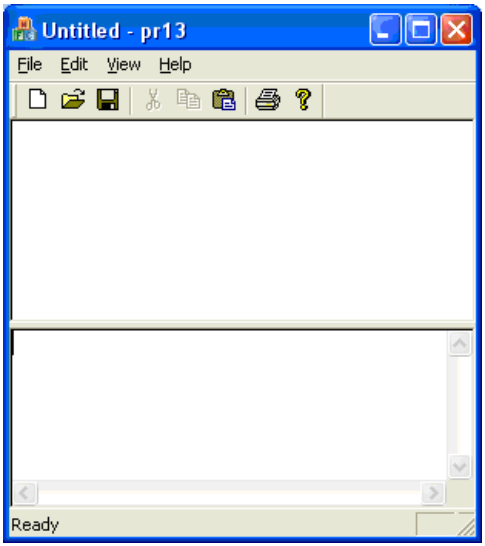


Рис. 13.4. Разделение окна представления на две части

Можно сделать так, чтобы нижнее окно (`CEditView2`) было только для чтения. Для этого надо в класс добавить функцию инициализации `OnInitialUpdate()` (рис. 13.5):

- в окне **Class View** вызвать контекстное меню для класса `CEditView2` и выполнить команду **Properties**;
- в окне **Properties** выбрать вкладку **Overrides**;
- в списке функций найти `OnInitialUpdate` и выбрать **<Add> OnInitialUpdate**.

Изменения в файлах приведены в листингах 13.7 и 13.8.

Листинг 13.7. Изменения в файле EditView2.h для инициализации нижней половины окна представления

```
// ...
class CEditView2 : public CEditView
{
// ...
public:
   virtual void OnInitialUpdate();
};
```

Листинг 13.8. Изменения в файле EditView2.cpp для инициализации нижней половины окна представления

```
// ...
void CEditView2::OnInitialUpdate()
{
   CEditView::OnInitialUpdate();

   // TODO: Add your specialized code here and/or call the base class
}
```

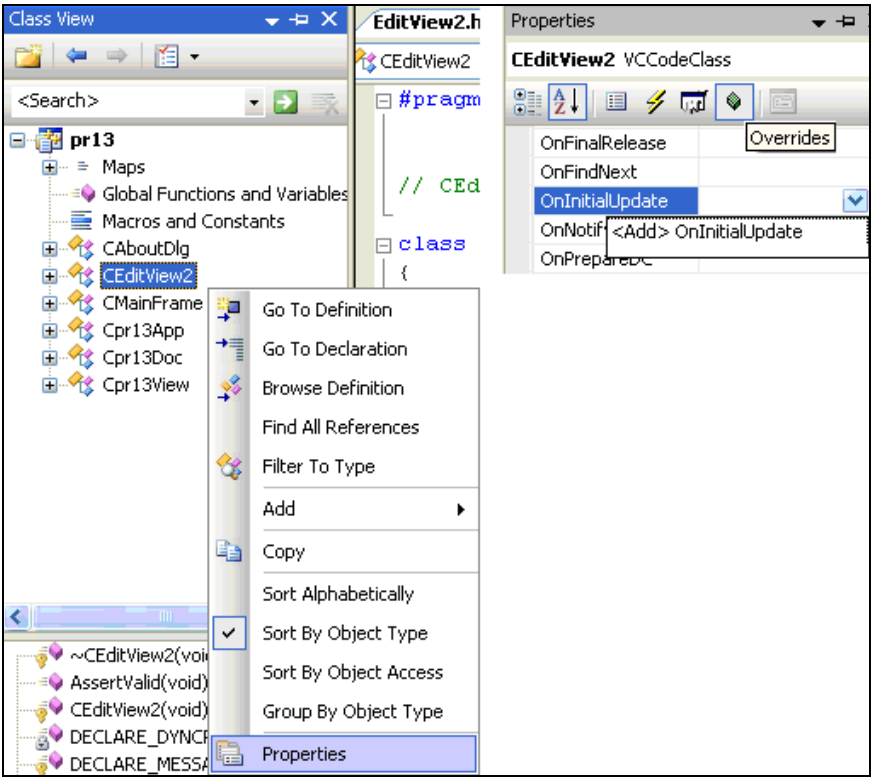


Рис. 13.5. Добавление функции инициализации окна редактирования

Эта функция вызывается приложением при первом связывании объекта представления с объектом документа (до вывода этого объекта на экран):

```
virtual void CView::OnInitialUpdate();
```

Теперь в эту функцию добавим код, делающий окно редактирования доступным только для чтения (такие окна по умолчанию отображаются не белым цветом, а серым). Изменения в файле приведены в листинге 13.9.

Листинг 13.9. Изменения в файле EditView2.cpp для установки свойств окна представления

```
// ...
void CEditView2::OnInitialUpdate()
{
   CEditView::OnInitialUpdate();

   // TODO: Add your specialized code here and/or call the base class
   this->GetEditCtrl().SetReadOnly();        // Окно только для чтения
}
```

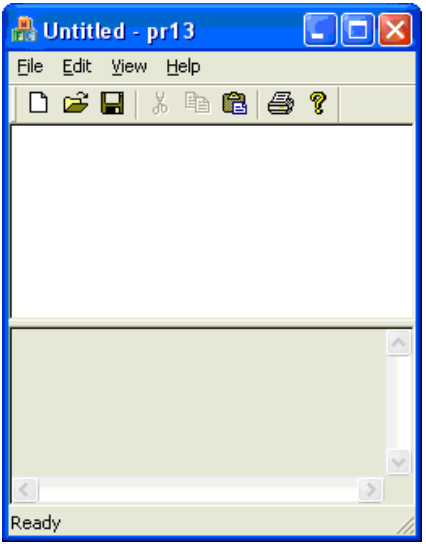


Рис. 13.6. Нижнее окно представления доступно только для чтения

Получить ссылку на управляющий элемент редактирования (для больших возможностей управления элементом) можно с помощью функции:

```
CEdit& CEditView::GetEditCtrl() const;
```

Установить флаг только для чтения можно функцией:

```
BOOL CEdit::SetReadOnly(        // 0 - ошибка
   BOOL bReadOnly = TRUE);      // TRUE - чтение, FALSE - чтение/запись
```

Результаты работы программы показаны на рис. 13.6.

13.1.3. Обработка действий в верхнем окне представления

Добавим в класс представления обработку нажатия правой кнопки мыши (рис. 13.7):

- в окне **Class View** вызвать контекстное меню для класса `Cpr13View` и выполнить команду **Properties**;

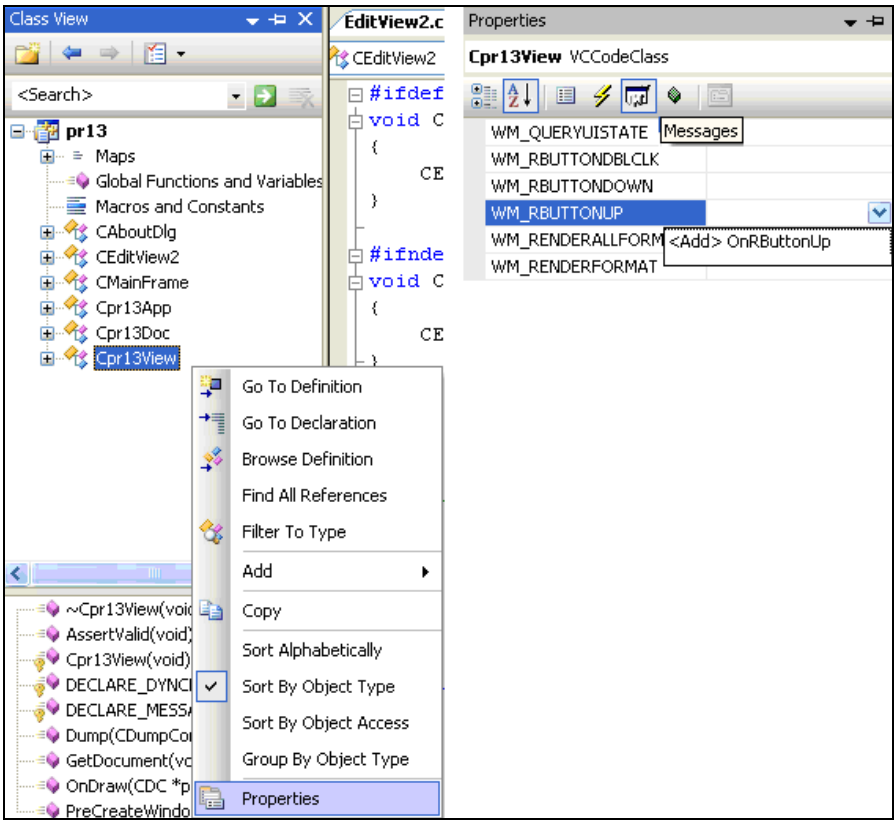


Рис. 13.7. Добавление обработки сообщения нажатия правой кнопки мыши

- в окне **Properties** выбрать вкладку **Messages**;
- в списке сообщений найти `WM_RBUTTONUP` и выбрать **<Add> OnRButtonUp**.

Изменения в файлах приведены в листингах 13.10 и 13.11.

Листинг 13.10. Изменения в файле pr13View.h для обработки сообщения о нажатии правой кнопки мыши

```
// ...
class Cpr13View : public CView
{
// ...
public:
   afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
};
```

Листинг 13.11. Изменения в файле pr13View.cpp для обработки сообщения о нажатии правой кнопки мыши

```
// ...
BEGIN_MESSAGE_MAP(Cpr13View, CView)
   ON_WM_RBUTTONUP()
END_MESSAGE_MAP()
// ...
void Cpr13View::OnRButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   CView::OnRButtonUp(nFlags, point);
}
```

Обработку рисования (появления `'X'` в месте нажатия правой кнопки мыши) надо производить в функции `Cpr13View::OnDraw()`. Если это делать в `Cpr13View::OnRButtonUp()`, то при изменении окна (например, при его сворачивании и разворачивании) данные исчезнут, т. к. для перерисовки окна будет вызвана `Cpr13View::OnDraw()`, где нет никаких действий. Все данные для удобства будем хранить в стандартном контейнерном классе `vector`. Изменения в файлах приведены в листингах 13.12 и 13.13.

Листинг 13.12. Изменения в файле pr13View.h для отображения 'X' в том месте, где была нажата правая кнопка мыши

```
// ...
#pragma once

#include <vector>
using namespace std;

class Cpr13View : public CView
{
// ...
public:
   afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
   vector<CPoint> vect;              // Вектор для хранения координат
};
```

Листинг 13.13. Изменения в файле pr13View.cpp для отображения 'X' в том месте, где была нажата правая кнопка мыши

```
// ...
Cpr13View::~Cpr13View()
{
   vect.clear();                  // Очистка вектора
}
// ...
void Cpr13View::OnDraw(CDC* pDC)
{
   Cpr13Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if(!pDoc)
      return;

   // TODO: add draw code for native data here
   int sz = (int)vect.size();      // Размер вектора
   for(int i=0; i<sz;i++)          // Вывод в окно
```

```
   {
      pDC->TextOutA(vect[i].x, vect[i].y, "X");
   }
}
// ...
void Cpr13View::OnRButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   vect.push_back(point);        // Занесение координат в вектор
   this->RedrawWindow();         // Перерисовка окна (вызов
                                 // Cpr13View::OnDraw())

   CView::OnRButtonUp(nFlags, point);
}
```

Контейнерный класс для работы с вектором определен как:

```
template <class Type,            // Тип хранимых данных
          class Allocator = allocator<Type>>
class vector
```

Очистить вектор можно с помощью функции:

```
void vector::clear();
```

Получить размер вектора можно функцией:

```
size_type vector:: size() const;
```

Новое значение в конец вектора заносится функцией:

```
void vector::push_back(
   const Type& _Val);               // Заносимое значение
```

Перерисовка окна или области: выполняется функцией

```
BOOL CWnd::RedrawWindow(          // 0 - ошибка
   LPCRECT lpRectUpdate = NULL,   // Указатель на координаты
                                  // перерисовываемого
                                  // прямоугольника (CRect)
   CRgn* prgnUpdate = NULL,       // Указатель на перерисовываемый регион
   UINT flags = RDW_INVALIDATE |  // Флаги перерисовки окна
                RDW_UPDATENOW | RDW_ERASE);
```

Если указатели `lpRectUpdate` и `prgnUpdate` равны `NULL`, то перерисовывается вся рабочая область окна.

Значения флагов перерисовки (`flags`) могут быть следующими:

- флаги, разрешающие перерисовку окна:
  - `RDW_ERASE` — посылает окну сообщение о перерисовке (`WM_ERASEBKGND`). Должен использоваться только вместе с `RDW_INVALIDATE`;
  - `RDW_FRAME` — посылает каждой составляющей окна (*не являющейся его клиентской* (рабочей) областью) сообщение `WM_NCPAINT`. Должен использоваться только вместе с `RDW_INVALIDATE`;
  - `RDW_INTERNALPAINT` — посылает окну сообщение `WM_PAINT` независимо от того, входит ли данное окно в обновляемую область;
  - `RDW_INVALIDATE` — перерисовывает область, заданную в `lpRectUpdate` *или* в `prgnUpdate` (только один из них может иметь значение, отличное от `NULL`);
- флаги, запрещающие перерисовку окна:
  - `RDW_NOERASE` — отменяет обработку сообщения о перерисовке (`WM_ERASEBKGND`);
  - `RDW_NOFRAME` — отменяет обработку сообщения `WM_NCPAINT`. Должен использоваться вместе с `RDW_VALIDATE` и, обычно, с `RDW_NOCHILDREN`;
  - `RDW_NOINTERNALPAINT` — запрещает посылку любых внутренних сообщений `WM_PAINT`;
  - `RDW_VALIDATE` — отменяет перерисовку областей, заданных в `lpRectUpdate` или в `prgnUpdate`;
- флаги, определяющие момент перерисовки окна:
  - `RDW_ERASENOW` — посылает в указанное окно сообщение `WM_NCPAINT` и `WM_ERASEBKGND` перед выходом из функции, при этом задерживается сообщение `WM_PAINT`;
  - `RDW_UPDATENOW` — посылает в указанное окно сообщение `WM_NCPAINT`, `WM_ERASEBKGND` и `WM_PAINT` перед выходом из функции;
- флаги, определяющие обрабатываемые окна:
  - `RDW_ALLCHILDREN` — перерисовываются все указанные окна;
  - `RDW_NOCHILDREN` — перерисовываются все указанные окна, кроме дочерних.

Результат работы программы показан на рис. 13.8.

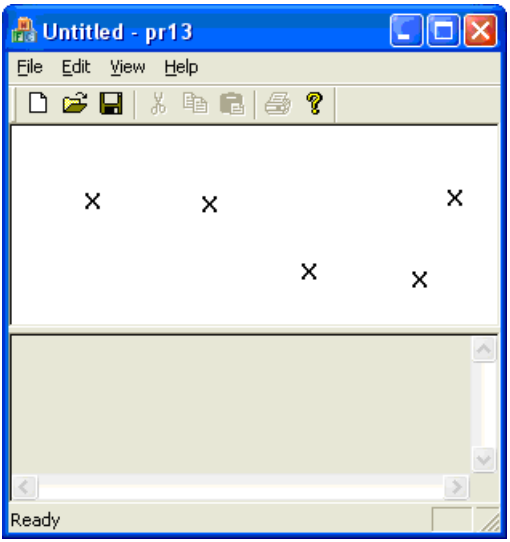


Рис. 13.8. Обработка нажатия правой кнопки мыши в верхнем окне представления

13.1.4. Обработка действий в нижнем окне редактирования

Изменения в файле приведены в листинге 13.14.

Листинг 13.14. Изменения в файле pr13View.cpp для вывода координат 'x' в нижнее окно

```
// ...
#include "MainFrm.h"
#include "EditView2.h"
// ...
void Cpr13View::OnRButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   vect.push_back(point);          // Занесение координат в вектор
```

```
    this->RedrawWindow();          // Перерисовка окна (вызов
                                   // Cpr13View::OnDraw())

    // Вывод координат в нижнее окно (редактирования)
    CString s,                     // Текущие координаты
            sold;                  // Все старые (уже выведенные) координаты
    // Получить указатель на фреймовое окно
    CMainFrame *pfrm = (CMainFrame *)theApp.m_pMainWnd;
    // Получить указатель на второе (нижнее) окно редактирования
    CEditView2 *pv2 = (CEditView2 *)pfrm->m_wndSplitter.GetPane(1,0);
    // Записать в строку текущие координаты
    s.Format( "(%d;%d)", point.x, point.y);
    s += "\r\n";    // Добавить возврат каретки и переход на другую строку
    pv2->GetWindowTextA(sold);     // Получить текст из окна редактирования
    sold += s;                     // Добавить к нему новые координаты
    pv2->SetWindowTextA(sold);     // Вывести новый текст

    CView::OnRButtonUp(nFlags, point);
}
```

Результаты работы программы показаны на рис. 13.9.

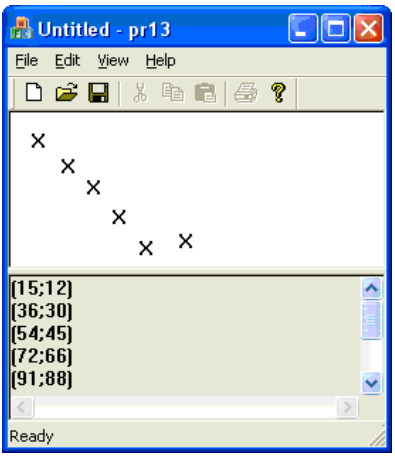


Рис. 13.9. Отображение точки нажатия правой кнопки мыши в верхнем окне и печать ее координат в нижнем окне

Недостаток в том, что когда много точек, нет автоматической прокрутки окна (всегда видно только первые значения). Чтобы увидеть последние, пользователь сам должен воспользоваться линейкой прокрутки. Это легко исправить. Изменения в файле приведены в листинге 13.15.

Листинг 13.15. Изменения в файле pr13View.cpp для автоматического изменения состояния вертикальной линейки прокрутки

```
// ..
void Cpr13View::OnRButtonUp(UINT nFlags, CPoint point)
{
   // ...
   pv2->SetWindowTextA(sold);              // Вывести новый текст

   // Получить ссылку на управляющий элемент редактирования
   CEdit &editctrl = pv2->GetEditCtrl();
   int sz = (int)vect.size();              // Размер вектора
   editctrl.LineScroll(sz);                // Прокрутить текст до конца

   CView::OnRButtonUp(nFlags, point);
}
```

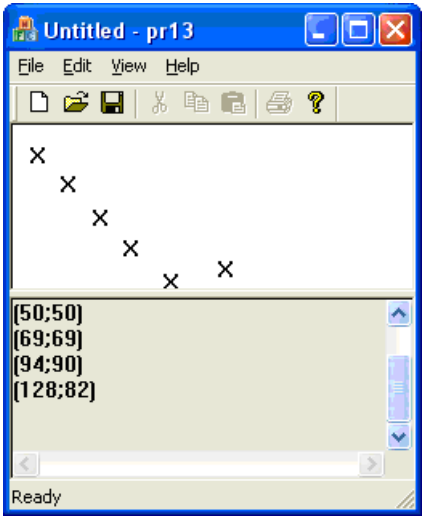


Рис. 13.10. Автоматическая прокрутка в нижнем окне представления

Прокрутка текста в окне редактирования выполняется функцией:

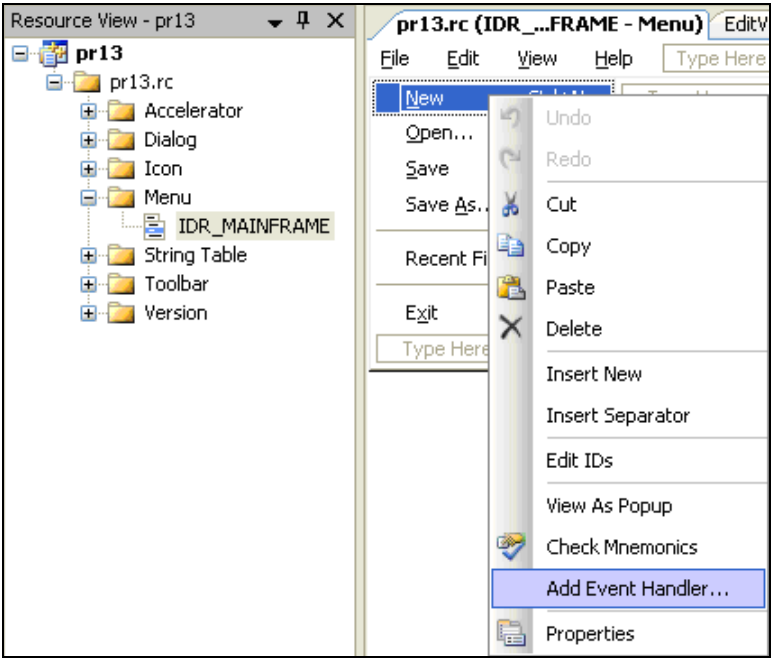
```
void CEdit::LineScroll(
   int nLines,                // Число строк, которые должны быть
                              // прокручены по вертикали
   int nChars = 0);           // Число символов для прокрутки по
                              // горизонтали
```

Результаты работы программы показаны на рис. 13.10.

13.1.5. Очистка экрана

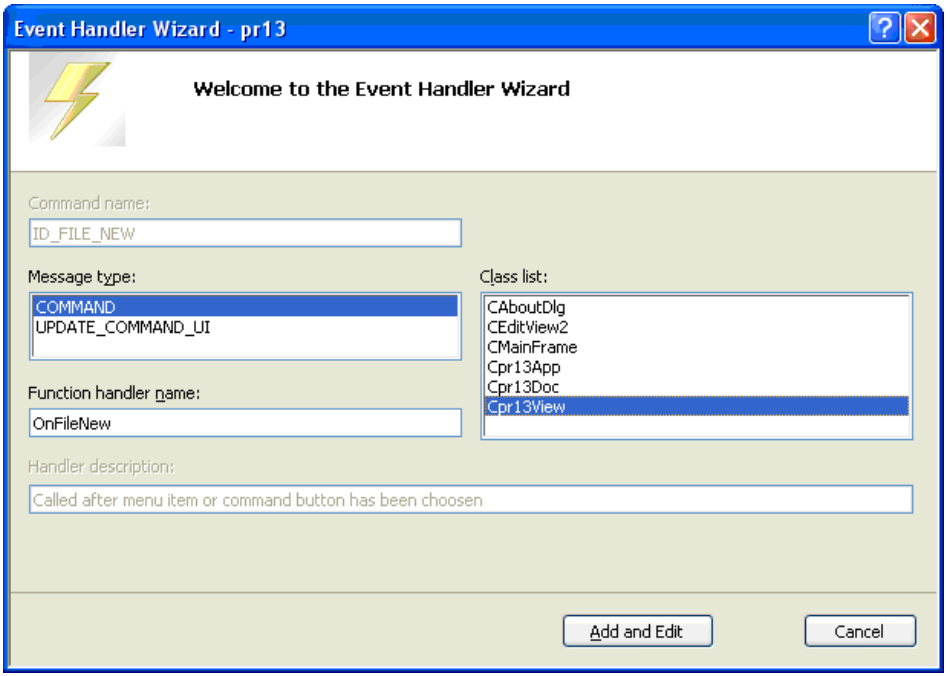
Добавим обработку очистки экрана. Для этого надо переопределить обработку меню **File | New** в классе окна представления:

- в окне ресурсов меню приложения вызвать для пункта **File | New** контекстное меню и выполнить команду **Add Event Handler** (рис. 13.11, *а*);



а

Рис. 13.11. Добавление обработки пункта меню (*а*)



б

Рис. 13.11. Функции обработки меню **File | New** (*б*)

- в окне **Event Handler Wizard** выбрать в списке классов **Class list** класс **Cpr13View** (рис. 13.11, *б*);
- нажать кнопку **Add and Edit**.

Изменения в файлах приведены в листингах 13.16 и 13.17.

Листинг 13.16. Изменения в файле pr13View.h для обработки меню выбора нового файла

```
// ...
class Cpr13View : public CView
{
   // ...
public:
   afx_msg void OnFileNew();
};
```

Листинг 13.17. Изменения в файле pr13View.cpp для обработки меню выбора нового файла

```
// ...
BEGIN_MESSAGE_MAP(Cpr13View, CView)
   ON_WM_RBUTTONUP()
   ON_COMMAND(ID_FILE_NEW, &Cpr13View::OnFileNew)
END_MESSAGE_MAP()
// ...
void Cpr13View::OnFileNew()
{
   // TODO: Add your command handler code here
   vect.clear();                    // Очистка вектора
   this->RedrawWindow();            // Перерисовка верхнего окна

   CMainFrame *pfrm = (CMainFrame *)theApp.m_pMainWnd;
   CEditView2 *pv2 = (CEditView2 *)pfrm->m_wndSplitter.GetPane(1,0);
   pv2->SetWindowTextA("");         // Очистка нижнего окна (вывод в него
                                    // пустой строки)
}
```

13.1.6. Некоторые полезные виды окон представления

Менять базовый класс для окна представления можно *только при включенной опции* **Document/View architecture support** (Поддержка архитектуры документ/представление) в свойствах создаваемого проекта (рис. 13.12).

Для работы с текстовыми файлами (с их автоматическим открытием, сохранением и возможностью редактировать, прокруткой и т. п.) надо выбрать на вкладке **Generated Classes** в свойствах создаваемого проекта базовый класс (список **Base class**) для окна представления **CEditView** (рис. 13.13).

Если работа с данными в окне представления аналогична работе со списком, то в качестве базового класса надо выбрать **CListView**.

Если выводимая в окно представления информация достаточно большая, для того чтобы не делать самостоятельно обработку сообщений линейки прокрутки, надо выбрать класс **CScroolView**.

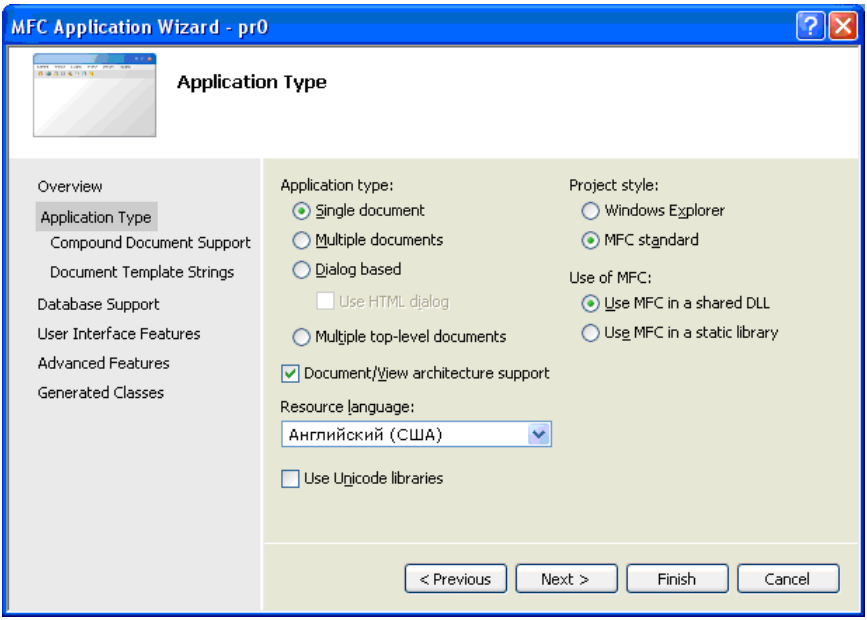


Рис. 13.12. Выбор поддержки архитектуры документ/представление

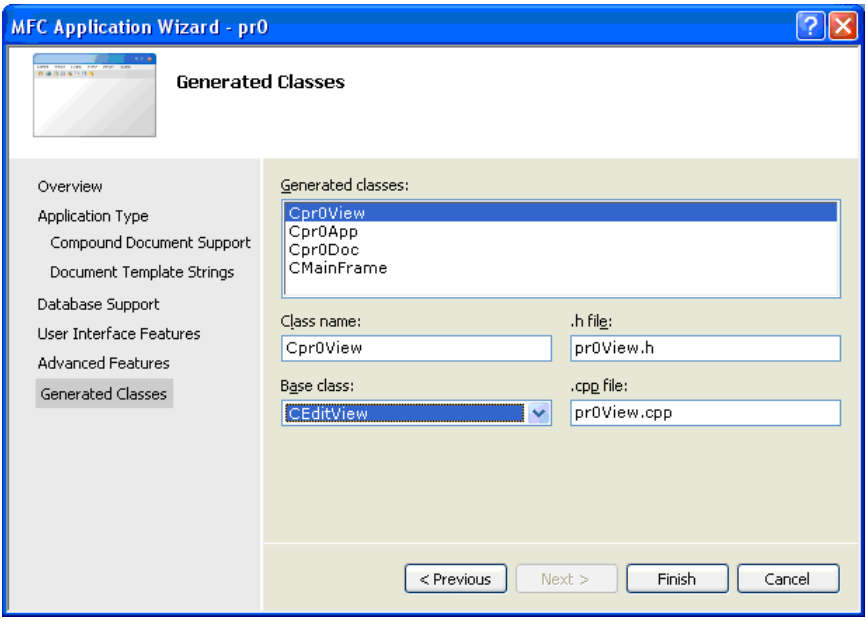


Рис. 13.13. Класс для работы с текстовыми файлами

Если в окне представления надо разместить кнопки или другие элементы управления (т. е. вид окна представления должен быть похож на диалоговое окно), то (чтобы это не делать вручную) надо выбрать класс **CFormView**. При нажатии кнопки **Finish** при завершении создания проекта появится окно сообщения (рис. 13.14), где надо нажать кнопку **Yes**. Для добавления управляющих элементов надо воспользоваться редактором ресурсов, где окно представления добавлено в ресурс **Dialog** в виде диалогового окна с идентификатором `IDD_PRO_FORM` (рис. 13.15). Добавление происходит аналогично добавлению в окно диалога. Обработка сообщений элементов управления добавляется аналогично тому, как в окне диалога, только в класс представления. Для начальной инициализации элементов управления надо использовать функцию `OnInitialUpdate()` (ее не надо добавлять, она генерируется автоматически). Вид такого приложения показан на рис. 13.16.

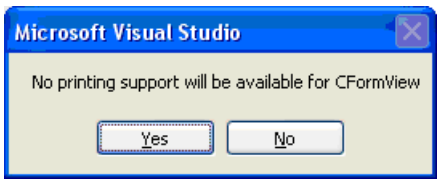


Рис. 13.14. Сообщение при выборе базового класса окна представления `CFormView`

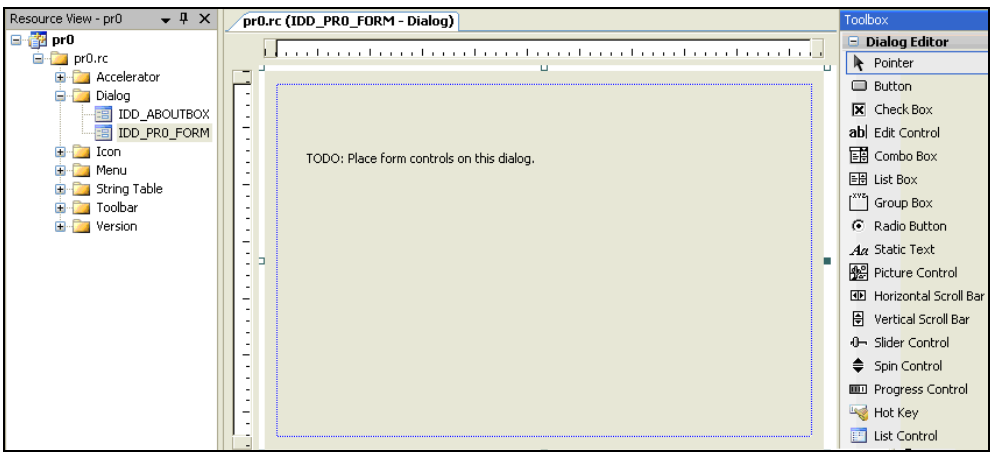


Рис. 13.15. Ресурс окна представления для класса `CFormView`

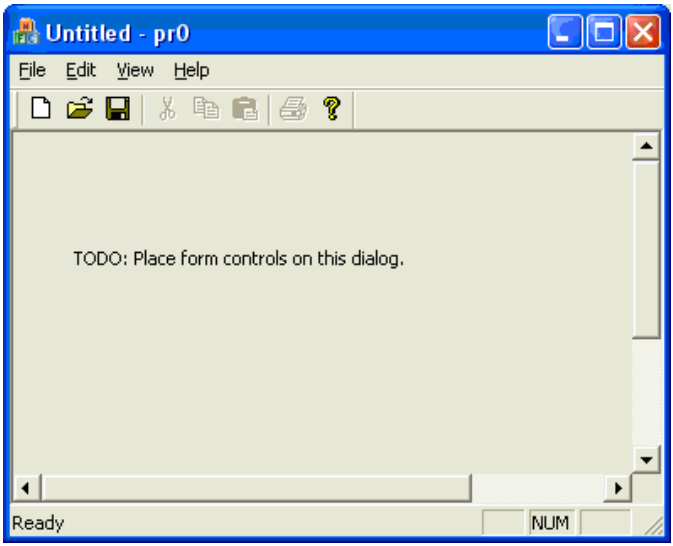


Рис. 13.16. Приложение в виде формы

13.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 13.18. Файл MainFrm.h (объявление класса фрейма)

```
// ...
class CMainFrame : public CFrameWnd
{
// ...
protected:
   //CSplitterWnd m_wndSplitter;
public:
   CSplitterWnd m_wndSplitter;
// ...
};
```

Листинг 13.19. Файл MainFrm.cpp (определение класса фрейма)

```
// ...
#include "MainFrm.h"

#include "EditView2.h"
// ...
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                                CCreateContext* pContext)
{
   //return m_wndSplitter.Create(this,
   //   2, 2,             // TODO: adjust the number of rows, columns
   //   CSize(10, 10),    // TODO: adjust the minimum pane size
   //   pContext);

   // Создание статического разделенного окна из двух горизонтальных
   // областей
   // (2 строки, 1 колонка)
   if(!this->m_wndSplitter.CreateStatic(this, 2,1))
      return FALSE;

   // Создание первой (верхней) области
   if(!this->m_wndSplitter.CreateView(0, 0, pContext->m_pNewViewClass,
                                      CSize(lpcs->cx, 200),pContext))
      return FALSE;
   // Создание второй (нижней) области
   if(!this->m_wndSplitter.CreateView(1, 0, RUNTIME_CLASS(CEditView2),
                                      CSize(lpcs->cx,200), pContext))
      return FALSE;

   // Сделать активной вторую область
   this->SetActiveView((CView *)m_wndSplitter.GetPane(1,0));

   return TRUE;
}
// ...
```

Листинг 13.20. Файл EditView2.h (объявление нового класса редактирования, добавленного в проект класса)

```
// ...
class CEditView2 : public CEditView
{
// ...
public:
   virtual void OnInitialUpdate();
};
```

Листинг 13.21. Файл EditView2.cpp (определение нового класса редактирования, добавленного в проект класса)

```
// ...
void CEditView2::OnInitialUpdate()
{
   CEditView::OnInitialUpdate();

   // TODO: Add your specialized code here and/or call the base class
   this->GetEditCtrl().SetReadOnly();      // Окно только для чтения
}
```

Листинг 13.22. Файл pr13View.h (объявление класса представления)

```
// ...
#pragma once

#include <vector>
using namespace std;

class Cpr13View : public CView
{
// ...
public:
   afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
   vector<CPoint> vect;            // Вектор для хранения координат
public:
   afx_msg void OnFileNew();
};
```

Листинг 13.23. Файл pr13View.cpp (определение класса представления)

```
// ...
#include "pr13View.h"

#include "MainFrm.h"
#include "EditView2.h"
// ...
BEGIN_MESSAGE_MAP(Cpr13View, CView)
   ON_WM_RBUTTONUP()
   ON_COMMAND(ID_FILE_NEW, &Cpr13View::OnFileNew)
END_MESSAGE_MAP()
// ...
Cpr13View::~Cpr13View()
{
   vect.clear();                    // Очистка вектора
}
// ...
void Cpr13View::OnDraw(CDC* pDC)
{
   Cpr13Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if(!pDoc)
      return;

   // TODO: add draw code for native data here
   int sz = (int)vect.size();       // Размер вектора
   for(int i=0; i<sz;i++)           // Вывод в окно
   {
      pDC->TextOutA(vect[i].x,  vect[i].y, "X");
   }
}
// ...
void Cpr13View::OnRButtonUp(UINT nFlags, CPoint point)
{
   // TODO: Add your message handler code here and/or call default
   vect.push_back(point);     // Занесение координат в вектор
```

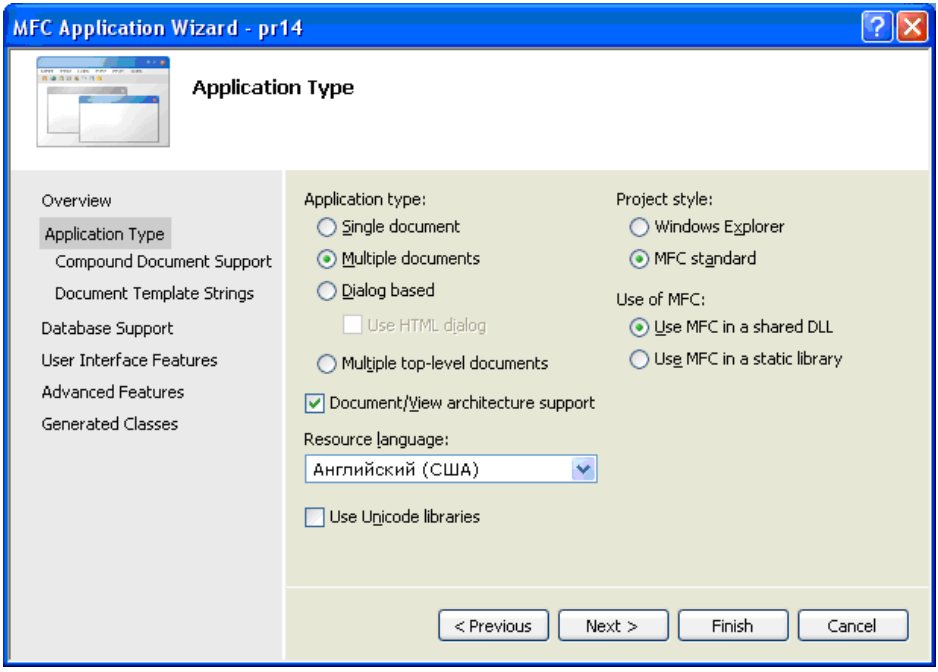
```
    this->RedrawWindow();       // Перерисовка окна (вызов
                                // Cpr13View::OnDraw())
    // Вывод координат в нижнее окно (редактирования)
    CString s,                  // Текущие координаты
            sold;               // Все старые (уже выведенные) координаты
    // Получить указатель на фреймовое окно
    CMainFrame *pfrm = (CMainFrame *)theApp.m_pMainWnd;
    // Получить указатель на второе (нижнее) окно редактирования
    CEditView2 *pv2 = (CEditView2 *)pfrm->m_wndSplitter.GetPane(1,0);
    // Записать в строку текущие координаты
    s.Format("(%d;%d)", point.x, point.y);
    s += "\r\n";    // Добавить возврат каретки и переход на другую строку
    pv2->GetWindowTextA(sold);    // Получить текст из окна редактирования
    sold += s;                    // Добавить к нему новые координаты
    pv2->SetWindowTextA(sold);    // Вывести новый текст

    // Получить ссылку на управляющий элемент редактирования
    CEdit &editctrl = pv2->GetEditCtrl();
    int sz = (int)vect.size();     // Размер вектора
    editctrl.LineScroll(sz);       // Прокрутить текст до конца

    CView::OnRButtonUp(nFlags, point);
}
void Cpr13View::OnFileNew()
{
    // TODO: Add your command handler code here
    vect.clear();                  // Очистка вектора
    this->RedrawWindow();          // Перерисовка верхнего окна

    CMainFrame *pfrm = (CMainFrame *)theApp.m_pMainWnd;
    CEditView2 *pv2 = (CEditView2 *)pfrm->m_wndSplitter.GetPane(1,0);
    pv2->SetWindowTextA("");       // Очистка нижнего окна (вывод в него
                                   // пустой строки)
}
```

Глава 14

Многодокументное приложение

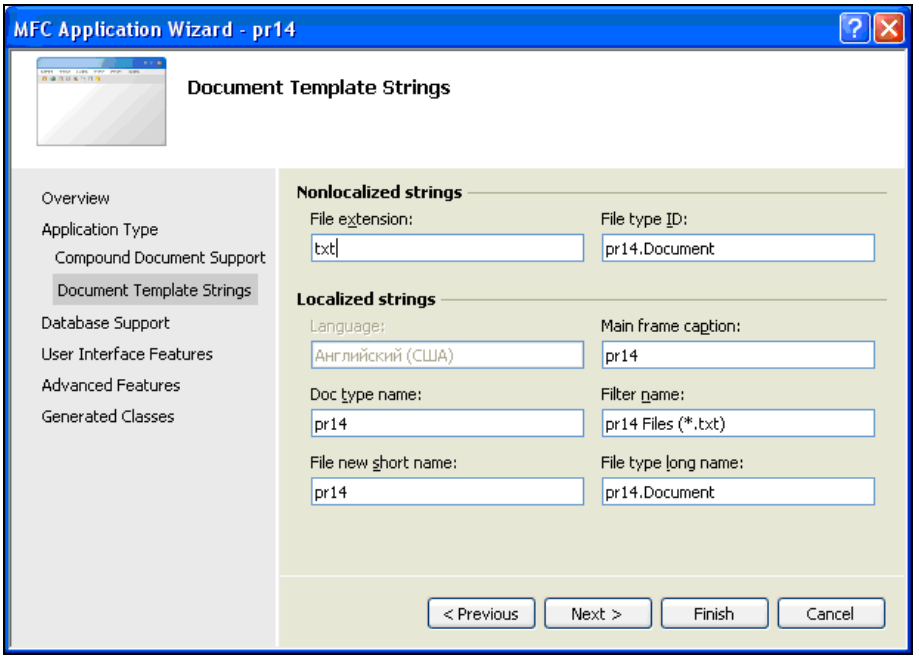
Создадим новый проект, на примере которого рассмотрим работу с несколькими окнами и типами документов.

Создайте проект **pr14**, *не изменяя тип приложения* на SDI, задав на вкладке **Document Template Strings** в поле **File extension** расширение `txt` и выбрав для окна представления базовый класс **CEditView**.

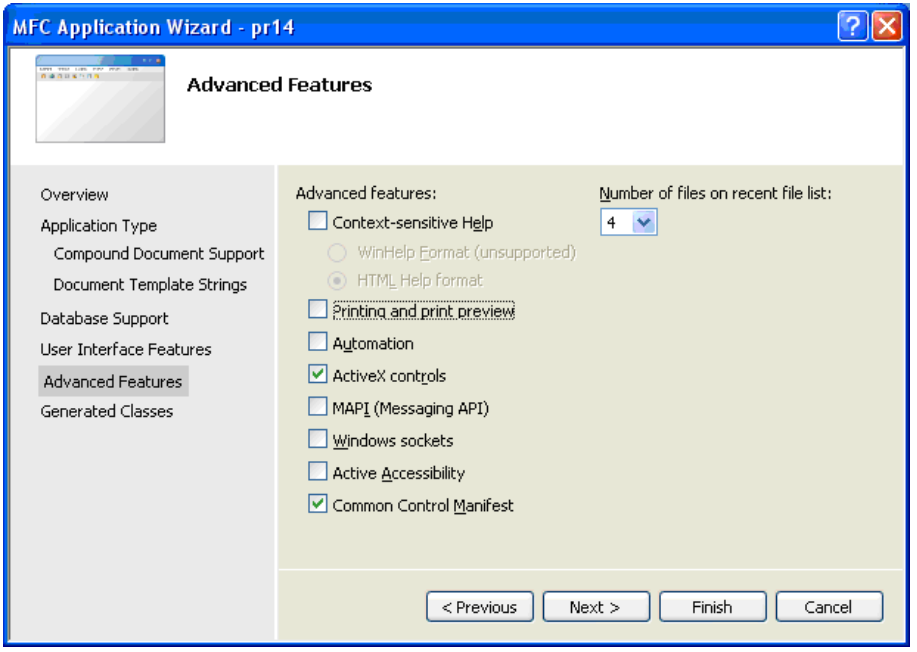


а

Рис. 14.1. Создание многодокументного (MDI) приложения (*а*)

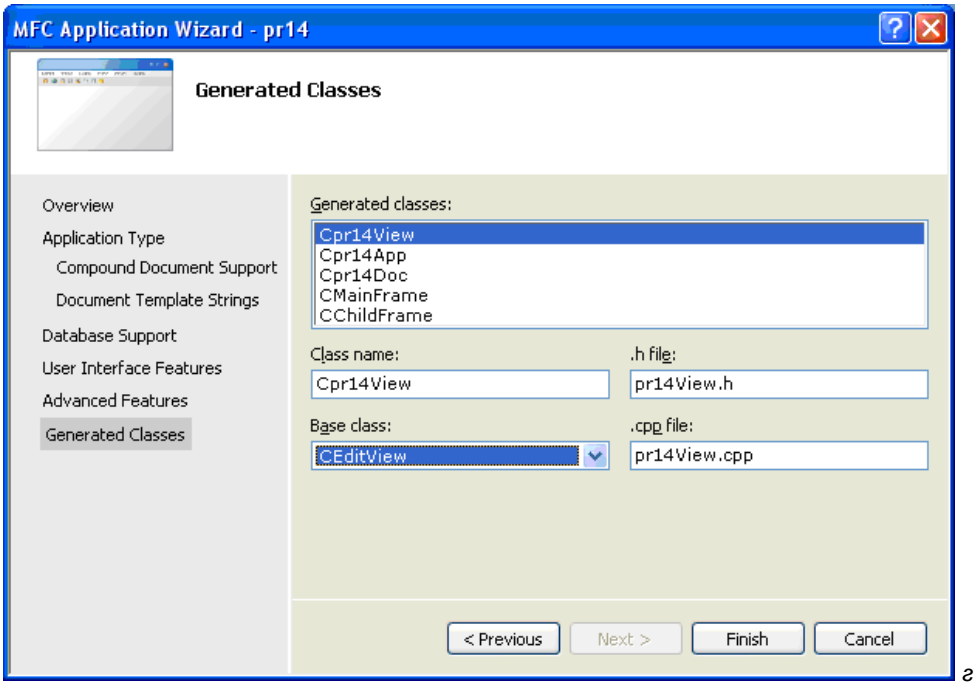


б



в

Рис. 14.1. Задание типа файлов по умолчанию (*б*),
отключение опции печати и предварительного просмотра (*в*)



г

Рис. 14.1. Выбор базового класса для окна представления (*г*)

Опции, которые надо изменить, относительно изначально предложенных мастером:

- на вкладке **Application Type** (рис. 14.1, *а*):
  - без Unicode;
- на вкладке **Document Template Strings** (рис. 14.1, *б*):
  - в поле **File extension** ввести `txt`;
- на вкладке **Advanced Features** (рис. 14.1, *в*):
  - без печати и предварительного просмотра;
- на вкладке **Generated Classes** (рис. 14.1, *г*):
  - в списке **Generated Classes** выбрать класс **Cpr14View**;
  - в списке **Base class** найти и выбрать класс **CEditView**.

14.1. Описание программы

14.1.1. Архитектура MDI-приложения

Архитектура многодокументного (MDI, Multiple Document Interface — многодокументный интерфейс) приложения похожа на архитектуру однодокументного (SDI) приложения *(см. разд. 10.1)* и приведена на рис. 14.2.

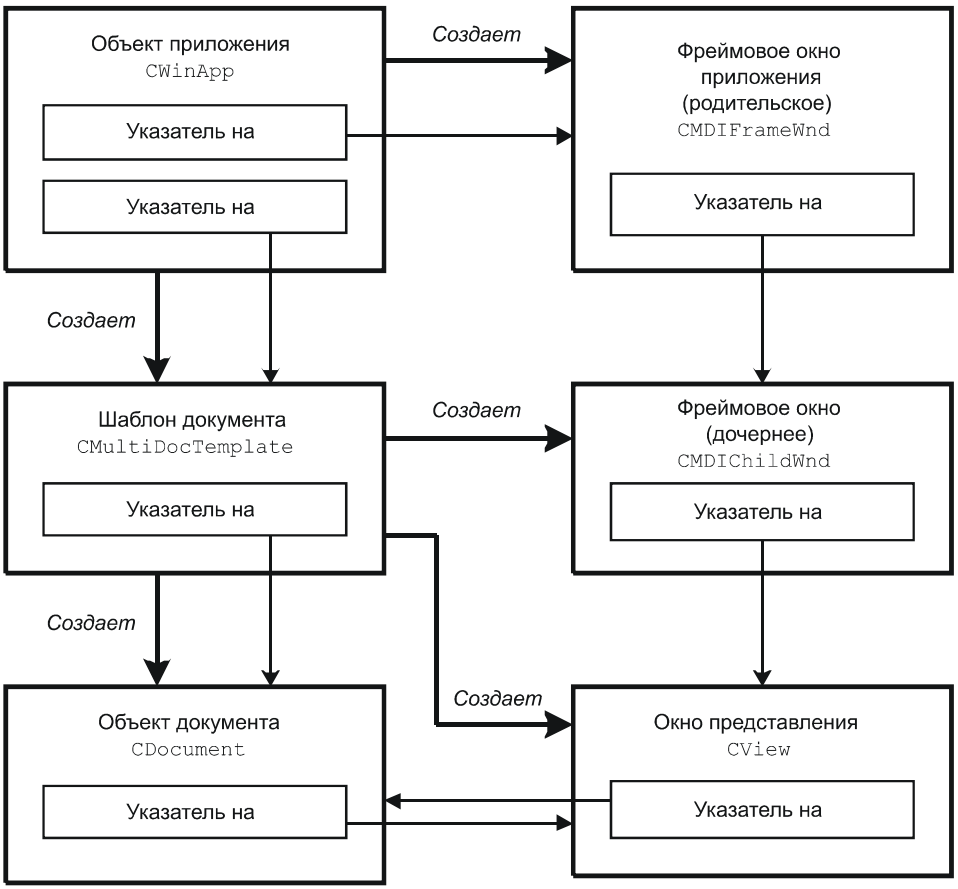


Рис. 14.2. Архитектура MDI-приложения

Изменения в программе, связанные с созданием MDI-приложения (автоматически сделанные мастером), приведены в листингах 14.1—14.5.

Листинг 14.1. Файл pr14.cpp (определение класса приложения)

```
// ...
#include "ChildFrm.h"
// ...
BOOL Cpr14App::InitInstance()
{
   // ...
   CMultiDocTemplate* pDocTemplate;
   pDocTemplate = new CMultiDocTemplate(IDR_pr14TYPE,
      RUNTIME_CLASS(Cpr14Doc),     // Текстовый документ
      RUNTIME_CLASS(CChildFrame),  // Стандартный дочерний фрейм MDI
      RUNTIME_CLASS(Cpr14View));   // Представление текстового документа
   if(!pDocTemplate)
      return FALSE;
   AddDocTemplate(pDocTemplate);   // Добавить в шаблон

   // create main MDI Frame window
   CMainFrame* pMainFrame = new CMainFrame;
   if (!pMainFrame || !pMainFrame->LoadFrame(IDR_MAINFRAME))
   {
      delete pMainFrame;
      return FALSE;
   }
   m_pMainWnd = pMainFrame;
   // ...
   // The main window has been initialized, so show and update it
   pMainFrame->ShowWindow(m_nCmdShow);
   pMainFrame->UpdateWindow();

   return TRUE;
}
```

Листинг 14.2. Файл `pr14Doc.cpp` (определение класса документа)

```
// ...
void Cpr14Doc::Serialize(CArchive& ar)
```

```
{
   // CEditView contains an edit control which handles all serialization
   reinterpret_cast<CEditView*>(m_viewList.GetHead())->SerializeRaw(ar);
}
```

Листинг 14.3. Файл ChildFrm.h (объявление класса фрейма дочернего окна)

```
// ChildFrm.h : interface of the CChildFrame class
//
#pragma once

class CChildFrame : public CMDIChildWnd
{
   DECLARE_DYNCREATE(CChildFrame)
public:
   CChildFrame();
// Attributes
public:
// Operations
public:
// Overrides
   virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
// Implementation
public:
   virtual ~CChildFrame();
#ifdef _DEBUG
   virtual void AssertValid() const;
   virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
   DECLARE_MESSAGE_MAP()
};
```

Листинг 14.4. Файл ChildFrm.cpp (определение класса фрейма дочернего окна)

```
// ChildFrm.cpp : implementation of the CChildFrame class
//
#include "stdafx.h"
#include "pr14.h"
#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CChildFrame
IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
END_MESSAGE_MAP()

// CChildFrame construction/destruction
CChildFrame::CChildFrame()
{
   // TODO: add member initialization code here
}
CChildFrame::~CChildFrame()
{
}
BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
   // TODO: Modify the Window class or styles here by modifying the
   // CREATESTRUCT cs
   if(!CMDIChildWnd::PreCreateWindow(cs))
      return FALSE;
   return TRUE;
}
// CChildFrame diagnostics
#ifdef _DEBUG
```

```
void CChildFrame::AssertValid() const
{
   CMDIChildWnd::AssertValid();
}
void CChildFrame::Dump(CDumpContext& dc) const
{
   CMDIChildWnd::Dump(dc);
}
#endif //_DEBUG
// CChildFrame message handlers
```

Листинг 14.5. Файл pr14View.h (объявление класса окна представления)

```
// ...
class Cpr14View : public CEditView
{
// ...
};
```

Здесь для создания шаблона документа используется не класс `CSingleDocTemplate`, а аналогичный ему:

```
class CMultiDocTemplate : public CDocTemplate
```

Теперь есть два фреймовых окна: главное `CMainFrame : public CMDIFrameWnd` и фрейм дочерних окон `CChildFrame : public CMDIChildWnd`, создаваемых внутри главного окна.

```
class CMDIFrameWnd : public CFrameWnd
class CMDIChildWnd : public CFrameWnd
```

У приложения теперь имеется два ресурса меню:

- `IDR_MAINFRAME` (рис. 14.3), которое загружается при создании главного фреймового окна `pMainFrame->LoadFrame(IDR_MAINFRAME)` и используется, если нет открытых дочерних окон;
- `IDR_pr14TYPE` (рис. 14.4), которое загружается в конструкторе шаблона документа:

  ```
  pDocTemplate = new CMultiDocTemplate(IDR_pr14TYPE, ...);
  ```

Это меню появляется, когда есть открытые дочерние окна (в нем, помимо новых пунктов **File | Close**, **File | Save** и **File | Save As**, добавляется новый пункт **Window** — для выбора расположения открытых окон).

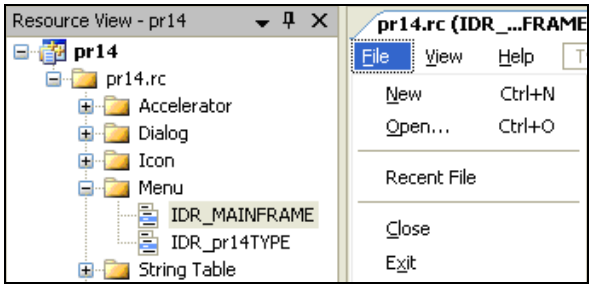
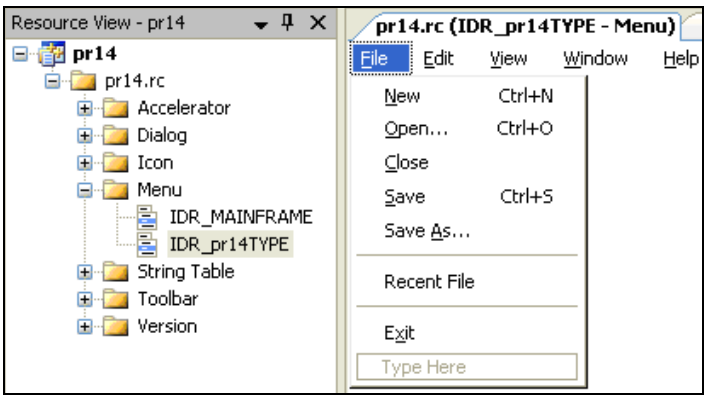
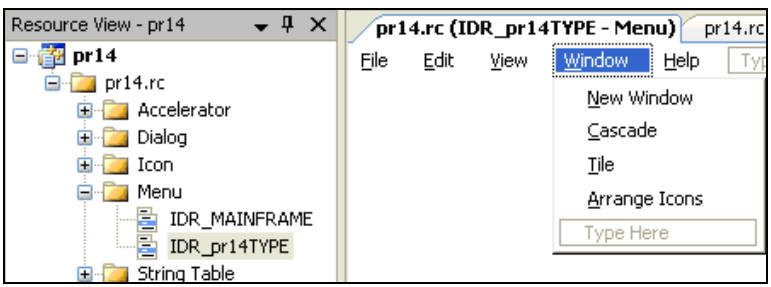


Рис. 14.3. Меню приложения, если нет открытых дочерних окон



а



б

Рис. 14.4. Меню приложения для работы с открытыми файлами (*а*) и дополнительное меню для выбора расположения открытых окон (*б*)

Обычно для работы достаточно добавлять новые пункты меню в ресурс `IDR_pr14TYPE`, но если новый пункт меню должен работать и при отсутствии дочерних окон, то его придется добавлять и в `IDR_pr14TYPE`, и в `IDR_MAINFRAME`.

Результат работы программы показан на рис. 14.5.

При запуске программы автоматически открывается новое окно. Если этого не должно происходить, то можно добавить код, приведенный в листинге 14.6 (в данном проекте он не добавлен для возможности внесения последующих изменений).

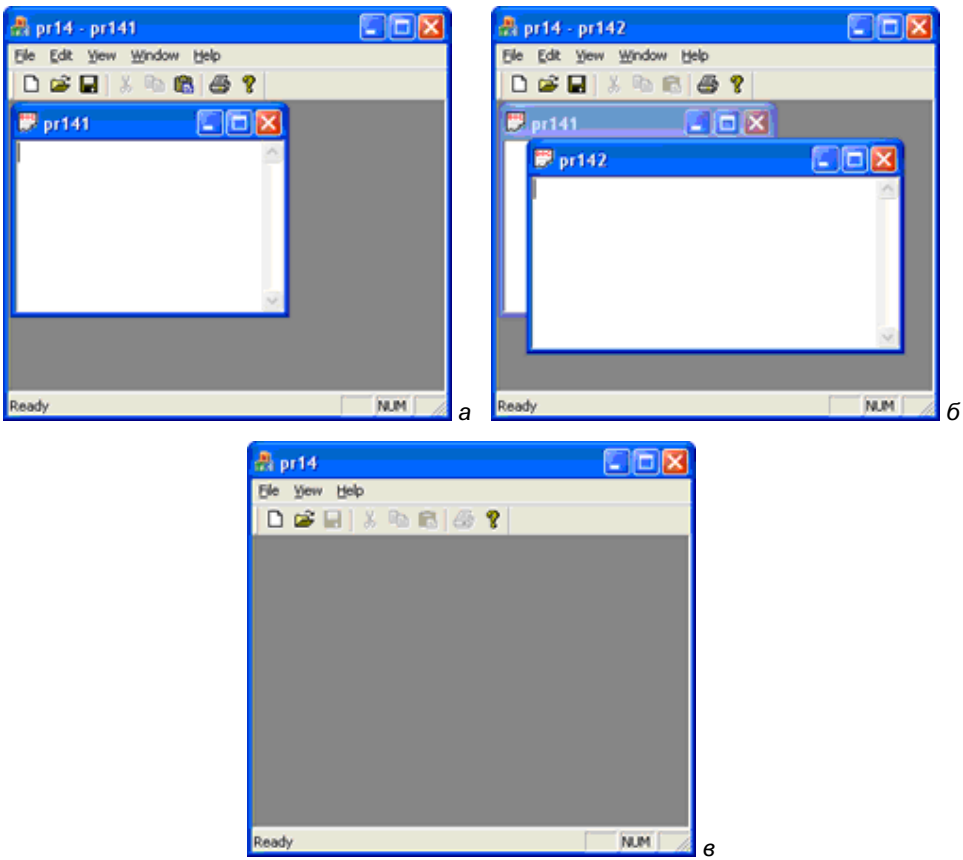


Рис. 14.5. Работа MDI-приложения при запуске программы (*а*), при открытии нескольких окон (*б*) и при закрытии всех окон (*в*)

Листинг 14.6. Изменения в файле pr14Doc.cpp для запрещения автоматического создания нового окна редактирования при запуске программы

```
// ...
BOOL Cpr14Doc::OnNewDocument()
{
   static int f = 0;
   if(!f)          // Если в первый раз -  новое окно не создавать
   {
      f = 1;
      return FALSE;
   }

   if(!CDocument::OnNewDocument())
      return FALSE;
   // TODO: add reinitialization code here
   // (SDI documents will reuse this document)
   return TRUE;
}
```

Изменим заданное по умолчанию имя для новых файлов. Для этого в ресурсах проекта в таблице строк **String Table** надо изменить в строке `IDR_pr14TYPE` текст, выделенный полужирным шрифтом:

```
\n|pr14|\npr14\npr14 Files (*.txt)\n.txt\npr14.Document\npr14.Document
```

на:

```
\n|New txt|\npr14\npr14 Files (*.txt)\n.txt\npr14.Document\npr14.Document
```

Строка, заданная в **String Table**, имеет следующую структуру (поля разделяются символом `'\n'`):

```
ПОЛЕ1|\n|ПОЛЕ2|\n|ПОЛЕ3|\n|ПОЛЕ4|\n|ПОЛЕ5|\n|ПОЛЕ6|\n|ПОЛЕ7
```

Значения полей в таблице строк:

- `ПОЛЕ1` — заголовок окна SDI-приложения. Для MDI-приложений это поле не используется;
- `ПОЛЕ2` — имя файла, присваиваемое новому документу по умолчанию. Если это поле отсутствует, то используется имя `Untitle`;

- `ПОЛЕ3` — имя типа документа, которое используется в списке документов при создании нового файла;
- `ПОЛЕ4` — описание типа документа в списке фильтра при открытии/сохранении файла;
- `ПОЛЕ5` — расширение для файлов этого типа документов;
- `ПОЛЕ6` — идентификатор типа документа, хранящийся в реестре Windows;
- `ПОЛЕ7` — имя типа документа, хранящееся в реестре Windows.

Результат работы программы показан на рис. 14.6.

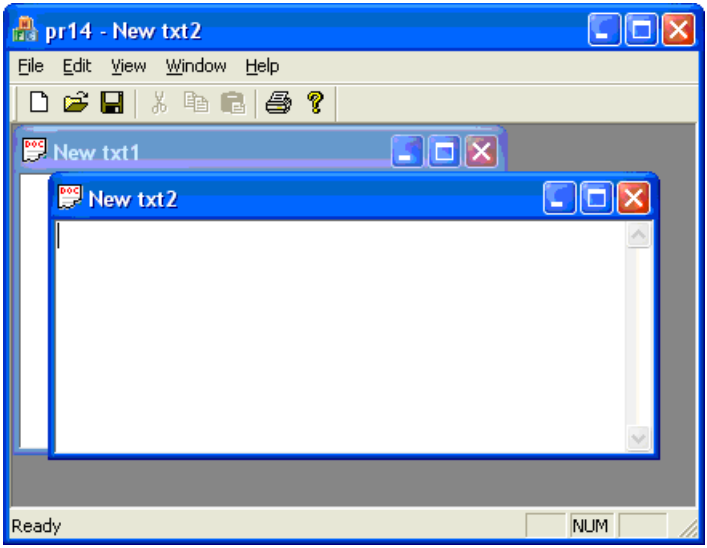


Рис. 14.6. Изменение заголовка дочерних окон

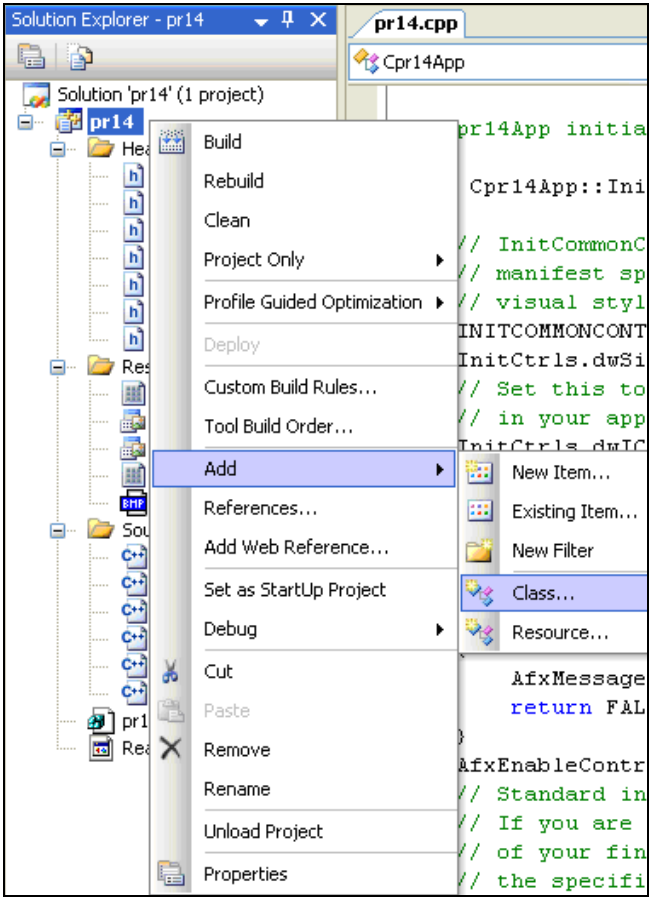
14.1.2. Работа с несколькими типами документов

MDI-приложения логично использовать не только для работы с несколькими документами одного типа, но и для работы с разными типами документов. В этом проекте будет сделана некоторая комбинация предыдущих проектов **pr12** и **pr13**. Пользователь сможет открывать *текстовые документы* (txt) для редактирования (это уже реализовано), где может задавать координаты точек, и *графические документы* (пусть это будут наши файлы grf), где можно по

нажатию правой кнопки мыши рисовать `'X'` (аналогично проекту **pr13**). И тот и другой документ будут поддерживать работу с архивом (для текстовых файлов это уже сделано автоматически), т. е. документы можно будет записывать/считывать с диска.

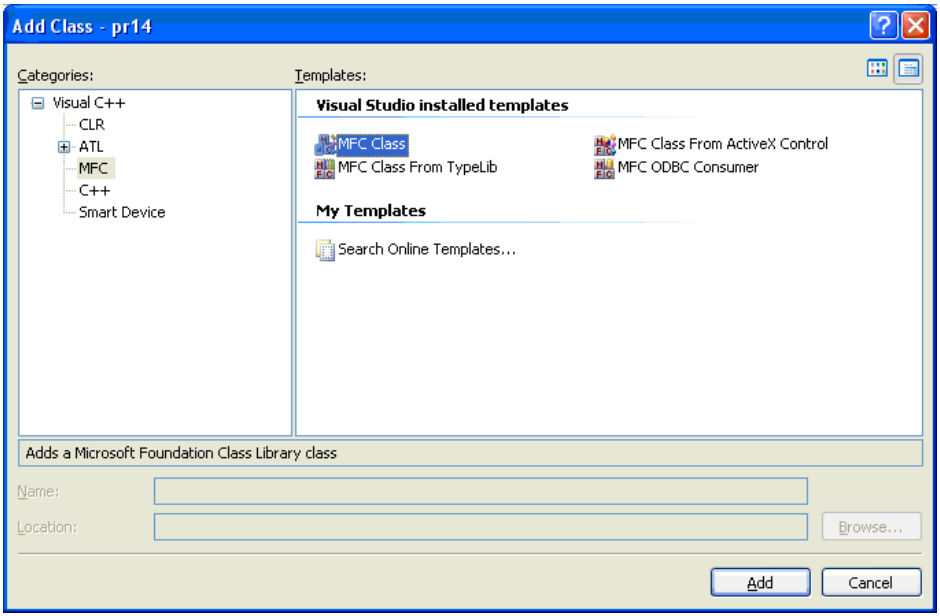
Для работы с новым типом документа надо добавить в проект два новых класса: `Cdoc2 : public CDocument` (рис 14.7) и `Cview2 : public CView` (рис. 14.8):

- в окне **Solution Explorer** вызвать контекстное меню для проекта **pr14** и выполнить команду **Add | Class**;

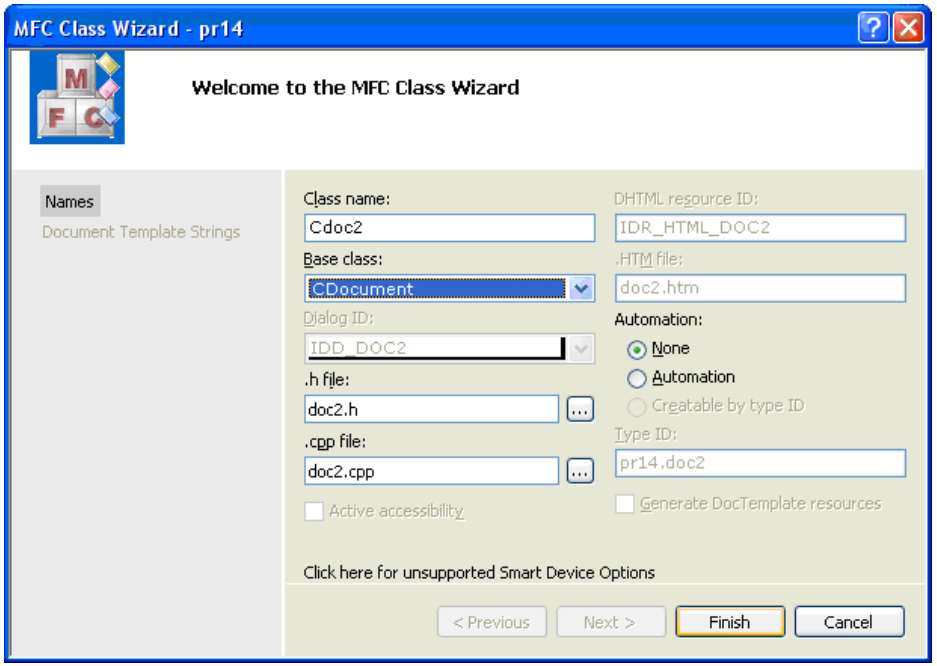


а

Рис. 14.7. Добавление нового класса в проект (*а*)



б



в

Рис. 14.7. Выбор типа нового класса (*б*)
и класс для работы с графическими документами (*в*)

- ❒ в окне **Add Class**:
  - выбрать в списке **Categories** узел **MFC**;
  - в списке **Templates** выбрать **MFC Class**;
  - нажать кнопку **Add**;
- ❒ в окне **MFC Class Wizard**:
  - в поле **Class name** ввести `Cdoc2`;
  - в списке классов **Base class** выбрать **CDocument**;
  - нажать кнопку **Finish**.

Класс `Cview2 : public CView` добавляется аналогичным образом:

- ❒ в окне **MFC Class Wizard**:
  - в поле **Class name** ввести `CView2`;
  - в списке классов **Base class** выбрать **CView**;
  - нажать кнопку **Finish**.

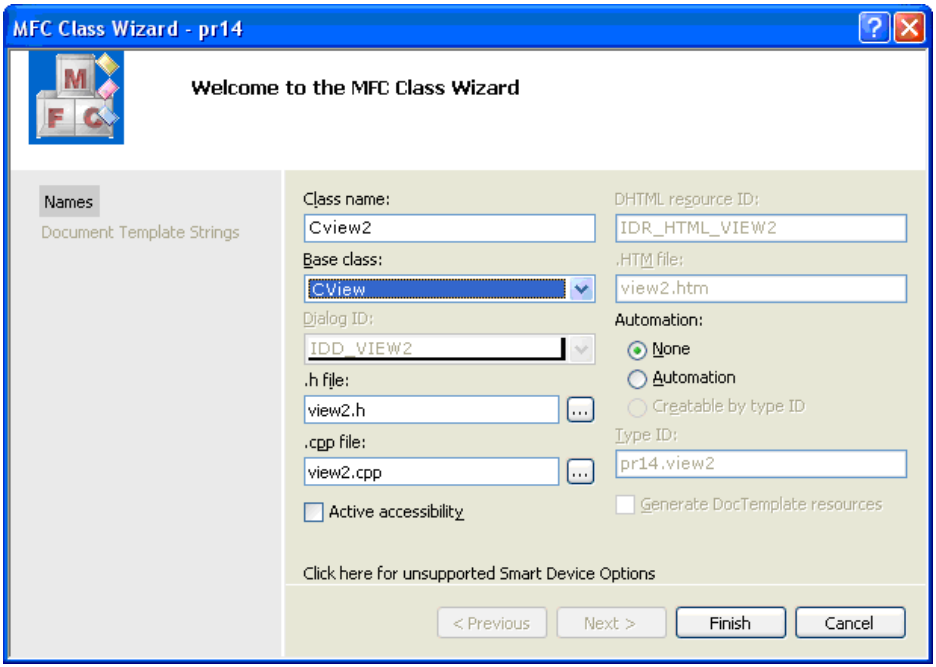


Рис. 14.8. Класс для представления графических документов

Изменения в программе приведены в листингах 14.7—14.10.

Листинг 14.7. Файл doc2.h (объявление класса документа графических данных)

```
#pragma once

// Cdoc2 document
class Cdoc2 : public CDocument
{
   DECLARE_DYNCREATE(Cdoc2)

public:
   Cdoc2();
   virtual ~Cdoc2();
#ifndef _WIN32_WCE
   virtual void Serialize(CArchive& ar);   // overridden for document i/o
#endif
#ifdef _DEBUG
   virtual void AssertValid() const;
#ifndef _WIN32_WCE
   virtual void Dump(CDumpContext& dc) const;
#endif
#endif

protected:
   virtual BOOL OnNewDocument();
   DECLARE_MESSAGE_MAP()
};
```

Листинг 14.8. Файл doc2.cpp (определение класса документа графических данных)

```
// doc2.cpp : implementation file
//
#include "stdafx.h"
#include "pr14.h"
#include "doc2.h"
```

```
// Cdoc2
IMPLEMENT_DYNCREATE(Cdoc2, CDocument)
Cdoc2::Cdoc2()
{
}
BOOL Cdoc2::OnNewDocument()
{
   if(!CDocument::OnNewDocument())
      return FALSE;
   return TRUE;
}
Cdoc2::~Cdoc2()
{
}
BEGIN_MESSAGE_MAP(Cdoc2, CDocument)
END_MESSAGE_MAP()

// Cdoc2 diagnostics
#ifdef _DEBUG
void Cdoc2::AssertValid() const
{
   CDocument::AssertValid();
}
#ifndef _WIN32_WCE
void Cdoc2::Dump(CDumpContext& dc) const
{
   CDocument::Dump(dc);
}
#endif
#endif //_DEBUG

#ifndef _WIN32_WCE
// Cdoc2 serialization
void Cdoc2::Serialize(CArchive& ar)
{
```

```
    if(ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}
#endif

// Cdoc2 commands
```

Листинг 14.9. Файл view2.h (объявление класса представления графических данных)

```
#pragma once

// Cview2 view
class Cview2 : public CView
{
    DECLARE_DYNCREATE(Cview2)
protected:
    Cview2();           // protected constructor used by dynamic creation
    virtual ~Cview2();
public:
    virtual void OnDraw(CDC* pDC);      // overridden to draw this view
#ifdef _DEBUG
    virtual void AssertValid() const;
#ifndef _WIN32_WCE
    virtual void Dump(CDumpContext& dc) const;
#endif
#endif

protected:
    DECLARE_MESSAGE_MAP()
};
```

Листинг 14.10. Файл view2.cpp (определение класса представления графических данных)

```
// view2.cpp : implementation file
//
#include "stdafx.h"
#include "pr14.h"
#include "view2.h"

// Cview2
IMPLEMENT_DYNCREATE(Cview2, CView)

Cview2::Cview2()
{
}
Cview2::~Cview2()
{
}
BEGIN_MESSAGE_MAP(Cview2, CView)
END_MESSAGE_MAP()

// Cview2 drawing
void Cview2::OnDraw(CDC* pDC)
{
   CDocument* pDoc = GetDocument();
   // TODO: add draw code here
}

// Cview2 diagnostics
#ifdef _DEBUG
void Cview2::AssertValid() const
{
   CView::AssertValid();
}
#ifndef _WIN32_WCE
void Cview2::Dump(CDumpContext& dc) const
```

```
{
    CView::Dump(dc);
}
#endif
#endif //_DEBUG

// Cview2 message handlers
```

Теперь надо добавить в ресурс таблицы строк **String Table** новую строку (рис. 4.9):

```
IDR_pr14TYPE2   101
   \nNew grf\nselect grf\nGraphic Files (*.grf)\n.grf\ndoc2.Document\ndoc2.D
ocument
```

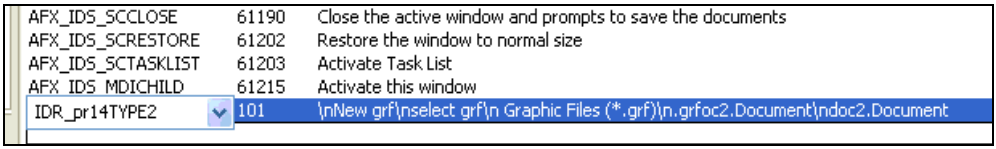


Рис. 14.9. Добавление строкового ресурса для работы с графическими файлами

После этого надо сохранить файл ресурса, тогда эта строка переместится наверх (строковые ресурсы сортируются по возрастанию идентификаторов). Изменения в файле приведены в листинге 14.11.

Листинг 14.11. Изменения в файле Resource.h при задании названий окон и фильтра для графических файлов

```
// ...
#define IDD_ABOUTBOX                100
#define IDP_OLE_INIT_FAILED         100
#define IDR_pr14TYPE2               101
#define IDR_MAINFRAME               128
#define IDR_pr14TYPE                129
// ...
```

Теперь надо добавить новый тип документа (с представлением) в шаблон документов. Изменения в файле приведены в листинге 14.12.

Листинг 14.12. Изменения в файле pr14.cpp для добавления второго шаблона документа

```
// ...
#include "pr14View.h"
#include "doc2.h"
#include "view2.h"
// ...
BOOL Cpr14App::InitInstance()
{
   // ...
CMultiDocTemplate* pDocTemplate;
   pDocTemplate = new CMultiDocTemplate(IDR_pr14TYPE,
      RUNTIME_CLASS(Cpr14Doc),      // Текстовый документ
      RUNTIME_CLASS(CChildFrame),   // Стандартный дочерний фрейм MDI
      RUNTIME_CLASS(Cpr14View));    // Представление текстового документа
   if(!pDocTemplate)
      return FALSE;
   AddDocTemplate(pDocTemplate);    // Добавить в шаблон

   pDocTemplate = new CMultiDocTemplate(IDR_pr14TYPE2,
      RUNTIME_CLASS(Cdoc2),        // Графический документ
      RUNTIME_CLASS(CChildFrame), // Стандартный дочерний фрейм MDI
      RUNTIME_CLASS(Cview2));      // Представление графического документа
   if(!pDocTemplate)
      return FALSE;
   AddDocTemplate(pDocTemplate);  // Добавить новый тип в шаблон
   // ...
}
```

Теперь при запуске программы будет выдаваться окно со списком возможных типов документов (рис. 14.10). Если нажать кнопку **OK**, то будет открыт новый документ выбранного типа. Если нажать кнопку **Cancel**, то будет открыто только главное окно приложения (без дочерних окон). Такое же окно будет появляться, если выбрать пункт меню **File | New**.

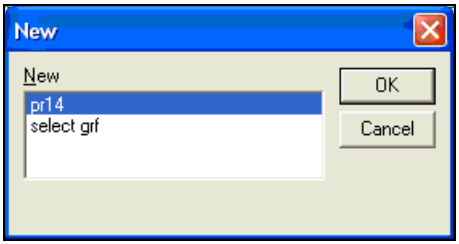
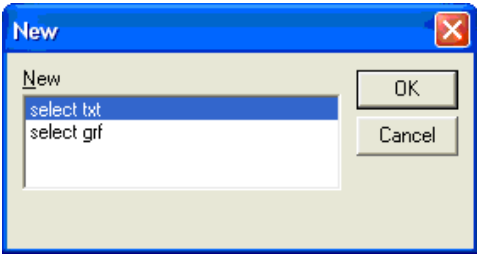


Рис. 14.10. Окно выбора типа открываемого документа

Теперь исправим в таблице строк шаблон для текстовых документов. Для этого надо изменить строку `IDR_pr14TYPE` (текст, выделенный полужирным шрифтом):

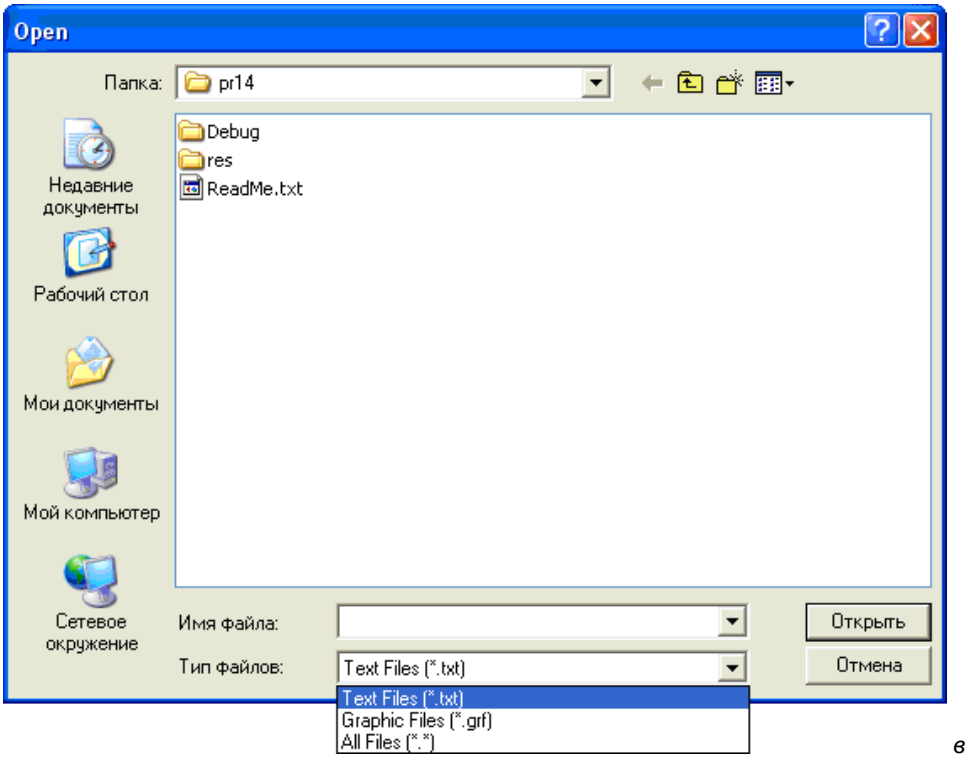
```
IDR_pr14TYPE  129
   \nNew txt\npr14\npr14 Files (*.txt)\n.txt\npr14.Document\npr14.Document
```



а

б

Рис. 14.11. Выбор типа создаваемого документа (*а*), созданные окна текстового и графического документов (*б*)



в

Рис. 14.11. Виды фильтров для открытия файлов (*в*)

на:

```
IDR_pr14TYPE   129
   \nNew txt\nselect txt\nText Files (*.txt)\n.txt\npr14.Document\npr14.Docu
ment
```

Результат работы программы показан на рис. 14.11.

14.1.3. Рисование в графическом окне

Сделаем так, чтобы при нажатии правой кнопки мыши в графическом окне в месте нажатия появлялся символ `'X'`. Для этого надо добавить в класс представления графического окна `Cview2` обработку общения `WM_RBUTTONUP` *(см. разд. 11.1.1)*. Также для возможности чтения/записи графических файлов надо добавить класс `CPointX` (аналогично классу `CLine` в проекте **pr12**). Изменения в файлах приведены в листингах 14.13—14.16.

Листинг 14.13. Изменения в файле doc2.h для возможности рисования в графическом окне

```
#pragma once

// Cdoc2 document
class CPointX : public CObject
{
public:
   CPoint point;                    // Координаты точки

   DECLARE_SERIAL (CPointX)         // Для работы с архивом
public:
   CPointX(CPoint p);
   virtual ~CPointX();

   CPointX(void);
public:
   virtual void Serialize(CArchive& ar);
};

class Cdoc2 : public CDocument
{
// ...
public:
   CTypedPtrArray<CObArray, CPointX*> arpoint;      // Массив точек
public:
   virtual void DeleteContents();
};
```

Листинг 14.14. Изменения в файле doc2.cpp для возможности рисования в графическом окне

```
// ...
void Cdoc2::Serialize(CArchive& ar)
{
```

```
   if(ar.IsStoring())
   {
      // TODO: add storing code here
      this->arpoint.Serialize(ar);
   }
   else
   {
      // TODO: add loading code here
      this->arpoint.Serialize(ar);
   }
}
// ...
void Cdoc2::DeleteContents()
{
   // TODO: Add your specialized code here and/or call the base class
   int sz = (int)this->arpoint.GetSize();
   for(int i = 0; i < sz; i++)
   {
      delete this->arpoint.GetAt(i);   // Удалить объект
   }
   this->arpoint.RemoveAll();          // Удалить все указатели на объект

   CDocument::DeleteContents();
}

IMPLEMENT_SERIAL(CPointX, CObject, 1)        // Для работы с архивом
CPointX::CPointX(CPoint p):point(p)          // Конструктор
{
}
CPointX::~CPointX()
{
}
CPointX::CPointX(void)
{
}
```

```
void CPointX::Serialize(CArchive& ar)
{
   if(ar.IsStoring())
   {  // storing code
      ar << point.x << point.y;        // Занесение данных в архив
   }                                   // для записи в файл
   else
   {  // loading code
      ar >> point.x >> point.y;        // Получение данных из архива
   }                                   // после чтения из файла
}
```

Листинг 14.15. Изменения в файле view2.h для возможности рисования в графическом окне

```
// ...
class Cview2 : public CView
{
// ...
public:
   afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
};
```

Листинг 14.16. Изменения в файле view2.cpp для возможности рисования в графическом окне

```
// ...
#include "doc2.h"
// ...
BEGIN_MESSAGE_MAP(Cview2, CView)
   ON_WM_RBUTTONUP()
END_MESSAGE_MAP()

void Cview2::OnDraw(CDC* pDC)
{
```

```
    CDocument* pDoc = GetDocument();
    // TODO: add draw code here
    Cdoc2* doc2 = (Cdoc2* )this->GetDocument();
    // Получить размер массива линий
    int sz = (int)doc2->arpoint.GetSize();
    for(int i = 0; i < sz; i++)
    {
       // Рисование текущего символа 'X'
       pDC->TextOutA(doc2->arpoint.GetAt(i)->point.x,
                     doc2->arpoint.GetAt(i)->point.y, "X");
    }
}
// ...
void Cview2::OnRButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    Cdoc2* doc2 = (Cdoc2* )this->GetDocument();
    CPointX *px = new CPointX(point);       // Создать новую точку
    doc2->arpoint.Add(px);                  // Добавить ее в массив
    doc2->SetModifiedFlag();                // Установить флаг изменений
    this->RedrawWindow();                   // Перерисовать окно

    CView::OnRButtonUp(nFlags, point);
}
```

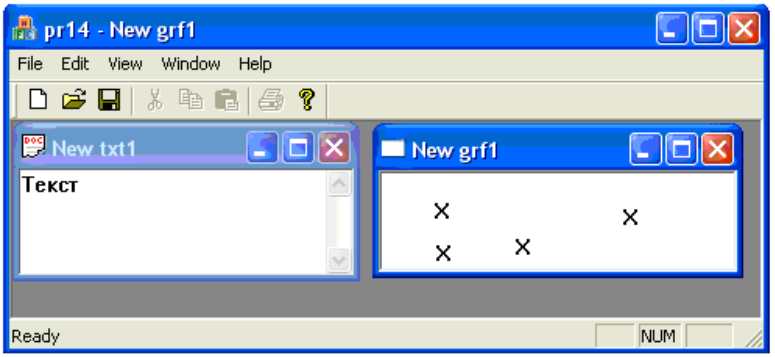


Рис. 14.12. Обработка нажатия правой кнопки мыши в графическом окне

Функция получения указателя на документ, связанный с окном представления, определена как:

```
CDocument* CView::GetDocument() const;     // NULL - если нет документа,
                                           // связанного с представлением
```

Результат работы программы показан на рис. 14.12.

14.1.4. Обмен данными между документами

Сделаем так, чтобы содержимое текущего текстового окна (с координатами точек в формате `X;Y`) можно было отобразить в графическом окне, а содержимое текущего графического окна — в текстовом окне в виде значений координат. Для этого надо в ресурс меню `IDR_pr14TYPE` (а не в `IDR_MAINFRAME`, т. к. это меню должно работать, только если есть хотя бы одно открытое окно) добавить новое меню **format** с подменю **TxtToGrf** и **GrfToTxt** (рис. 14.13).

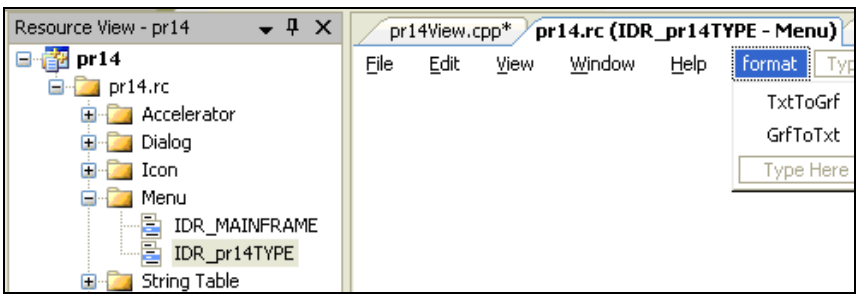


Рис. 14.13. Меню для преобразования текстового файла в графический и наоборот

Обработку меню **format | TxtToGrf** сделаем в классе представления текстового документа `Cpr14View`, т. к. это меню должно активизироваться, только если текущим (активным) окном является текстовое окно (рис. 14.14). Обработку **format | GrfToTx** сделаем в классе представления графического документа `Cview2`, т. к. это меню должно активизироваться только если текущим является графическое окно (рис. 14.15). Изменения в файлах приведены в листингах 14.17—14.21.

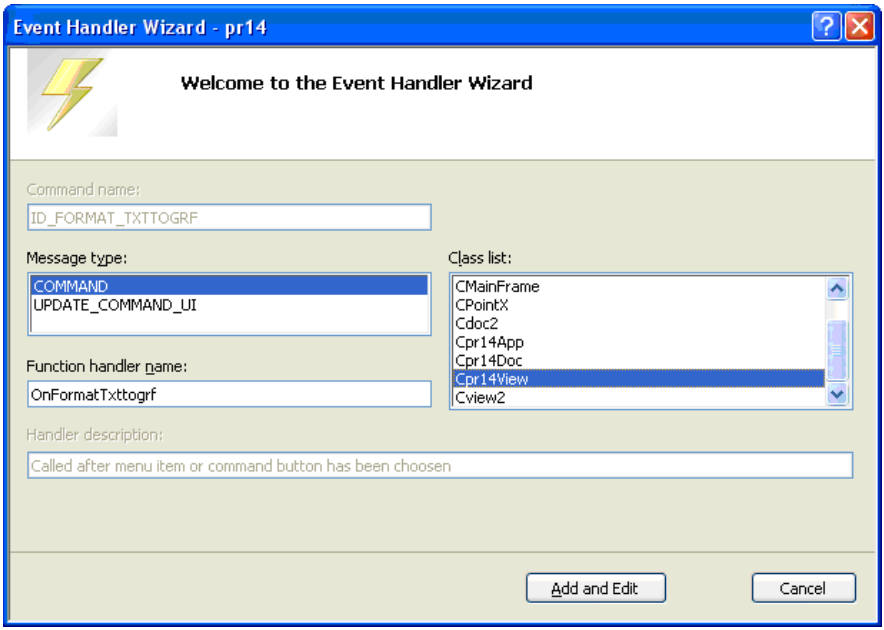


Рис. 14.14. Функция обработки меню преобразования текстового файла в графический

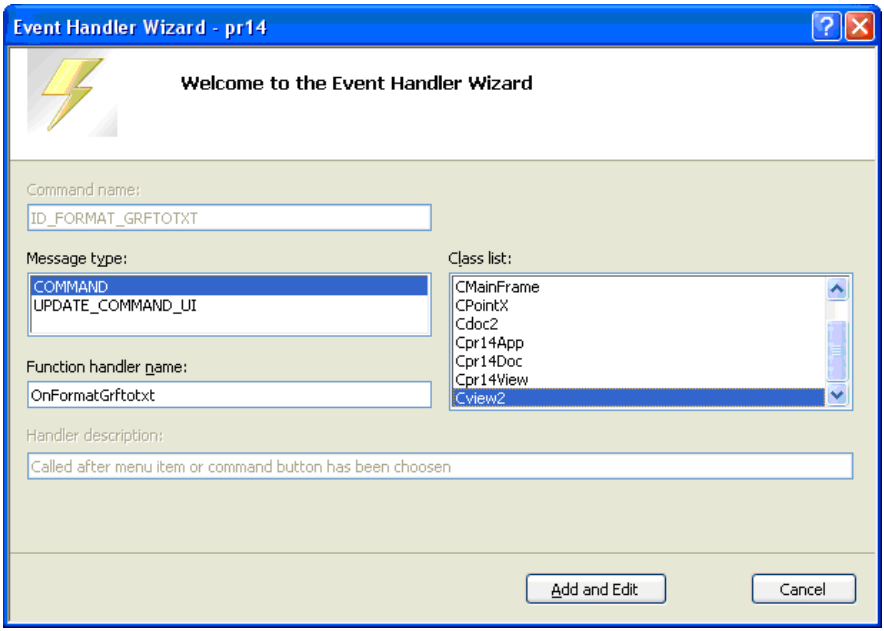


Рис. 14.15. Функция обработки меню преобразования графического файла в текстовый

Листинг 14.17. Изменения в файле Resource.h для обработки меню преобразования текстового файла в графический и наоборот

```
// ...
#define ID_FORMAT_TXTTOGRF              32771
#define ID_FORMAT_GRFTOTXT              32772
// ...
```

Листинг 14.18. Изменения в файле pr14View.h для обработки меню преобразования текстового файла в графический

```
// ...
class Cpr14View : public CEditView
{
// ...
public:
   afx_msg void OnFormatTxttogrf();
};
```

Листинг 14.19. Изменения в файле pr14View.cpp для обработки меню преобразования текстового файла в графический

```
// ...
BEGIN_MESSAGE_MAP(Cpr14View, CEditView)
   ON_COMMAND(ID_FORMAT_TXTTOGRF, &Cpr14View::OnFormatTxttogrf)
END_MESSAGE_MAP()
// ...
void Cpr14View::OnFormatTxttogrf()
{
   // TODO: Add your command handler code here
}
```

Листинг 14.20. Изменения в файле view2.h для обработки меню преобразования графического файла в текстовый

```
// ...
class Cview2 : public CView
```

```
{
// ...
public:
   afx_msg void OnFormatGrftotxt();
};
```

Листинг 14.21. Изменения в файле view2.cpp для обработки меню преобразования графического файла в текстовый

```
// ...
BEGIN_MESSAGE_MAP(Cview2, CView)
   ON_WM_RBUTTONUP()
   ON_COMMAND(ID_FORMAT_GRFTOTXT, &Cview2::OnFormatGrftotxt)
END_MESSAGE_MAP()
// ...
void Cview2::OnFormatGrftotxt()
{
   // TODO: Add your command handler code here
}
```

Преобразование текстового документа в графический

Сделаем так, чтобы из окна текстового файла считывались координаты точек, и эти точки отображались в окне графического файла в виде символов `'X'`. Изменения в файле приведены в листинге 14.22.

Листинг 14.22. Изменения в файле pr14View.cpp для отображения координат точек из текстового файла в окне графического файла

```
// ...
#include "doc2.h"
#include "view2.h"
#include "MainFrm.h"
#include "ChildFrm.h"
// ...
void Cpr14View::OnFormatTxttogrf()
```

```
{
   // TODO: Add your command handler code here

   // Запрос на сохранение всех изменных файлов
   theApp.m_pDocManager->SaveAllModified();

   // Получить позицию первого шаблона документа
   POSITION pos = theApp.GetFirstDocTemplatePosition();
   // Получить первый шаблон документа (для txt) и позицию следующего
   // шаблона (в pos), документы в шаблоне располагаются в том порядке,
   // в каком были добавлены
   CMultiDocTemplate* doctempl1 =
                  (CMultiDocTemplate* )theApp.GetNextDocTemplate(pos);
   // Получить второй шаблон документа (для grf)
   CMultiDocTemplate* doctempl2 =
                  (CMultiDocTemplate* )theApp.GetNextDocTemplate(pos);

   Cpr14Doc *doc1 = this->GetDocument();     // Указатель на txt документ
   CString s1 = doc1->GetTitle();            // Получить заголовой окна
                                             // текстового документа

   // Формирование заголовка графического окна
   int indtz = s1.Find('.');          // Найти в заголовке '.' (если это
                                      // открытый файл)
   CString s2,                        // Заголовок нового grf-окна
           s3;
   if(indtz != -1)                    // Если было расширение (нашли '.')
      s2 = s1.Left(indtz);            // Получить имя файла без расширения
   else
      s2 = s1;
   s2 += " to Grf - ";
   static int num=1;                  // Счетчик новых окон
   s3.Format("%d", num);
   s2 += s3;
```

```
Cdoc2 *doc2;                        // Указатель на последний созданный
                                    // документ grf
int fwin=0;                         // Новое окно grf уже создавалось
                                    // (0 - нет, 1-да)

// Получение указателя на графический документ, куда будут выводиться
// данные
// Получить позицию первого документа grf второго шаблона
pos = doctempl2->GetFirstDocPosition();
while(pos)                         // Если новое окно grf уже создавалось
{
   fwin = 1;
   // Поиск последнего созданного документа grf
   doc2 = (Cdoc2 *)doctempl2->GetNextDoc(pos);
}
Cdoc2 *newdoc2;                    // Указатель на новый документ grf
// Если окно grf еще не создавалось или создавалось, но пользователь
// попросил создать еще одно новое графическое окно
if(!fwin || (fwin && ::AfxMessageBox("Создать GRF новое окно вывода?",
                                MB_YESNO | MB_ICONQUESTION) == IDYES))
{
   // Создать новое окно grf
   newdoc2 = (Cdoc2 *)doctempl2->OpenDocumentFile(NULL);
   newdoc2->SetTitle(s2);          // Задать его заголовок
   num++;
   doc2 = newdoc2;                 // Новое окно становится последним
}

// Получить позицию первого окна представления grf-документа
pos = doc2->GetFirstViewPosition();
// Получить указатель на представление grf-документа
Cview2 *view2 = (Cview2 *)doc2->GetNextView(pos);
// Передача текстовых данных в графический документ
doc2->DeleteContents();      // Очистить массив точек второго документа
CString sx, sy;              // Строки с координатами X, Y
```

```
int dx, dy;                  // Координаты X, Y
CPoint point;                // Координаты X, Y

// Получить ссылку на управляющий элемент редактирования txt
CEdit &editcontrl = this->GetEditCtrl();
CString es;          // Строка открытого текущего текстового документа
int sz;              // Ее длина
int i = 0;

// Количество строк в окне редактирования (включая пустые строки)
int linecount = editcontrl.GetLineCount();
// Просмотр всего текстового окна
for(i=0;i < linecount;i++)
{
   // Длина текущей строки в окне редактирования
   int len = editcontrl.LineLength(editcontrl.LineIndex(i));
   // Считать текущую строку
   if(!editcontrl.GetLine(i, es.GetBuffer(len), len))
   {
      // Если она пустая
      break;
   }
   es.ReleaseBuffer(len);              // Добавить "конец строки"
   sz = es.GetLength();                // Длина строки
   indtz = es.Find(';');               // Найти ';'
   sx = es.Left(indtz);                // Взять то, что слева от ';'
                                       // (первое число X)
   sy = es.Right(sz-indtz-1);          // Взять то, что справа от ';'
                                       // (второе число Y)
   // Преобразовать строки в числа
   dx = atoi(sx.GetBuffer());
   dy = atoi(sy.GetBuffer());
   point.x = dx;
   point.y = dy;
```

```
      CPointX *px = new CPointX(point);     // Создать новую точку
      doc2->arpoint.Add(px);                // Добавить ее в массив
   }

   view2->RedrawWindow();                  // Перерисовать графическое окно
   doc2->SetModifiedFlag();                // Установить флаг изменений

   // Делаем полученное графическое окно активным (расположенным поверх
   // всех окон)
   // Указатель на главное фреймовой окно
   CMainFrame *mfr = (CMainFrame *)theApp.m_pMainWnd;
   CChildFrame *frch;              // Указатель на дочернее фреймовое окно
   // Получить родительский фрейм графического представления
   frch = (CChildFrame *)view2->GetParentFrame();
   mfr->MDIActivate(frch);          // Сделать его активным
}
```

С целью уменьшения кода точки хранятся в формате `X;Y`, и здесь не делается никаких проверок (существования окон, чтения данных и т. п.), которые в реальной программе, конечно, надо делать. Не приведенные здесь функции можно найти в *гл. 10—12*.

Сохранение всех модифицированных документов выполняется функцией:

```
virtual BOOL CDocTemplate::SaveAllModified();     // 0 - ошибка
```

Поиск подстроки или символа в строке выполняется функцией:

```
int CStringT::Find(             // Индекс в строке, с которого начинается
                                // найденная подстрока, или -1 – если она
                                // не найдена
   PCXSTR pszSub,               // Подстрока для поиска
   int iStart=0) const throw(); // Индекс в строке, с которого надо
                                // начать поиск
int CStringT::Find(             // Индекс в строке, где найден символ,
                                // или -1
   XCHAR ch,                    // Символ для поиска
   int iStart=0) const throw();
```

Получить часть начала строки можно с помощью функции:

```
CStringT CStringT::Left(        // Полученная подстрока
   int nCount) const;           // Количество берущихся символов
                                // от начала строки
```

Получить часть конца строки:

```
CStringT CStringT::Right(       // Полученная подстрока
   int nCount) const;           // Количество берущихся символов от конца
                                // строки
```

Функция открытия файла (она является чисто виртуальной и переопределена в классах `CSingleDocTemplate:public CDocTemplate` и `CMultiDocTemplate:public CDocTemplate`):

```
virtual CDocument* CDocTemplate::OpenDocumentFile(  // Указатель на
                                    // открытый документ или
                                    // NULL - при ошибке
   LPCTSTR lpszPathName,            // Имя файла с нужным документом.
   BOOL bMakeVisible = TRUE) = 0;   // Определяет, что окно с документом
                                    // должно быть видимым
```

Если задать `lpszPathName = NULL`, то будет создан новый документ.

Получить позицию первого окна представления, связанного с документом, можно функцией:

```
virtual POSITION CDocument::GetFirstViewPosition() const;
```

Получить указатель на окно представления по его позиции в списке можно функцией:

```
virtual CView* CDocument::GetNextView( // Указатель на окно представления
   POSITION& rPosition) const;         // Его позиция в списке
```

После получения указателя на окно представления в параметр `rPosition` автоматически записывается позиция следующего окна представления в списке или 0, если окон больше нет.

Получить количество строк (включая пустые) в окне редактирования можно с помощью функции:

```
int CEdit::GetLineCount() const;
```

Получить длину строки в текстовом окне:

```
int CEdit::LineLength(      // Длина строки, индекс которой задан в nLine
   int nLine = -1) const;   // Индекс проверяемой строки
```

Если `nLine = -1`, то возвращается длина текущей строки.

Получить индекс первого символа строки от начала буфера можно функцией:

```
int CEdit::LineIndex(       // Индекс строки, определенной в nLine
   int nLine = -1) const;   // Индекс проверяемой строки
```

Если `nLine = -1`, то возвращается индекс текущей строки.

Получить строку из текстового поля:

```
int CEdit::GetLine(         // Число реально скопированных байт текста
   int nIndex,              // Индекс строки, которую надо получить
   LPTSTR lpszBuffer) const; // Буфер, в который будет скопирована строка
int CEdit::GetLine(
   int nIndex,
   LPTSTR lpszBuffer,
   int nMaxLength) const;    // Число байт, которые будут скопированы в
                             // lpszBuffer
```

Получить указатель на символьный буфер:

```
PXSTR CSimpleStringT::GetBuffer(  // Указатель на символьный буфер
   int nMinBufferLength);         // Минимальный размер буфера в символах
PXSTR CSimpleStringT::GetBuffer();
```

Получить указатель на родительское (фреймовое) окно:

```
CFrameWnd* CWnd::GetParentFrame() const;   // Указатель на фреймовое окно
                                           // или NULL - если его нет
```

Сделать активным дочернее окно:

```
void CMDIFrameWnd::MDIActivate(
   CWnd* pWndActivate);            // Указатель на активизируемое окно
```

Преобразование графического документа в текстовый

Сделаем так, чтобы из окна графического документа узнавались координаты отображенных символов `'X'` и записывались в окно текстового документа. Изменения в файле приведены в листинге 14.23.

Листинг 14.23. Изменения в файле view2.cpp для преобразования данных из окна графического документа в окно текстового документа

```
// ...
#include "doc2.h"
#include "pr14Doc.h"
#include "pr14View.h"
#include "MainFrm.h"
#include "ChildFrm.h"
// ...
void Cview2::OnFormatGrftotxt()
{
   // TODO: Add your command handler code here

   // Запрос на сохранение всех измененных файлов
   theApp.m_pDocManager->SaveAllModified();
   // Получить позицию первого шаблона документа
   POSITION pos = theApp.GetFirstDocTemplatePosition();
   // Получить первый шаблон документа (для txt)
   CMultiDocTemplate* doctempl1 =
                    (CMultiDocTemplate* )theApp.GetNextDocTemplate(pos);

   // Указатель на документ grf
   Cdoc2 *doc2 = (Cdoc2 *)this->GetDocument();
   // Получить заголовок окна документа grf
   CString s1 = doc2->GetTitle();

   // Формирование заголовка текстового окна
   int indtz = s1.Find('.');    // Найти в заголовке '.' (если это
                                // открытый файл)
   CString s2,                  // Заголовок нового txt окна
           s3;
   if(indtz != -1)              // Если было расширение
      s2 = s1.Left(indtz);      // Получить имя файла без расширения
```

```
else
   s2 = s1;
s2 += " to Txt - ";
static int num=1;            // Счетчик новых окон
s3.Format("%d", num);
s2 += s3;
Cpr14Doc *doc1;              // Указатель на последний созданный
                             // документ txt
int fwin=0;                  // Новое окно grf уже создавалось
                             // (0 - нет, 1-да)

// Получение указателя на текстовый документ, куда будут выводиться
// данные
// Получить позицию первого документа txt первого шаблона
pos = doctempl1->GetFirstDocPosition();
while(pos)                   // Если новое окно txt уже создавалось
{
   fwin = 1;
   // Поиск последнего созданного документа txt
   doc1 = (Cpr14Doc *)doctempl1->GetNextDoc(pos);
}

Cpr14Doc *newdoc1;           // Указатель на новый документ txt

// Если окно *.lin еще не создавалось или создавалось, но пользователь
// попросил создать еще одно новое
if(!fwin || (fwin && ::AfxMessageBox("Создать TXT новое окно вывода?",
                              MB_YESNO | MB_ICONQUESTION ) == IDYES))
{
   // Создать новое окно txt
   newdoc1 = (Cpr14Doc *)doctempl1->OpenDocumentFile(NULL);
   newdoc1->SetTitle(s2);     // Задать его заголовок
   num++;
   doc1 = newdoc1;            // Новое окно становится последним
}
```

```
   // Получить позицию первого окна представления документа txt
   pos = doc1->GetFirstViewPosition();
   // Получить указатель на представление документа txt
   Cpr14View *view1 = (Cpr14View *)doc1->GetNextView(pos);
   // Передача графических данных в текстовый документ
   CString es,          // Будущее (полученное) содержимое текстового окна
           ss;          // Строка с координатами текущей точки
   int sz = (int)doc2->arpoint.GetSize();   // Количество точек в массиве
   for(int i = 0; i < sz; i++)
   {
      // Сформировать строку с координатами
      ss.Format("%d;%d\r\n", doc2->arpoint.GetAt(i)->point.x,
                             doc2->arpoint.GetAt(i)->point.y);
      es += ss;
   }
   // Получить ссылку на управляющий элемент редактирования txt
   CEdit &editcontrl = view1->GetEditCtrl();
   editcontrl.SetWindowTextA(es);          // Вывести текст в окно
   doc1->SetModifiedFlag();                // Установить флаг изменений

   // Делаем полученное текстовое окно активным (расположенным поверх
   ///всех окон)
   // Указатель на главное фреймовое окно
   CMainFrame *mfr = (CMainFrame *)theApp.m_pMainWnd;
   CChildFrame *frch;            // Указатель на дочернее фреймовое окно
   // Получить родительский фрейм текстового представления
   frch = (CChildFrame *)view1->GetParentFrame();
   mfr->MDIActivate(frch);       // Сделать его активным
}
```

Результат работы программы показан на рис. 14.16 и 14.17.

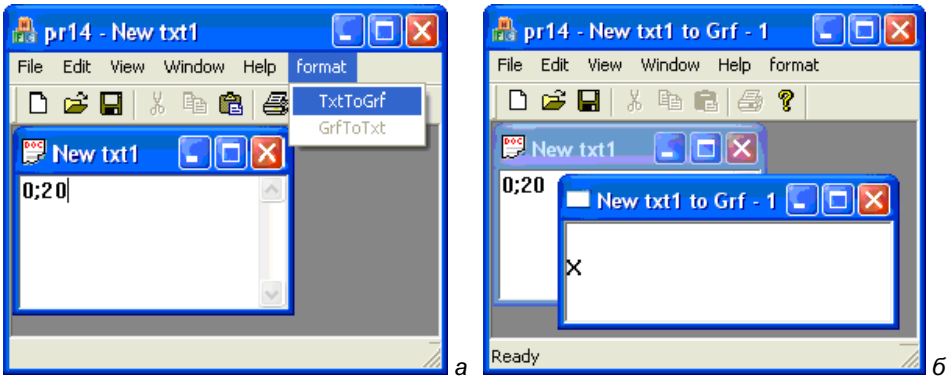


Рис. 14.16. Создание текстового файла (*а*)
и преобразование его в графический файл (*б*)

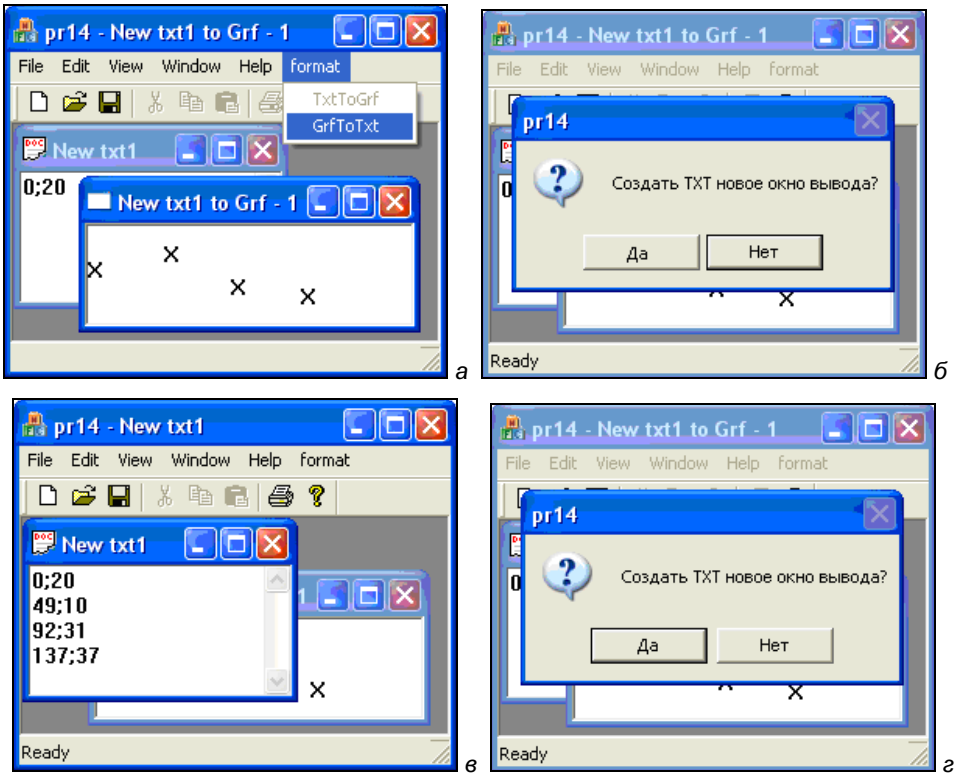
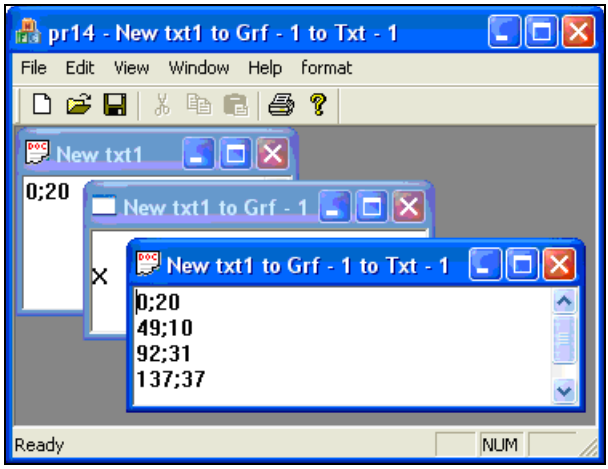


Рис. 14.17. Преобразование графического файла (*а*)
в уже открытый текстовый файл (*б*, *в*) или в новый текстовый файл (*г*)



д

Рис. 14.17. Преобразование графического файла в новый текстовый файл (*д*)

14.1.5. Некоторые полезные функции для работы с дочерними окнами

Функции управления из главного фрейма дочерними окнами

Получить указатель на текущее активное дочернее окно можно с помощью функции:

```
CMDIChildWnd* CMDIFrameWnd::MDIGetActive(    // Указатель на активное окно
   BOOL* pbMaximized = NULL) const;          // Дополнительно возвращаемая
                                             // информация о том,
                                             // свернуто окно или нет
```

Упорядочить все свернутые дочерние окна можно функцией:

```
void CMDIFrameWnd::MDIIconArrange();
```

Развернуть до максимума дочернее окно:

```
void CMDIFrameWnd::MDIMaximize(              // Указатель на дочернее окно
   CWnd* pWnd);
```

Активизировать следующее дочернее окно:

```
void CMDIFrameWnd::MDINext();
```

Активизировать предыдущее дочернее окно:

```
void CMDIFrameWnd::MDIPrev();
```

Восстановить исходные размеры дочернего окна:

```
void CMDIFrameWnd::MDIRestore(
   CWnd* pWnd);                          // Указатель на дочернее окно
```

Расположить дочерние окна мозаикой (не перекрывая друг друга):

```
void CMDIFrameWnd::MDITile();            // По вертикали
void CMDIFrameWnd::MDITile(              // По заданному типу nType
   int nType);
```

Виды мозаики (`nType`) могут быть такими :

- `MDITILE_HORIZONTAL` — по горизонтали;
- `MDITILE_VERTICAL` — по вертикали;
- `MDITILE_SKIPDISABLED` — не отображать блокированные окна.

Расположить дочерние окна каскадом можно с помощью функции:

```
void CMDIFrameWnd::MDICascade();         // Все окна
```

или

```
void CMDIFrameWnd::MDICascade(           // Все, кроме блокированных
   int nType);                           // MDITILE_SKIPDISABLED
```

Функции дочерних окон

Сделать окно активным можно с помощью функции:

```
void CMDIChildWnd::MDIActivate();
```

Разрушить (закрыть) окно:

```
void CMDIChildWnd::MDIDestroy();
```

Развернуть окно:

```
void CMDIChildWnd::MDIMaximize();
```

Восстановить размеры окна:

```
void CMDIChildWnd::MDIRestore();
```

Получить указатель на главное (родительское) окно фрейма:

```
CMDIFrameWnd* CMDIChildWnd::GetMDIFrame();
```

14.2. Листинги программы

Здесь приведены только те листинги, коды которых были изменены относительно изначально построенных мастером.

Листинг 14.24. Файл Resource.h (идентификаторы ресурсов приложения)

```
// ...
#define IDD_ABOUTBOX            100
#define IDP_OLE_INIT_FAILED     100
#define IDR_pr14TYPE2           101
#define IDR_MAINFRAME           128
#define IDR_pr14TYPE            129
#define ID_FORMAT_TXTTOGRF      32771
#define ID_FORMAT_GRFTOTXT      32772
// ...
```

Листинг 14.25. Файл pr14.cpp (определение класса приложения)

```
// ...
#include "stdafx.h"
#include "pr14.h"
#include "MainFrm.h"
#include "ChildFrm.h"
#include "pr14Doc.h"
#include "pr14View.h"
#include "doc2.h"
#include "view2.h"
// ...
BOOL Cpr14App::InitInstance()
{
   // ...
   CMultiDocTemplate* pDocTemplate;
   pDocTemplate = new CMultiDocTemplate(IDR_pr14TYPE,
      RUNTIME_CLASS(Cpr14Doc),     // Текстовый документ
```

```
      RUNTIME_CLASS(CChildFrame),  // Стандартный дочерний фрейм MDI
      RUNTIME_CLASS(Cpr14View));   // Представление текстового документа
   if(!pDocTemplate)
      return FALSE;
   AddDocTemplate(pDocTemplate);   // Добавить в шаблон

   pDocTemplate = new CMultiDocTemplate(IDR_pr14TYPE2,
      RUNTIME_CLASS(Cdoc2),        // Графический документ
      RUNTIME_CLASS(CChildFrame), // Стандартный дочерний фрейм MDI
      RUNTIME_CLASS(Cview2));      // Представление графического документа
   if(!pDocTemplate)
      return FALSE;
   AddDocTemplate(pDocTemplate);  // Добавить новый тип в шаблон
   // ...
```

Листинг 14.26. Файл pr14View.h (объявление класса представления графического документа)

```
// ...
class Cpr14View : public CEditView
{
// ...
public:
   afx_msg void OnFormatTxttogrf();
};
```

Листинг 14.27. Файл pr14View.cpp (определение класса представления графического документа)

```
// ...
#include "stdafx.h"
#include "pr14.h"
#include "pr14Doc.h"
#include "pr14View.h"
#include "doc2.h"
```

```
#include "view2.h"
#include "MainFrm.h"
#include "ChildFrm.h"
// ...
BEGIN_MESSAGE_MAP(Cpr14View, CEditView)
   ON_COMMAND(ID_FORMAT_TXTTOGRF, &Cpr14View::OnFormatTxttogrf)
END_MESSAGE_MAP()
// ...
void Cpr14View::OnFormatTxttogrf()
{
   // TODO: Add your command handler code here

   // Запрос на сохранение всех изменных файлов
   theApp.m_pDocManager->SaveAllModified();

   // Получить позицию первого шаблона документа
   POSITION pos = theApp.GetFirstDocTemplatePosition();
   // Получить первый шаблон документа (для txt) и позицию следующего
   // шаблона (в pos), документы в шаблоне располагаются в том порядке, в
   ///каком были добавлены
   CMultiDocTemplate* doctempl1 =
                  (CMultiDocTemplate* )theApp.GetNextDocTemplate(pos);
   // Получить второй шаблон документа (для grf)
   CMultiDocTemplate* doctempl2 =
                  (CMultiDocTemplate* )theApp.GetNextDocTemplate(pos);

   Cpr14Doc *doc1 = this->GetDocument();     // Указатель на txt документ
   CString s1 = doc1->GetTitle();            // Получить заголовок окна
                                             // текстового документа

   // Формирование заголовка графического окна
   int indtz = s1.Find('.');          // Найти в заголовке '.' (если это
                                      // открытый файл)
   CString s2,                        // Заголовок нового grf-окна
           s3;
```

```
if(indtz != -1)                    // Если было расширение (нашли '.')
   s2 = s1.Left(indtz);            // Получить имя файла без расширения
else
   s2 = s1;
s2 += " to Grf - ";
static int num=1;                  // Счетчик новых окон
s3.Format("%d", num);
s2 += s3;

Cdoc2 *doc2;                       // Указатель на последний созданный
                                   // документ grf
int fwin=0;                        // Новое окно grf уже создавалось
                                   // (0 - нет, 1-да)

// Получение указателя на графический документ, куда будут выводиться
// данные
// Получить позицию первого документа grf второго шаблона
pos = doctempl2->GetFirstDocPosition();
while(pos)                         // Если новое окно grf уже создавалось
{
   fwin = 1;
   // Поиск последнего созданного документа grf
   doc2 = (Cdoc2 *)doctempl2->GetNextDoc(pos);
}
Cdoc2 *newdoc2;                    // Указатель на новый документ grf
// Если окно grf еще не создавалось или создавалось, но пользователь
// попросил создать еще одно новое графическое окно
if(!fwin || (fwin && ::AfxMessageBox("Создать GRF новое окно вывода?",
                                MB_YESNO | MB_ICONQUESTION) == IDYES))
{
   // Создать новое окно grf
   newdoc2 = (Cdoc2 *)doctempl2->OpenDocumentFile(NULL);
   newdoc2->SetTitle(s2);          // Задать его заголовок
   num++;
```

```
   doc2 = newdoc2;                  // Новое окно становится последним
}

// Получить позицию первого окна представления grf-документа
pos = doc2->GetFirstViewPosition();
// Получить указатель на представление grf документа
Cview2 *view2 = (Cview2 *)doc2->GetNextView(pos);
// Передача текстовых данных в графический документ
doc2->DeleteContents();      // Очистить массив точек второго документа
CString sx, sy;              // Строки с координатами X, Y
int dx, dy;                  // Координаты X, Y
CPoint point;                // Координаты X, Y

// Получить ссылку на управляющий элемент редактирования txt
CEdit &editcontrl = this->GetEditCtrl();
CString es;          // Строка открытого текущего текстового документа
int sz;              // Ее длина
int i = 0;

// Количество строк в окне редактирования (включая пустые строки)
int linecount = editcontrl.GetLineCount();
// Просмотр всего текстового окна
for(i=0;i < linecount;i++)
{
   // Длина текущей строки в окне редактирования
   int len = editcontrl.LineLength(editcontrl.LineIndex(i));
   // Считать текущую строку
   if(!editcontrl.GetLine(i, es.GetBuffer(len), len))
   {
      // Если она пустая
      break;
   }
   es.ReleaseBuffer(len);              // Добавить "конец строки"
   sz = es.GetLength();                // Длина строки
   indtz = es.Find(';');               // Найти ';'
```

```
      sx = es.Left(indtz);                  // Взять то, что слева от ';'
                                            // (первое число X)
      sy = es.Right(sz-indtz-1);            // Взять то, что справа от ';'
                                            // (второе число Y)
      // Преобразовать строки в числа
      dx = atoi(sx.GetBuffer());
      dy = atoi(sy.GetBuffer());
      point.x = dx;
      point.y = dy;
      CPointX *px = new CPointX(point);     // Создать новую точку
      doc2->arpoint.Add(px);                // Добавить ее в массив
   }

   view2->RedrawWindow();                  // Перерисовать графическое окно
   doc2->SetModifiedFlag();                // Установить флаг изменений

   // Делаем полученное графическое окно активным (расположенным поверх
   // всех окон)
   // Указатель на главное фреймовое окно
   CMainFrame *mfr = (CMainFrame *)theApp.m_pMainWnd;
   CChildFrame *frch;              // Указатель на дочернее фреймовое окно
   // Получить родительский фрейм графического представления
   frch = (CChildFrame *)view2->GetParentFrame();
   mfr->MDIActivate(frch);          // Сделать его активным
}
```

Листинг 14.28. Файл doc2.h (объявление класса графического документа, был добавлен в проект)

```
// ...
class CPointX : public CObject
{
public:
   CPoint point;                    // Координаты точки
   DECLARE_SERIAL (CPointX)         // Для работы с архивом
```

```
public:
   CPointX(CPoint p);
   virtual ~CPointX();
   CPointX(void);
public:
   virtual void Serialize(CArchive& ar);
};
class Cdoc2 : public CDocument
{
// ...
public:
   CTypedPtrArray<CObArray, CPointX*> arpoint;   // Массив точек
public:
   virtual void DeleteContents();
};
```

Листинг 14.29. Файл doc2.cpp (определение класса графического документа, был добавлен в проект)

```
// ...
void Cdoc2::Serialize(CArchive& ar)
{
   if(ar.IsStoring())
   {
      // TODO: add storing code here
      this->arpoint.Serialize(ar);
   }
   else
   {
      // TODO: add loading code here
      this->arpoint.Serialize(ar);
   }
}
// ...
void Cdoc2::DeleteContents()
```

```
{
   // TODO: Add your specialized code here and/or call the base class
   int sz = (int)this->arpoint.GetSize();
   for(int i = 0; i < sz; i++)
   {
      delete this->arpoint.GetAt(i);    // Удалить объект
   }
   this->arpoint.RemoveAll();           // Удалить все указатели на объект

   CDocument::DeleteContents();
}

IMPLEMENT_SERIAL(CPointX, CObject, 1)  // Для работы с архивом
CPointX::CPointX(CPoint p):point(p)    // Конструктор
{
}
CPointX::~CPointX()
{
}
CPointX::CPointX(void)
{
}
void CPointX::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {   // storing code
      ar << point.x << point.y;      // Занесение данных в архив
   }                                 // для записи в файл
   else
   {   // loading code
      ar >> point.x >> point.y;      // Получение данных из архива
   }                                 // после чтения из файла
}
```

Листинг 14.30. Файл view2.h (объявление класса представления графического документа, был добавлен в проект)

```
// ...
class Cview2 : public CView
{
// ...
public:
   afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
public:
   afx_msg void OnFormatGrftotxt();
};
```

Листинг 14.31. Файл view2.cpp (определение класса представления графического документа, был добавлен в проект)

```
// ...
#include "stdafx.h"
#include "pr14.h"
#include "view2.h"
#include "doc2.h"
#include "pr14Doc.h"
#include "pr14View.h"
#include "MainFrm.h"
#include "ChildFrm.h"
// ...
BEGIN_MESSAGE_MAP(Cview2, CView)
   ON_WM_RBUTTONUP()
   ON_COMMAND(ID_FORMAT_GRFTOTXT, &Cview2::OnFormatGrftotxt)
END_MESSAGE_MAP()

// Cview2 drawing
void Cview2::OnDraw(CDC* pDC)
{
   CDocument* pDoc = GetDocument();
   // TODO: add draw code here
```

```
    Cdoc2* doc2 = (Cdoc2* )this->GetDocument();
    // Получить размер массива линий
    int sz = (int)doc2->arpoint.GetSize();
    for(int i = 0; i < sz; i++)
    {
       // Рисование текущего символа 'X'
       pDC->TextOutA(doc2->arpoint.GetAt(i)->point.x,
                     doc2->arpoint.GetAt(i)->point.y, "X");
    }
}
// ...
void Cview2::OnRButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    Cdoc2* doc2 = (Cdoc2* )this->GetDocument();
    CPointX *px = new CPointX(point);       // Создать новую точку
    doc2->arpoint.Add(px);                  // Добавить ее в массив
    doc2->SetModifiedFlag();                // Установить флаг изменений
    this->RedrawWindow();                   // Перерисовать окно

    CView::OnRButtonUp(nFlags, point);
}
void Cview2::OnFormatGrftotxt()
{
    // TODO: Add your command handler code here

    // Запрос на сохранение всех измененных файлов
    theApp.m_pDocManager->SaveAllModified();
    // Получить позицию первого шаблона документа
    POSITION pos = theApp.GetFirstDocTemplatePosition();
    // Получить первый шаблон документа (для txt)
    CMultiDocTemplate* doctempl1 =
                    (CMultiDocTemplate* )theApp.GetNextDocTemplate(pos);

    // Указатель на документ grf
    Cdoc2 *doc2 = (Cdoc2 *)this->GetDocument();
```

```
// Получить заголовок окна документа grf
CString s1 = doc2->GetTitle();

// Формирование заголовка текстового окна
int indtz = s1.Find('.');   // Найти в заголовке '.' (если это
                            // открытый файл)
CString s2,                 // Заголовок нового txt-окна
        s3;
if(indtz != -1)             // Если было расширение
   s2 = s1.Left(indtz);     // Получить имя файла без расширения
else
   s2 = s1;
s2 += " to Txt - ";
static int num=1;           // Счетчик новых окон
s3.Format("%d", num);
s2 += s3;

Cpr14Doc *doc1;             // Указатель на последний созданный
                            // документ txt
int fwin=0;                 // Новое окно grf уже создавалось
                            // (0 - нет, 1-да)

// Получение указателя на текстовый документ, куда будут выводиться
// данные
// Получить позицию первого документа txt первого шаблона
pos = doctempl1->GetFirstDocPosition();
while(pos)                  // Если новое окно txt уже создавалось
{
   fwin = 1;
   // Поиск последнего созданного докумета txt
   doc1 = (Cpr14Doc *)doctempl1->GetNextDoc(pos);
}

Cpr14Doc *newdoc1;          // Указатель на новый документ txt
```

```
// Если окно *.lin еще не создавалось или создавалось, но пользователь
// попросил создать еще одно новое
if(!fwin || (fwin && ::AfxMessageBox("Создать TXT новое окно вывода?",
                                MB_YESNO | MB_ICONQUESTION ) == IDYES))
{
   // Создать новое окно txt
   newdoc1 = (Cpr14Doc *)doctempl1->OpenDocumentFile(NULL);
   newdoc1->SetTitle(s2);      // Задать его заголовок
   num++;
   doc1 = newdoc1;             // Новое окно становится последним
}

// Получить позицию первого окна представления документа txt
pos = doc1->GetFirstViewPosition();
// Получить указатель на представление документа txt
Cpr14View *view1 = (Cpr14View *)doc1->GetNextView(pos);

// Передача графических данных в текстовый документ
CString es,          // Будущее (полученное) содержимое текстового окна
        ss;          // Строка с координатами текущей точки
int sz = (int)doc2->arpoint.GetSize();   // Количество точек в массиве
for(int i = 0; i < sz; i++)
{
   // Сформировать строку с координатами
   ss.Format("%d;%d\r\n", doc2->arpoint.GetAt(i)->point.x,
                          doc2->arpoint.GetAt(i)->point.y);
   es += ss;
}
// Получить ссылку на управляющий элемент редактирования txt
CEdit &editcontrl = view1->GetEditCtrl();
editcontrl.SetWindowTextA(es);          // Вывести текст в окно
```

```
    doc1->SetModifiedFlag();                // Установить флаг изменений

    // Делаем полученное текстовое окно активным (расположенным поверх
    ///всех окон)
    // Указатель на главное фреймовое окно
    CMainFrame *mfr = (CMainFrame *)theApp.m_pMainWnd;
    CChildFrame *frch;              // Указатель на дочернее фреймовое окно
    // Получить родительский фрейм текстового представления
    frch = (CChildFrame *)view1->GetParentFrame();
    mfr->MDIActivate(frch);         // Сделать его активным
}
```

Глава 15

Создание справки приложения

Создадим новый проект, на примере которого рассмотрим, как создавать справку для приложения.

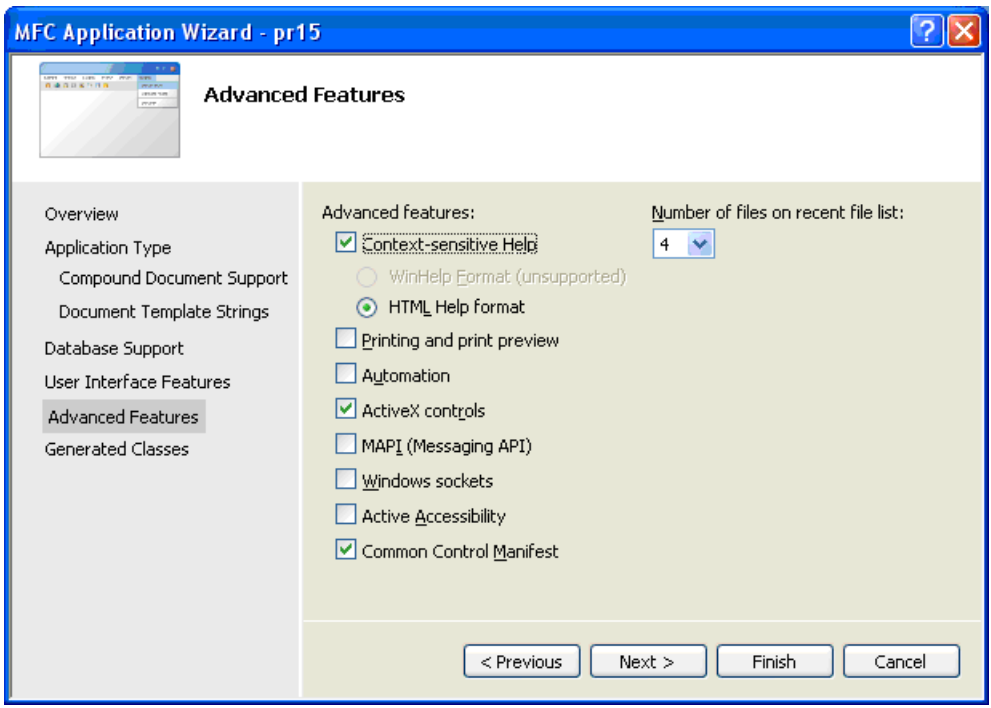


Рис. 15.1. Подключение справки в настройках проекта

Создайте проект **pr15** аналогично проекту **pr14**, *изменив тип приложения* на SDI, выбрав для окна представления базовый класс `CEditView` и *подключив*

справку (флажок **Context-sensitive Help**) на вкладке **Advanced Features** (рис. 15.1). Нужно изменить следующие опции относительно изначально предложенных мастером:

- на вкладке **Application Type**:
  - SDI-документ;
  - без Unicode;
- на вкладке **Advanced Features**:
  - установить флажок **Context-sensitive Help** (переключатель в положении **HTML Help format**);
  - без печати и предварительного просмотра;
- на вкладке **Generated Classes**:
  - в списке **Generated Classes** выбрать класс **Cpr15View**;
  - в списке **Base class** выбрать класс **CEditView**.

15.1. Описание программы

15.1.1. Работа справочной системы

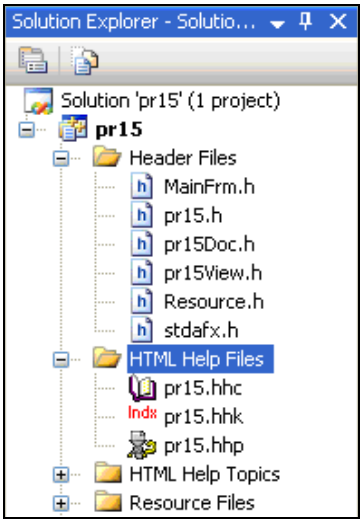
В каталоге проекта появится новый подкаталог hlp, где будут храниться все сгенерированные мастером файлы справки в формате HTML (файлы с расширением htm) и служебные файлы для подключения html-страниц к справке приложения. В окне **Solution Explorer** в составе проекта появятся две новые папки **HTML Help Files** (рис. 15.2, *а*), где находятся служебные файлы для работы со справкой, и **HTML Help Topics** (рис. 15.2, *б*), где находятся файлы с текстами справочных страниц.

В текст программы будет автоматически добавлен код для поддержки работы справочной системы, приведенный в листингах 15.1 и 15.2.

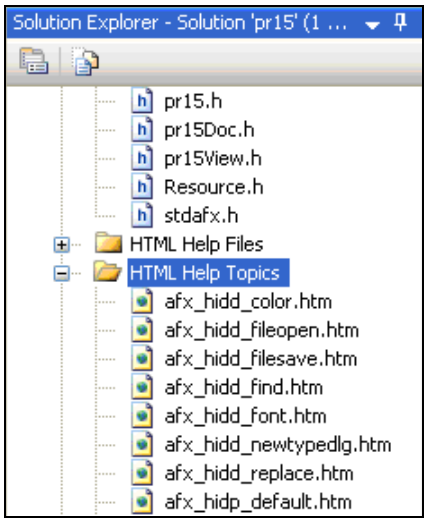
Листинг 15.1. Файл pr15.cpp для поддержки работы со справкой

```
// ...
Cpr15App::Cpr15App()
{
   EnableHtmlHelp();
```

```
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}
// ...
```



а



б

Рис. 15.2. Файлы для работы со справкой (*а*) и файлы с текстами справочных страниц (*б*)

Листинг 15.2. Файл MainFrm.cpp для поддержки работы со справкой

```
// ...
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
    // Global help commands
    ON_COMMAND(ID_HELP_FINDER, &CFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, &CFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP, &CFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP, &CFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()
// ...
```

Следующая функция для возможности использования HTML-справки должна вызываться в конструкторе класса, производного от `CWinApp`:

```
void CWinApp::EnableHtmlHelp();
```

Идентификатор `ID_HELP_FINDER` относится к пункту меню приложения **Help | Help Topics** (рис. 15.3). Идентификатор `ID_CONTEXT_HELP` относится к кнопке контекстной справки (\?) (рис. 15.4). Идентификатор `ID_HELP` относится к обработке нажатия клавиши <F1>. Эти идентификаторы заданы в ресурсе **String Table** (рис. 15.5). Идентификатор `ID_DEFAULT_HELP` является стандартным.

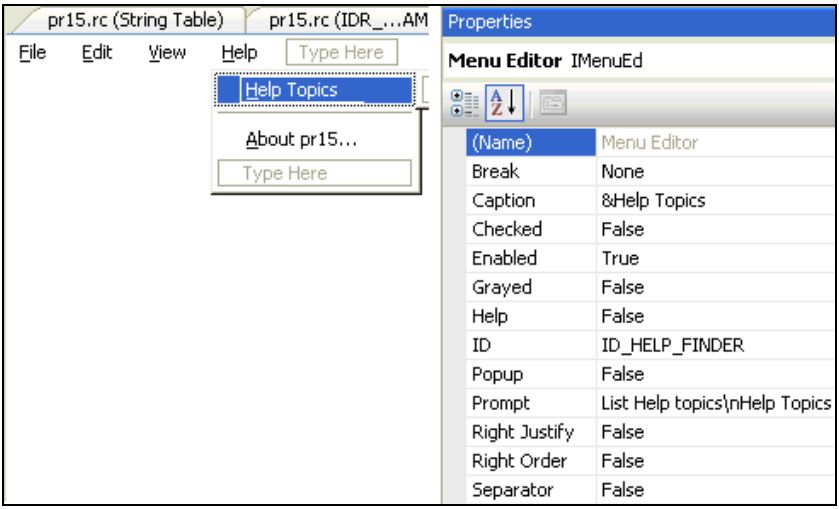


Рис. 15.3. Меню вызова справки

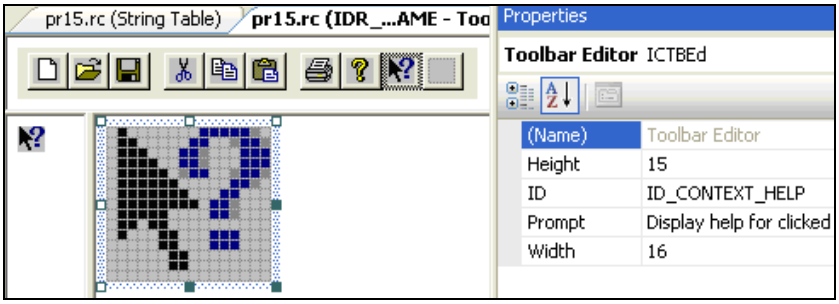


Рис. 15.4. Кнопка для вызова контекстной справки

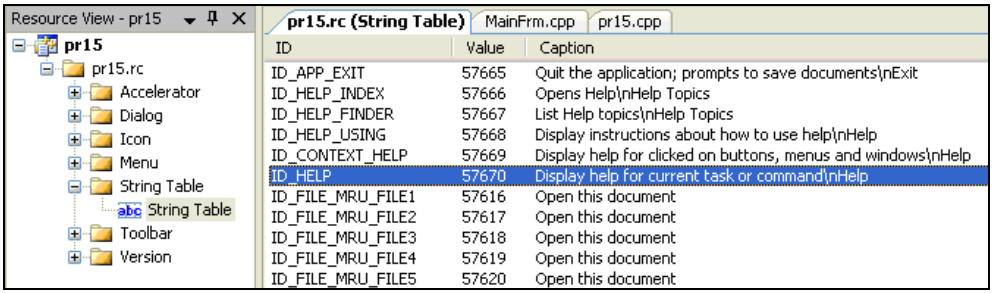


Рис. 15.5. Список идентификаторов для работы справки

Функция `OnHelpFinder()` используется для обработки команд `ID_HELP_FINDER` и `ID_DEFAULT_HELP`. Эта функция вызывает WinHelp.exe со стандартной темой `ID_HELP_FINDER` и определена как:

```
afx_msg void CWnd::OnHelpFinder();
```

или

```
afx_msg void CWinApp::OnHelpFinder();
```

Функция `OnContextHelp()` обработки вызова контекстной справки вызывается, если пользователь нажал кнопку (**\?**) или использовал комбинацию клавиш <Shift>+<F1>. Эта функция вызывает WinHelp.exe с контекстом подсказки для объекта, на который пользователь указал (щелкнул левой кнопкой мыши):

```
afx_msg void CFrameWnd::OnContextHelp();
```

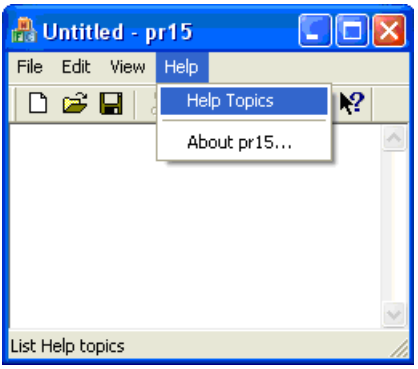
Функция `OnHelp()` обработки вызова быстрой справки вызывается, если пользователь нажал клавишу <F1>. Эта функция вызывает WinHelp.exe для текущего объекта. Если для данного объекта не определен контекст, то используется контекст, определенный по умолчанию:

```
afx_msg void CWnd::OnHelp();
```

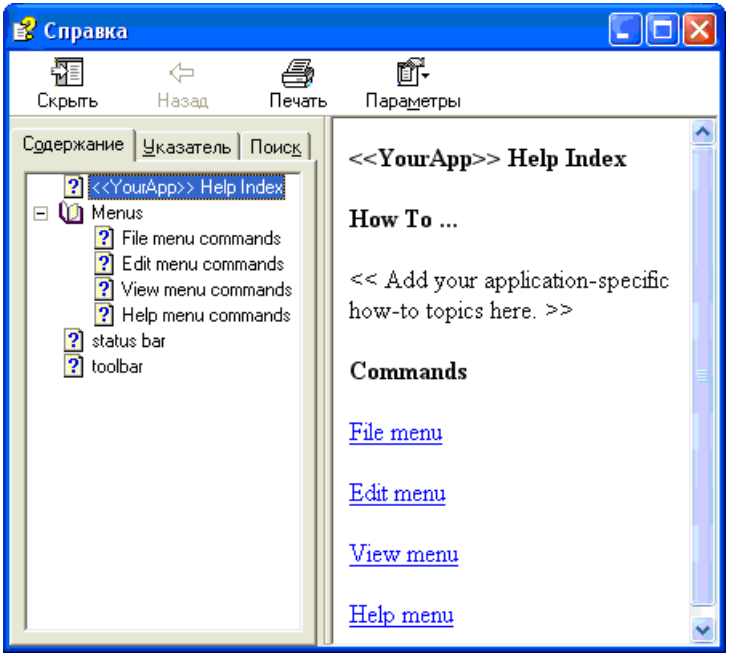
или

```
afx_msg void CWinApp::OnHelp();
```

Результат работы программы показан на рис. 15.6. На панели инструментов приложения добавились новые кнопки: (**?**) — для вызова справки (аналогично меню **Help | Help Topics**) и (**\?**) — для вызова контекстной справки. Если нажать эту кнопку, она "залипает", курсор принимает вид стрелки с вопросом и, при выборе таким курсором пункта меню или кнопки на панели инструментов, выполнение команды не происходит, а вызывается справка по данной команде.



а



б

Рис. 15.6. Вызов окна справки приложения (*а*) и окно справки приложения (*б*)

15.1.2. Файлы справочной системы

Документы HTML состоят из двух разделов: заголовка, содержащего установки глобальных параметров страницы, и основного раздела, содержащего текст и элементы страницы, отображаемые в окне. Данные разделы задаются с помощью пары специальных открывающих (`<DESCRIPTOR>`) и закрывающих

(</DESCRIPTOR>) дескрипторов (тегов), определяющих начало и конец раздела (дескрипторы можно писать заглавными или прописными буквами):

```
<HTML>
<HEAD>
   параметры страницы
</HEAD>
<BODY>
   текст страницы
</BODY>
</HTML>
```

Дескрипторы <HTML>...</HTML> устанавливаются в начале и в конце документа HTML. Дескрипторы бывают парными (<HTML>...</HTML>) и непарными (
 — начать текст с новой строки). Парные дескрипторы могут быть вложенными (<P> <I> текст </I> </P>). Для пояснения документа используют комментарии. Текст комментариев заключается между командными символами <!-- и -->.

```
<--Комментарии могут быть написаны и в несколько
строк, как здесь -->
```

Файл, отвечающий за содержание справки (с расширением hhc)

В этом файле в формате HTML хранится информация о содержании справки (вкладка **Содержание** в окне справки, рис. 15.6, *б*). Текст файла приведен в листинге 15.3.

Листинг 15.3. Файл pr15.hhc с содержанием справки

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<!-- Sitemap 1.0 -->
</HEAD><BODY>
<UL>
```

```
   <LI> <OBJECT type="text/sitemap">
      <param name="Name" value="&lt;&lt;YourApp&gt;&gt; Help Index">
      <param name="Local" value="main_index.htm">
      </OBJECT>
   <LI> <OBJECT type="text/sitemap">
      <param name="Name" value="Menus">
      </OBJECT>
   <UL>
      <LI> <OBJECT type="text/sitemap">
         <param name="Name" value="File menu commands">
         <param name="Local" value="menu_file.htm">
         </OBJECT>
      <LI> <OBJECT type="text/sitemap">
         <param name="Name" value="Edit menu commands">
         <param name="Local" value="menu_edit.htm">
         </OBJECT>
      <LI> <OBJECT type="text/sitemap">
         <param name="Name" value="View menu commands">
         <param name="Local" value="menu_view.htm">
         </OBJECT>
      <LI> <OBJECT type="text/sitemap">
         <param name="Name" value="Help menu commands">
         <param name="Local" value="menu_help.htm">
         </OBJECT>
   </UL>
   <LI> <OBJECT type="text/sitemap">
      <param name="Name" value="status bar">
      <param name="Local" value="afx_hidw_status_bar.htm">
      </OBJECT>
   <LI> <OBJECT type="text/sitemap">
      <param name="Name" value="toolbar">
      <param name="Local" value="afx_hidw_toolbar.htm">
      </OBJECT>
</UL>
</BODY></HTML>
```

Дескриптор `<meta>` является непарным и используется для сохранения обобщающих данных о содержимом страницы. Он используется так:

```
<meta name="тип метаописателя" content="значение метаописателя">
```

Если тип метаописателя `GENERATOR`, то значение метаописателя — это название приложения, которое применялось для создания страницы справки (Microsoft HTML Help Workshop). В названии приложения используется специальный символ `&reg;` — ® (см. рис. 15.9, *б*).

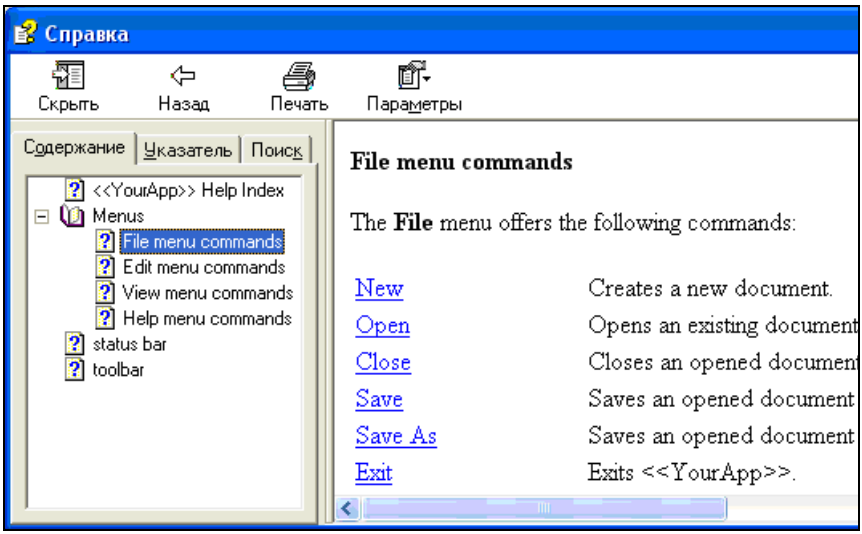


Рис. 15.7. Вид содержания справки

Здесь создается маркированный список содержания. Маркированный список (дескрипторы `<UL>...</UL>`) применяется для перечисления неупорядоченных равнозначных объектов. Пункты списка задаются дескрипторами `<LI>...</LI>`. Для добавления своего объекта используется дескриптор `<OBJECT>...</OBJECT>` (при этом `</LI>` теперь ставить не надо). У каждого объекта задаются его тип (**type**="text/sitemap") и два значения — название элемента, отображаемое в окне справки слева (<**param name**="Name" **value**="File menu commands">) и файл с текстом справочной страницы для данного элемента, отображаемым в окне справки справа при выборе данного элемента (<**param name**="Local" **value**="menu_file.htm">) (рис. 15.7). Для элемента **Menus** указан только заголовок (<**param name**="Name" **value**="Menus">), т. к. при его выборе только открывается или закрывается

иконка-"книжка". Для заголовка верхнего элемента используются специальные символы: `<` — левая угловая скобка (<) и `>` — правая угловая скобка (>).

Файл, отвечающий за работу с указателем и поиском в справке (с расширением hhk)

В этом файле в формате HTML хранится информация о списке ключевых слов справки (вкладка **Указатель** в окне справки, рис. 15.8). Текст файла приведен в листинге 15.4.

Листинг 15.4. Файл pr15.hhk со списком ключевых слов справки

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<!-- Sitemap 1.0 -->
</HEAD><BODY>
<UL>
</UL>
</BODY></HTML>
```

Здесь, хотя в справке и есть ключевые слова в указателе (рис. 15.8), в списке они не отображены. Сюда будут добавляться только ключевые слова, которые были добавлены программистом при работе с проектом. Тогда список мог бы выглядеть так, как показано в листинге 15.5.

Листинг 15.5. Возможный вариант файла pr15.hhk с добавленными ключевыми словами справки

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<!-- Sitemap 1.0 -->
</HEAD><BODY>
<UL>
```

```
   <LI> <OBJECT type="text/sitemap">
        <param name="Name" value="key1">
        <param name="Local" value="file1.htm">
        </OBJECT>
   <LI> <OBJECT type="text/sitemap">
        <param name="Name" value="key2">
        <param name="Local" value="file2.htm">
        </OBJECT>
</UL>
</BODY></HTML>
```

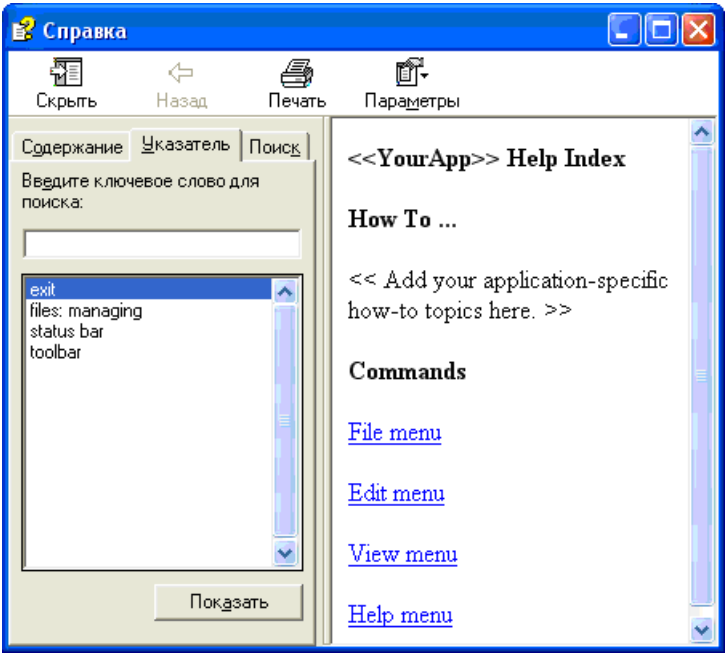


Рис. 15.8. Список ключевых слов справки

Файл проекта справочной системы (с расширением hhp)

Этот файл задает опции проекта справки и необходим для работы компилятора справочной системы, который создает выполняемый файл справки chm

(он находится в каталоге hlp вместе со всеми файлами справки и может быть запущен отдельно, например двойным щелчком мыши).

Текст файла приведен в листинге 15.6.

Листинг 15.6. Файл pr15.hhp

```
[OPTIONS]
Auto Index=Yes
Compatibility=1.1 or later
Compiled file=pr15.chm
Contents file=pr15.hhc
Index file=pr15.hhk
Default topic=main_index.htm
Display compile progress=No
Full-text search=Yes
Language=0x0409 Английский (США)

[FILES]
...
hid_file_new.htm
hid_file_open.htm
hid_file_save.htm.
...

[ALIAS]
HIDR_MAINFRAME = main_index.htm
main_index      = main_index.htm
...
hid_file_new    = hid_file_new.htm
hid_file_open   = hid_file_open.htm
hid_file_save   = hid_file_save.htm
...

[MAP]
#include HTMLDefines.h
```

```
[TEXT POPUPS]
HTMLDefines.h

[INFOTYPES]
```

Этот файл состоит из нескольких секций. Секция `[OPTIONS]` управляет процессом создания справочной системы, где задается конечный файл справки, файлы со списками содержания и индексов, файл со страницей справки, которая используется по умолчанию (при запуске или если не найдена нужная страница), язык справки. Секция `[FILES]` содержит список всех файлов (страниц справки), из которых создается справочная система. Секция `[ALIAS]` содержит идентификаторы и связанные с ними файлы для получения контекстной справки. Секция `[MAP]` предназначена для установки соответствия между идентификаторами и значениями контекстной справки. Они записаны в файле HTMLDefines.h. Он находится в каталоге hlp вместе со всеми файлами справки и содержит определения идентификаторов контекстной справки. Содержимое файла HTMLDefines.h приведено в листинге 15.7.

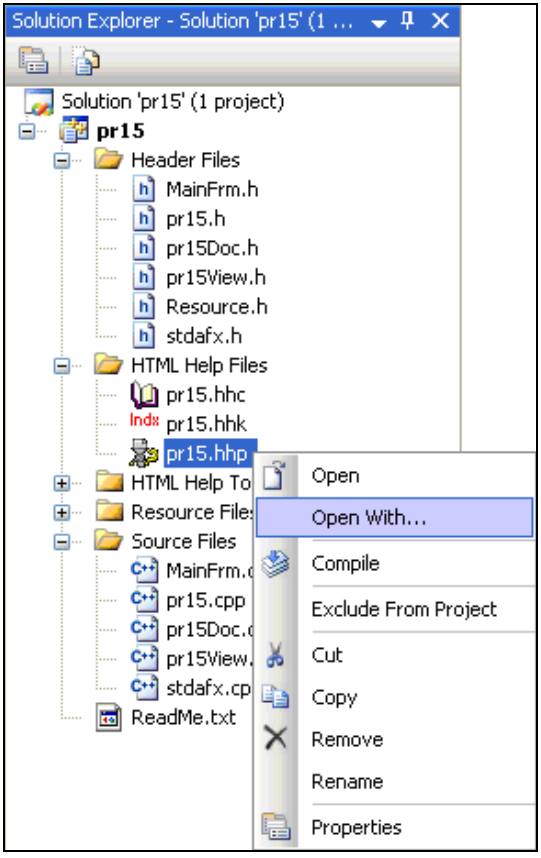
Листинг 15.7. Файл HTMLDefines.h с идентификаторами контекстной справки

```
// ...
#define HIDP_OLE_INIT_FAILED        0x30064
// ...
// Resources (IDR_*)
#define HIDR_MAINFRAME              0x20080
#define HIDR_pr15TYPE               0x20081
// ...
// Dialogs (IDD_*)
#define HIDD_ABOUTBOX               0x20064
// ...
// Commands (HID_*)
#define HID_FILE_NEW                0x1E100
#define HID_FILE_OPEN               0x1E101
#define HID_FILE_CLOSE              0x1E102
#define HID_FILE_SAVE               0x1E103
// ...
```

15.1.3. Добавление своей справки

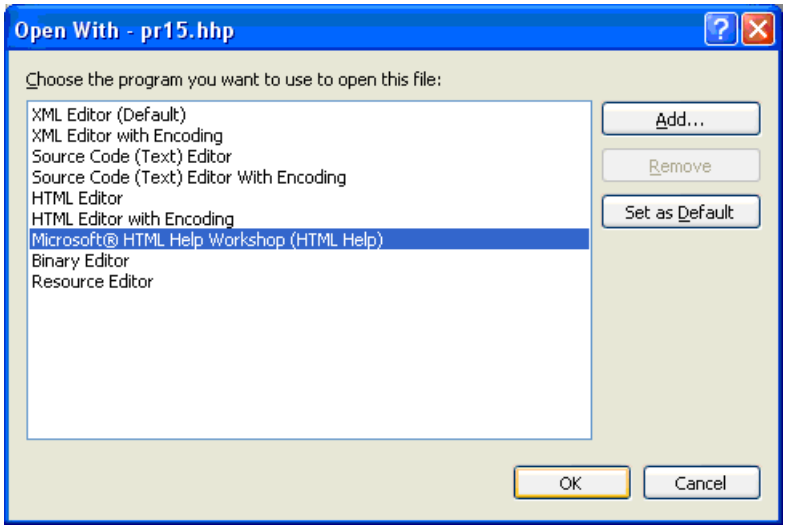
Можно добавить свои разделы справки, отредактировав файлы pr15.hhc, pr15.hhk, pr15.hhp. Но это намного удобнее делать, используя специальный редактор HTML-справки Microsoft HTML Help Workshop. Он поставляется с пакетом Microsoft Visual Studio и по умолчанию запускается как C:\Program Files\HTML Help Workshop\hhw.exe.

Для редактирования справки надо в окне **Solution Explorer** проекта **pr15** вызвать контекстное меню для файла pr15.hhp и выбрать пункт **Open With** (Открыть с помощью). В списке программ нужно выбрать редактор HTML-справки **Microsoft HTML Help Workshop (HTML Help)** (рис. 15.9).



а

Рис. 15.9. Открытие файла проекта справки (*а*)



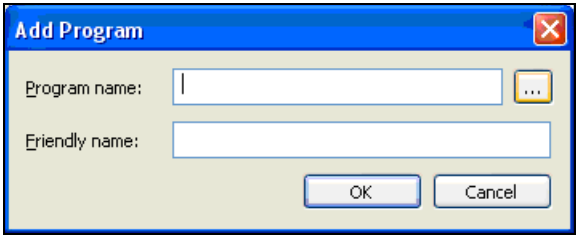
б

Рис. 15.9. Выбор программы для редактирования справки (*б*)

Если в списке такого редактора нет, то его можно добавить самостоятельно, нажав кнопку **Add** и указав явно файл hhw.exe (рис. 15.10). Вид редактора справки показан на рис. 15.11.

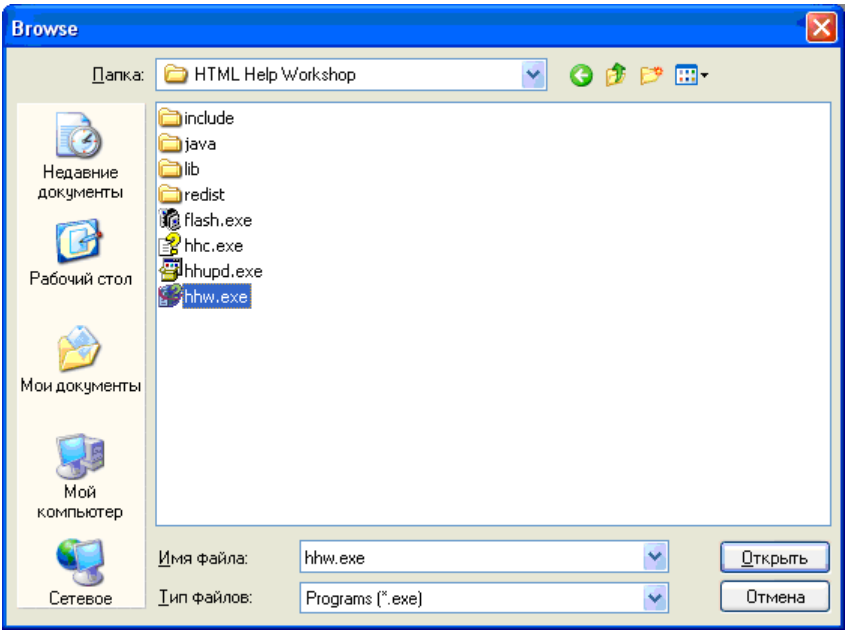
Примечание

При самостоятельном добавлении редактора в список и открытии файла hhp, редактор *не позволяет записывать изменения в файле проекта* справки. Чтобы этого избежать, надо закрыть в редакторе справки текущий проект и, *не выходя из редактора*, открыть проект снова (рис. 15.12), тогда все изменения будут *записываться*. Если редактор справки изначально был в списке программ, этого делать не надо.



а

Рис. 15.10. Добавление
новой программы редактирования (*а*)



б

Рис. 15.10. Выбор программы HTML Help Workshop (б)

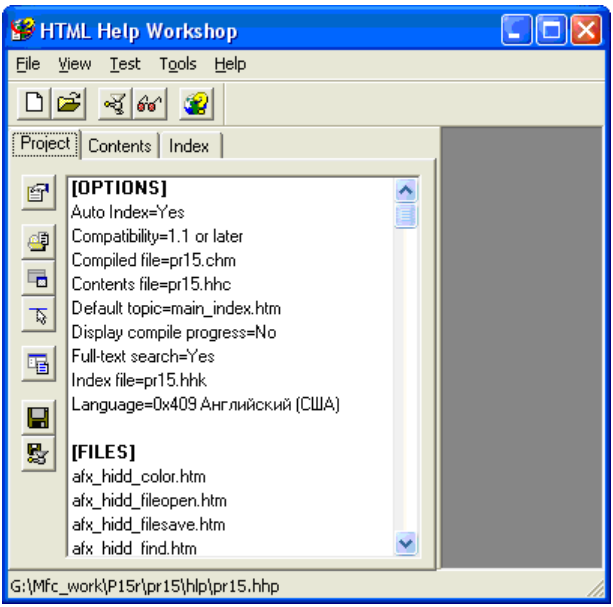


Рис. 15.11. Вид редактора справки

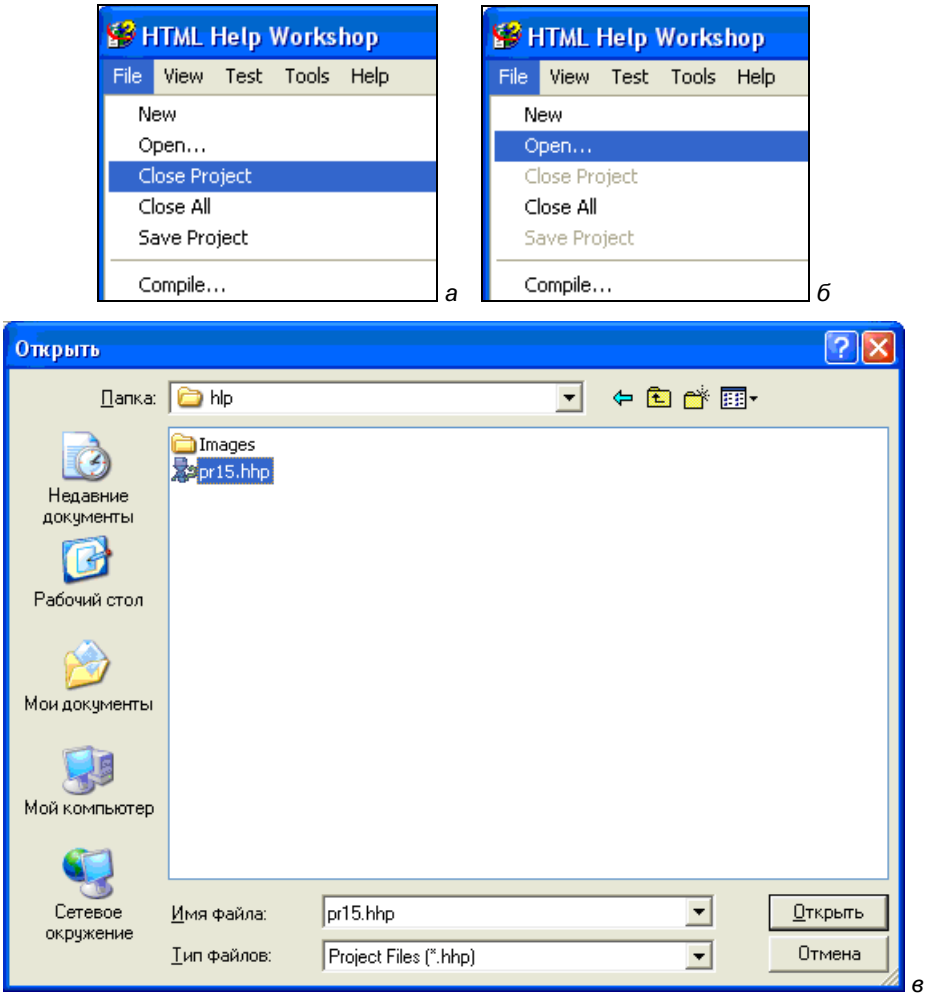


Рис. 15.12. Закрытие (*а*) и открытие (*б*) проекта справки с помощью меню, выбор файла проекта справки (*в*)

Для того чтобы можно было пользоваться справкой на *русском языке*, надо изменить секцию `[OPTIONS]` в файле pr15.hhp. Это можно сделать, выбрав на левой вертикальной панели инструментов верхнюю кнопку **Change project options** (Изменение опций проекта) или просто дважды щелкнув мышью по строке с названием языка (рис. 15.13). В появившемся окне в списке языков **Language** (Язык) надо выбрать **Русский** и нажать кнопку **OK** (рис. 15.14).

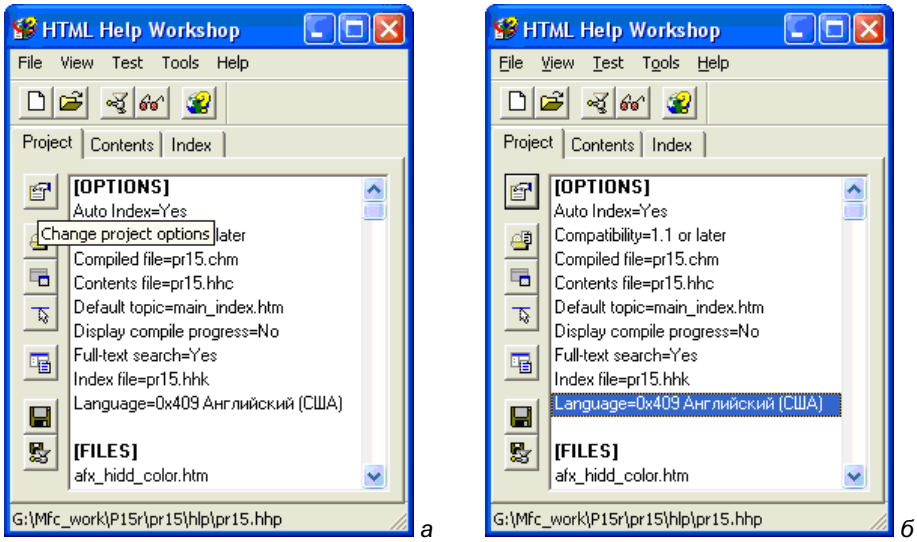


Рис. 15.13. Изменение опций проекта с помощью кнопки на панели инструментов (*а*) или с помощью выбора опции (*б*)

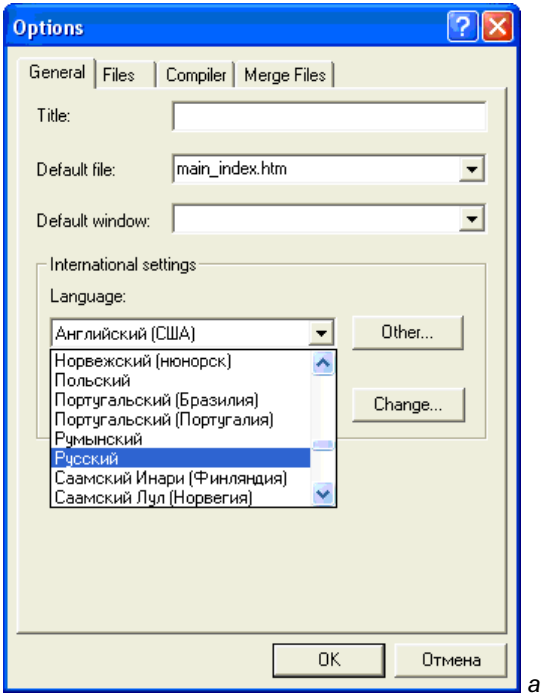
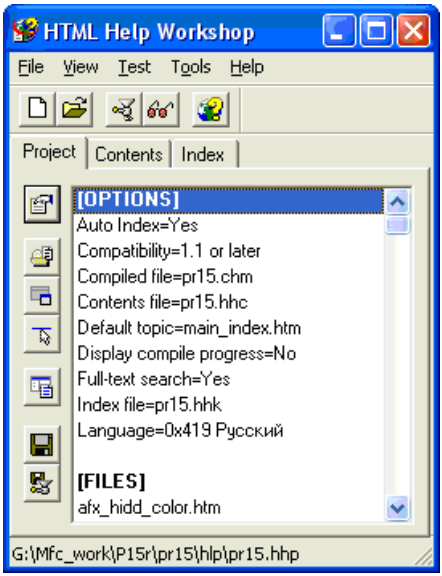


Рис. 15.14. Изменение языка справки (*а*)



б

Рис. 15.14. Новый вид опций проекта (*б*)

Добавление пункта в содержание справки

Для добавления нового пункта (в виде иконки-"книжки") надо с вкладки **Project** перейти на вкладку **Contents** (Содержание). Работать с содержанием справки можно с помощью контекстного меню или с помощью кнопок на левой панели инструментов (рис. 15.15).

Добавим новый пункт **My menu** после пункта **status bar** с помощью второй кнопки на левой панели инструментов **Insert a heading** (Вставить заголовок) с изображением книжки (рис. 15.16). В открывшемся окне **Table of Contents Entry** (Параметры пункта оглавления) надо в поле **Entry title** (Название заголовка) задать имя пункта (рис. 15.17) и нажать кнопку **OK**. В этом случае при выборе этого пункта (заголовка) "книжка" будет раскрываться и закрываться. Если в окне **Table of Contents Entry** нажать кнопку **Add**, то будет предложено связать эту книжку с файлом справки, и при выборе этого пункта будет открываться страница справки. Но это не принято.

Новый пункт будет добавлен после выделенного пункта (рис. 15.18). Выбранный пункт можно переместить с помощью кнопок со стрелками, которые находятся в нижней части левой панели инструментов (рис. 15.19).

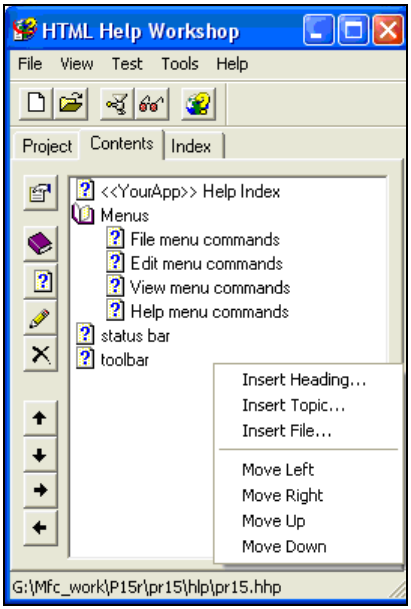


Рис. 15.15. Работа с содержанием справки

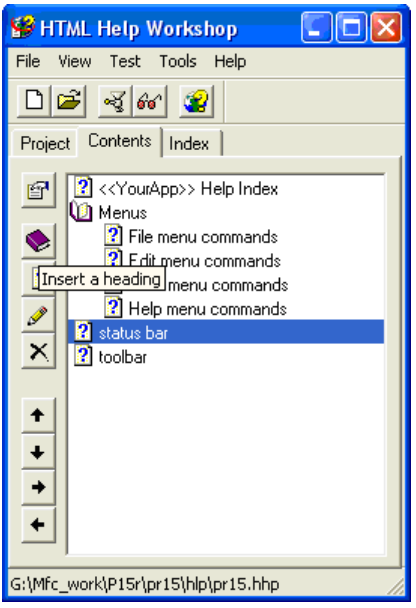


Рис. 15.16. Добавление нового заголовка в меню справки

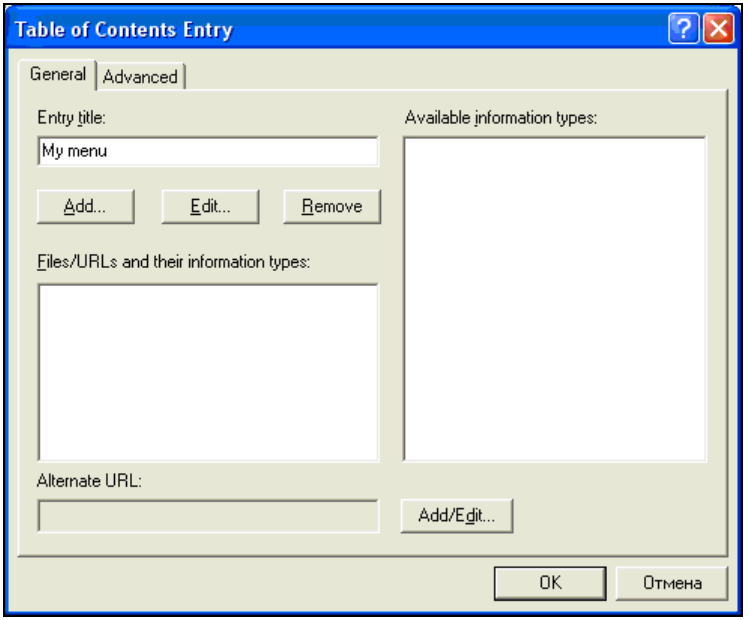


Рис. 15.17. Задание имени заголовка в меню справки

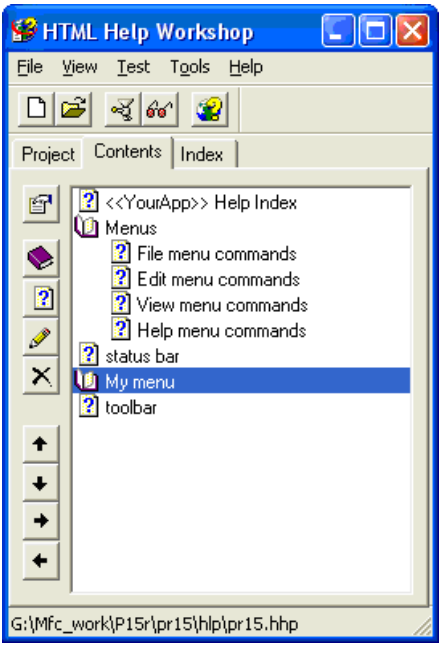


Рис. 15.18. Новый пункт в меню справки

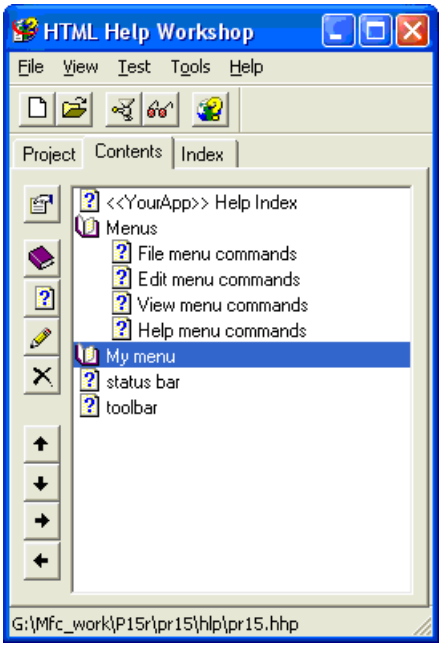


Рис. 15.19. Перемещение пунктов в меню справки

Добавление подпунктов в содержание справки

Для того чтобы можно было пользоваться страницами справки, надо для каждого пункта подготовить HTML-файл.

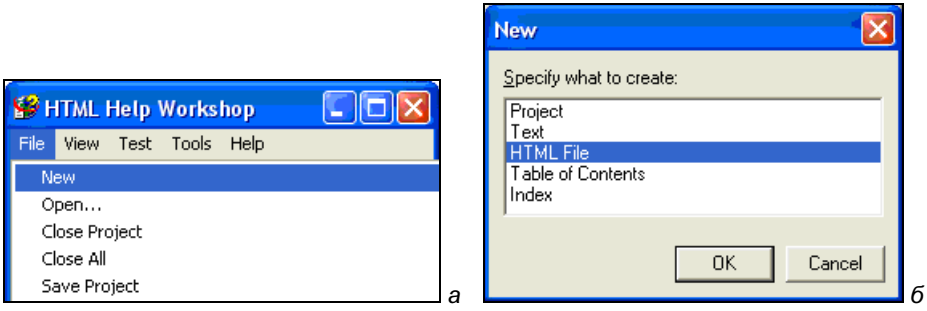


Рис. 15.20. Добавление нового файла (*а*) для страницы справки и выбор его типа (*б*)

У нас будет 3 подпункта, поэтому подготовим три файла. Для создания файла справки в редакторе HTML Help Workshop надо:

- выбрать меню **File | New** (рис. 15.20, *а*);
- в окне **New** выбрать тип нового файла — **HTML File** (рис. 15.20, *б*);
- в окне **HTML Title** ввести заголовок страницы справки `my_title1` (это именно заголовок страницы, а *не имя файла*, рис. 15.21).

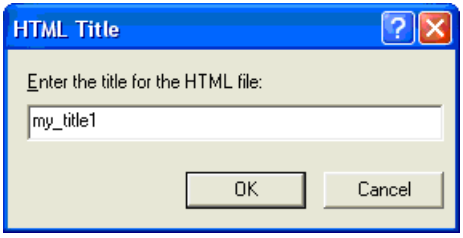


Рис. 15.21. Задание заголовка страницы

В результате в правой половине окна появится новый (пока безымянный) файл (рис. 15.22). Содержимое файла приведено в листинге 15.8.

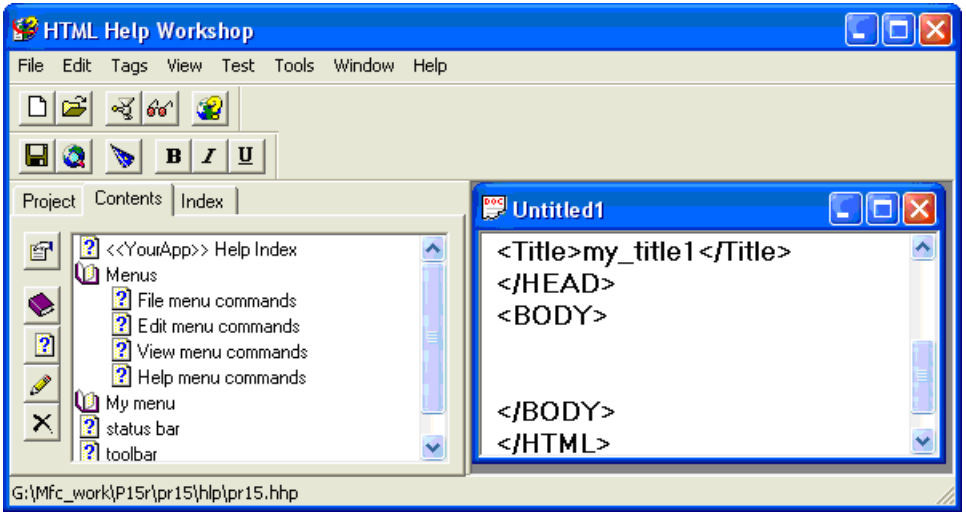


Рис. 15.22. Шаблон нового файла справки

Листинг 15.8. Шаблон файла новой страницы справки

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<Title>my_title1</Title>
</HEAD>
<BODY>


</BODY>
</HTML>
```

Дескрипторы `<meta>`, `<HTML>` и `<HEAD>` были описаны ранее.

Дескриптор `<Title>...</Title>` задает заголовок страницы, отображаемый в строке заголовка обозревателя (будет показано далее).

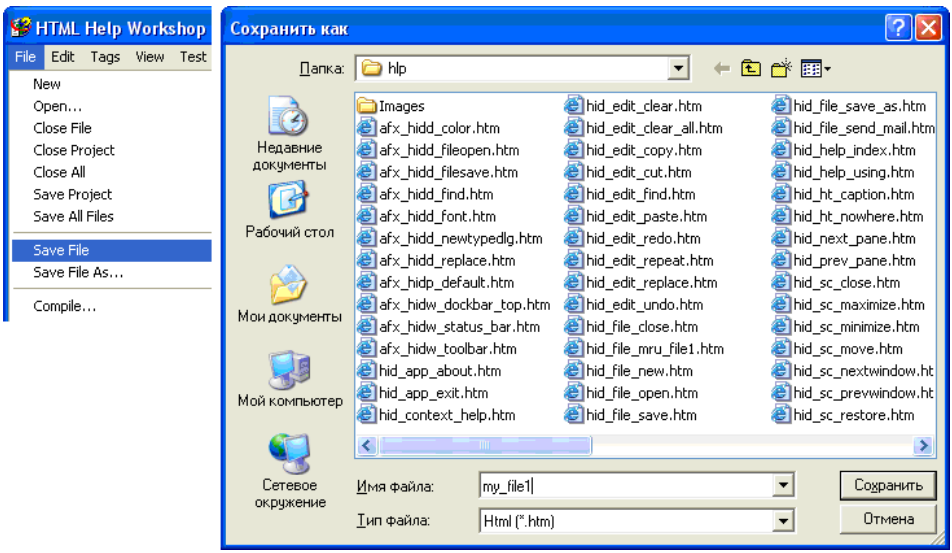


Рис. 15.23. Сохранение файла новой страницы справки

Между дескрипторами `<BODY>...</BODY>` записывается непосредственный текст страницы справки. Сохраним файл с именем my_file1.htm (рис. 15.23).

Добавим текст страницы справки. Изменения в файле приведены в листинге 15.9.

Листинг 15.9. Изменения в файле my_file1.htm для первой страницы справки

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<Title>my_title1</Title>
</HEAD>
<BODY>
Моя справка1
</BODY>
</HTML>
```

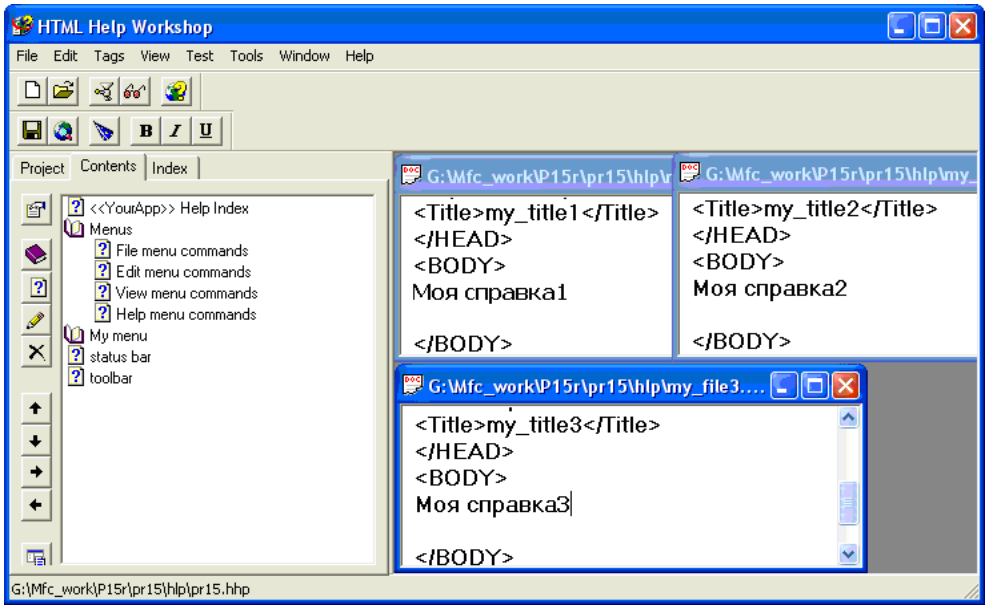


Рис. 15.24. Три подготовленные страницы справки

Аналогично надо подготовить файл my_file2.htm (с заголовком страницы `my_title2` и текстом `Моя справка2`) и my_file3.htm (с заголовком страницы `my_title3` и текстом `Моя справка3`) (рис. 15.24).

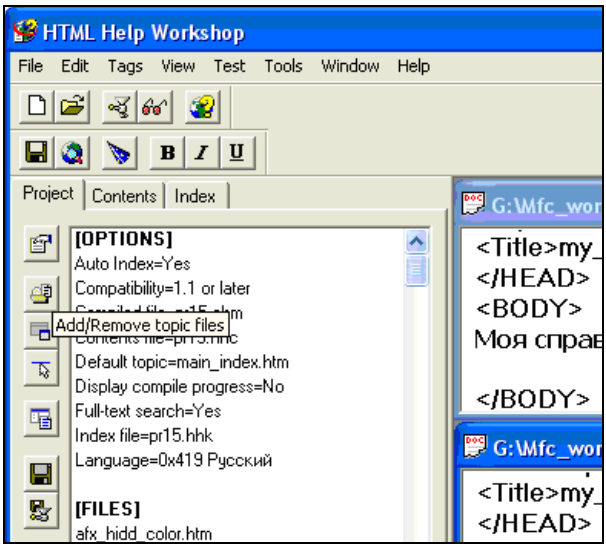


Рис. 15.25. Добавление файлов справки

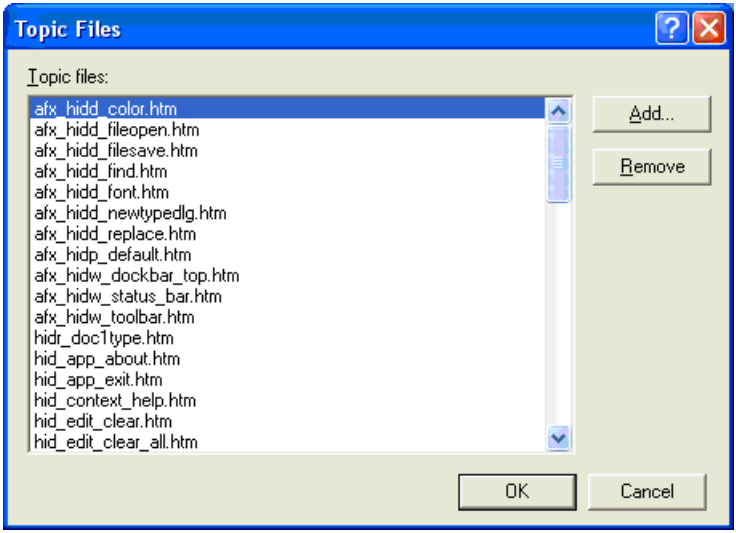


Рис. 15.26. Список файлов справки

Теперь надо добавить эти файлы в проект справки (секция `[FILES]`). Для этого на вкладке **Project** на левой панели инструментов выбрать вторую сверху кнопку **Add/Remove topic files** (рис. 15.25). В появившемся окне **Topic Files** со списком файлов нажать кнопку **Add** (рис. 15.26). Выбрать и добавить три своих файла (рис. 15.27). Эти файлы появятся в секции `[FILES]` (рис. 15.28). Нажмите кнопку **OK**.

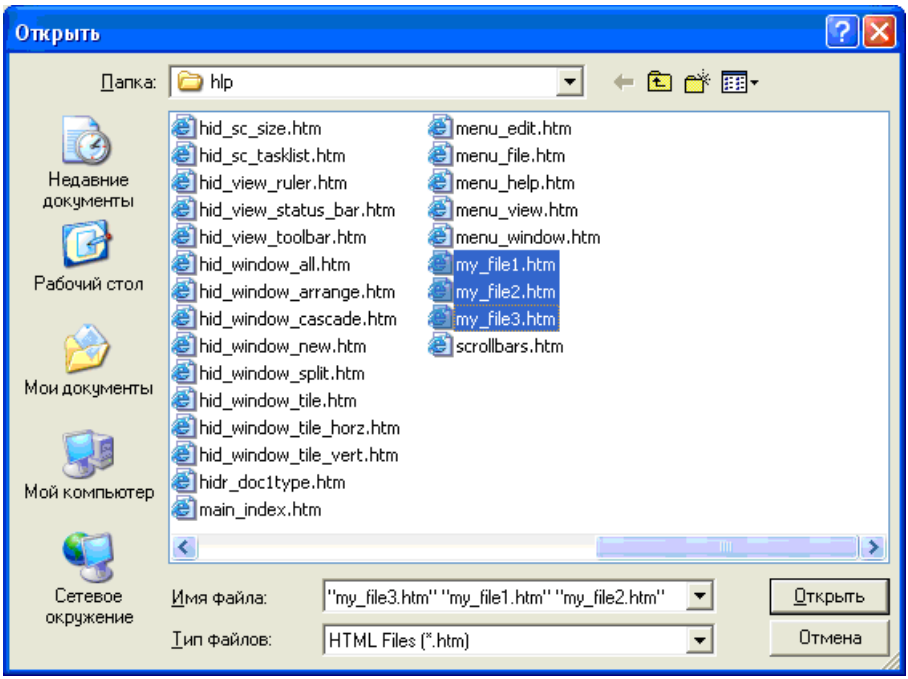


Рис. 15.27. Выбор добавляемых файлов справки

Добавим в пункт справки **My menu** подпункт **my_page1** и свяжем его с содержимым страницы, хранящимся в файле my_file1.htm. Для этого надо перейти на вкладку **Contents**, сделать текущим пункт **My menu** и нажать третью кнопку **Insert a page** на левой панели инструментов (рис. 15.29). В открывшемся окне **Table of Contents Entry** надо в поле **Entry title** задать имя `my_page1` (рис. 15.30) и нажать кнопку **Add**. В появившемся окне **Path or URL** можно с помощью кнопки **Browse** добавить свой файл my_file1.htm или, что гораздо удобнее, найти в списке **HTML titles** заголовок нужной страницы `my_title1`, и связанный с ней файл добавится автоматиче-

ски (рис. 15.31). Аналогичным образом добавим пункт **my_page2** (связанный с `my_title2` и my_file2.htm) и пункт **my_page3** (связанный с `my_title3` и my_file3.htm) (рис. 15.32).

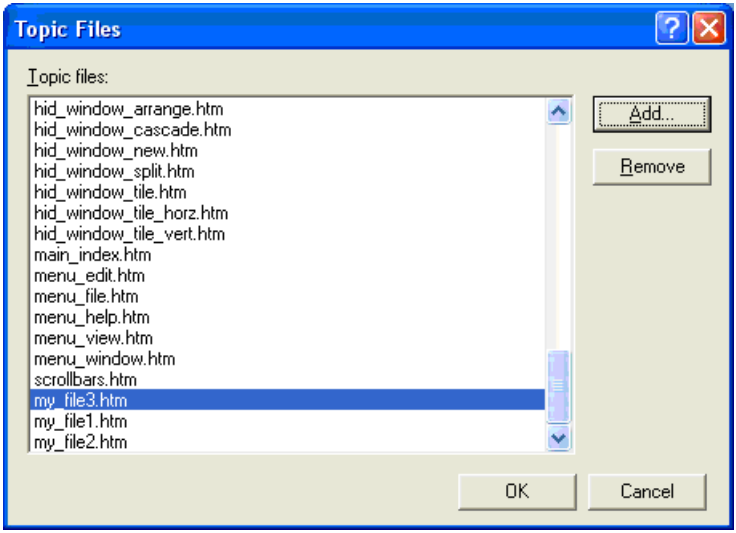


Рис. 15.28. Новые файлы добавлены в проект справки

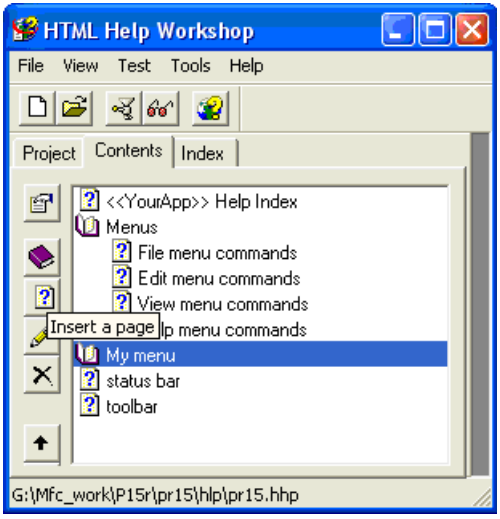


Рис. 15.29. Добавление новых пунктов в содержание справки

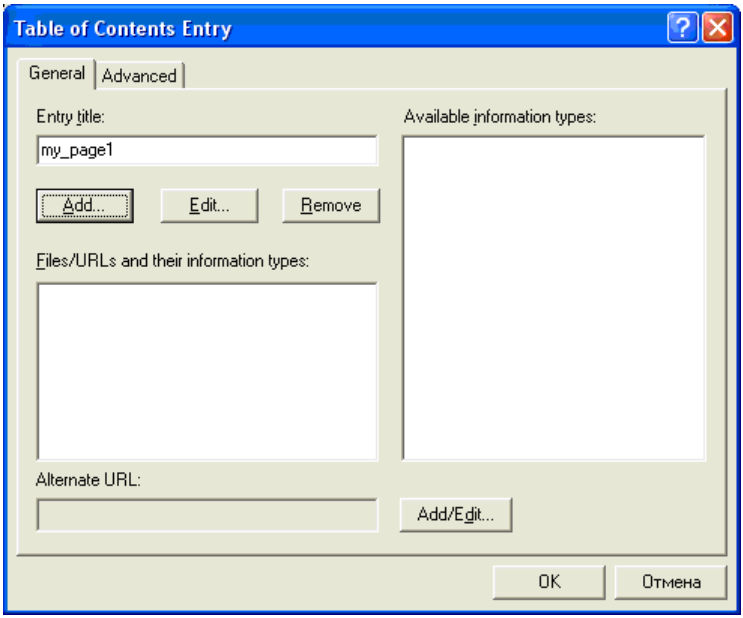


Рис. 15.30. Задание названия нового пункта

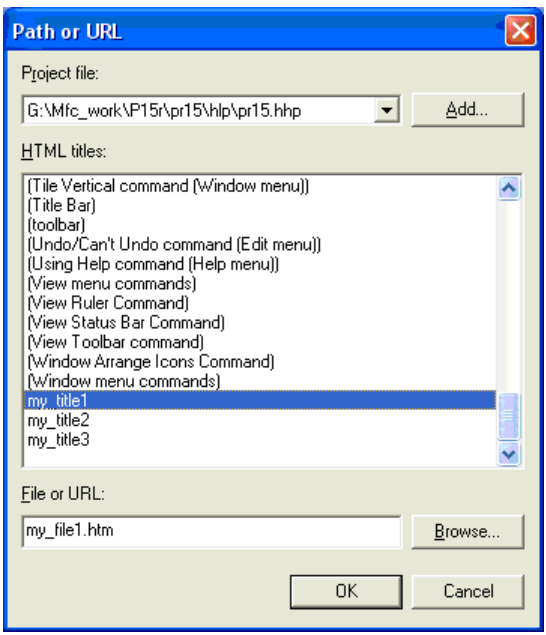


Рис. 15.31. Связывание нового пункта с файлом страницы справки

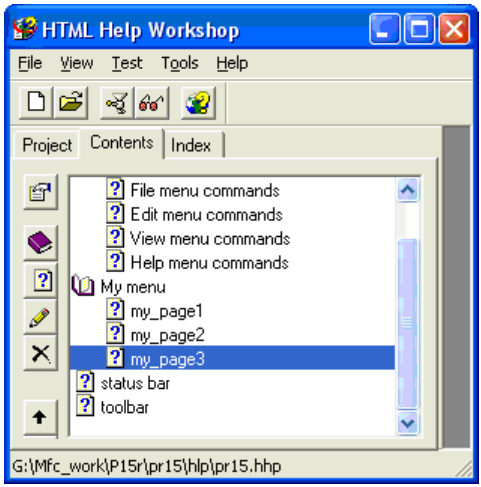


Рис. 15.32. Измененное содержание справки

Проверка работы справки

Чтобы проверить работу справки, надо на вкладке **Project** на левой панели инструментов выбрать нижнюю кнопку **Save all files and compile** (рис. 15.33).

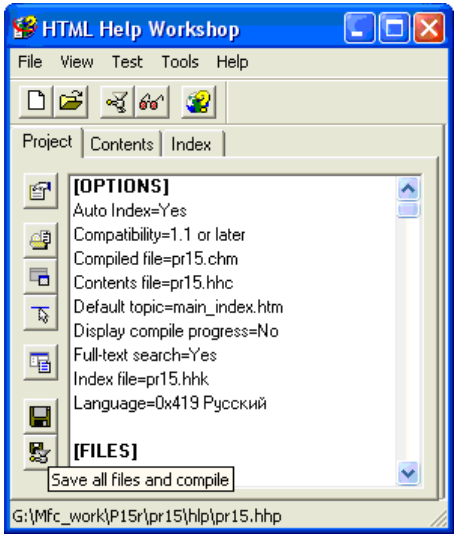


Рис. 15.33. Сохранение и компиляция проекта справки

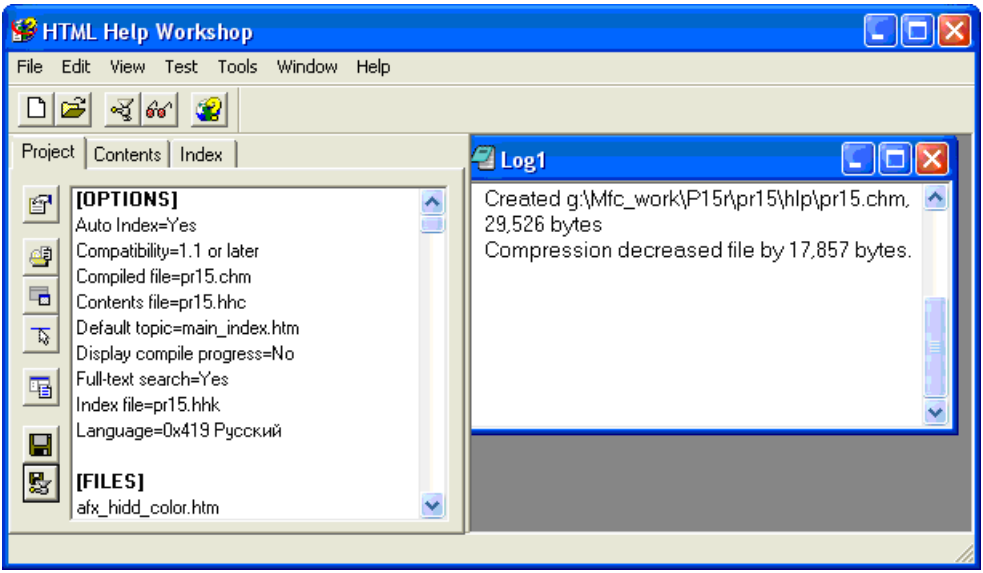
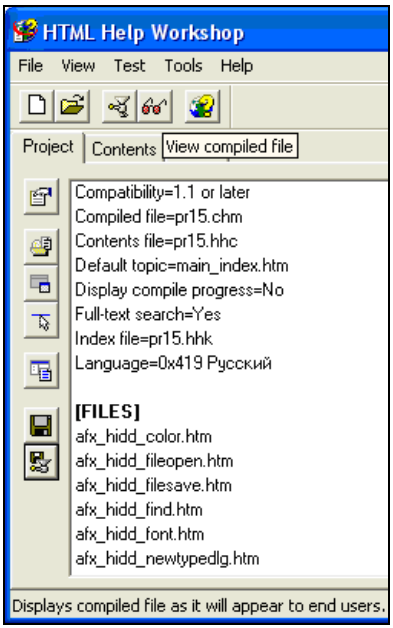
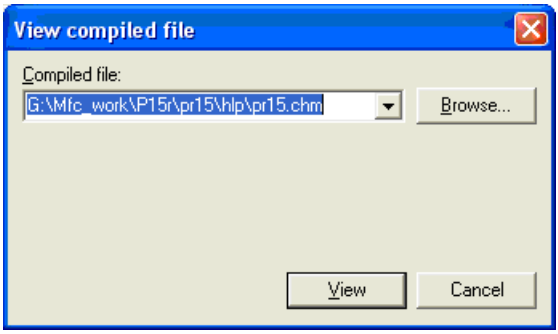


Рис. 15.34. Результаты компиляции проекта справки



а

Рис. 15.35. Просмотр работы готового файла справки с помощью кнопки на панели инструментов (*а*)



б

Рис. 15.35. Выбор запускаемого файла справки (*б*)

Все файлы справки будут сохранены, и файл проекта будет скомпилирован. Результат компиляции будет отражен в окне справа, где появится информация о названии запускаемого файла справки, которое задается в секции `[OPTIONS]` проекта (pr15.chm) и его размер (рис. 15.34).

Теперь можно запустить справку с помощью кнопки с изображением очков на верхней панели инструментов **View compiled file** (рис. 15.35, *а*). В появившемся окне указывается запускаемый файл — откомпилированный файл проекта (рис. 15.35, *б*). При нажатии кнопки **View** начинается работа справочной системы, которую можно протестировать (рис. 15.36).

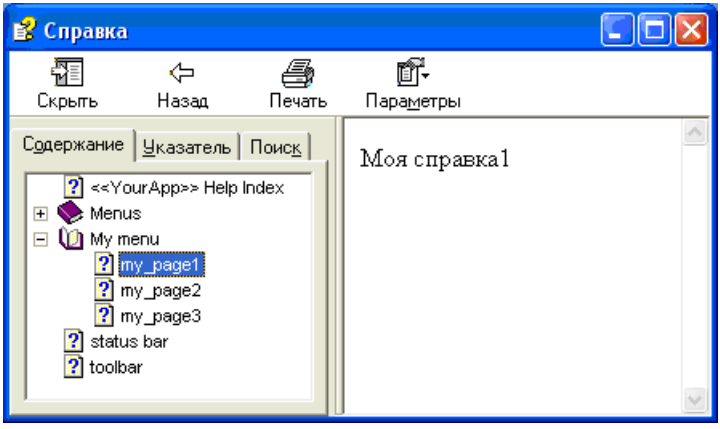


Рис. 15.36. Тестирование справочной системы

Добавление гиперссылок

Сделаем так, чтобы со страницы **my_page1** нашей справки можно было переходить на страницу **my_page2**. Для этого надо отредактировать файл my_file1.htm, добавив в него гиперссылку на файл my_file2.htm. Чтобы быстро открыть файл my_file1.htm, дважды щелкните по пункту **my_page1** на вкладке **Contents**.

Для редактирования HTML-файлов в редакторе в пункте главного меню **Tags** есть некоторые команды (рис. 15.37), но гиперссылку надо добавлять вручную. Изменения в файле приведены в листинге 15.10.

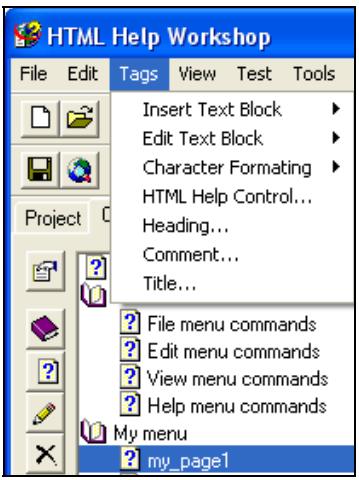


Рис. 15.37. Меню для редактирования HTML-файлов

Листинг 15.10. Изменения в файле my_file1.htm для добавления ссылки на вторую страницу справки

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft&reg; HTML Help Workshop 4.1">
<Title>my_title1</Title>
</HEAD>
```

```
<BODY>
Моя справка1
переход к <A HREF="my_file2.htm">справке2</A>
</BODY>
</HTML>
```

Для создания гиперссылки используется дескриптор `<A>...</A>`:

```
<A HREF="файл страницы">текст гиперссылки</A>
```

Текст гиперссылки, по умолчанию, отображается на странице синим подчеркнутым шрифтом и при щелчке по нему мышью происходит переход к заданному файлу страницы (рис. 15.38).

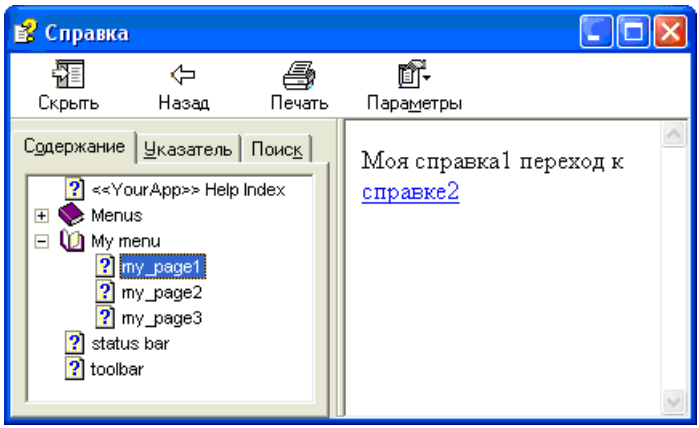


Рис. 15.38. Использование гиперссылки

Добавление ключевых слов в указатель

Добавим в список указателей справки ключевое слово `page1`, по которому будет отображаться страница **my_page1**. Для добавления ключевых слов надо перейти на вкладку **Index** и выбрать вторую сверху кнопку **Insert a keyword** (Добавить ключевое слово) на левой панели инструментов (рис. 15.39). В появившемся окне **Index Entry** нужно задать в поле **Keyword** ключевое слово `page1` (рис. 15.40) и нажать кнопку **Add**.

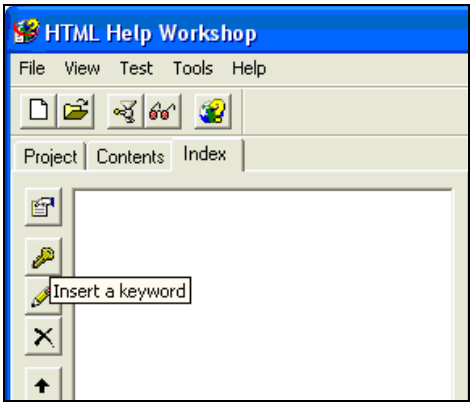


Рис. 15.39. Добавление ключевых слов

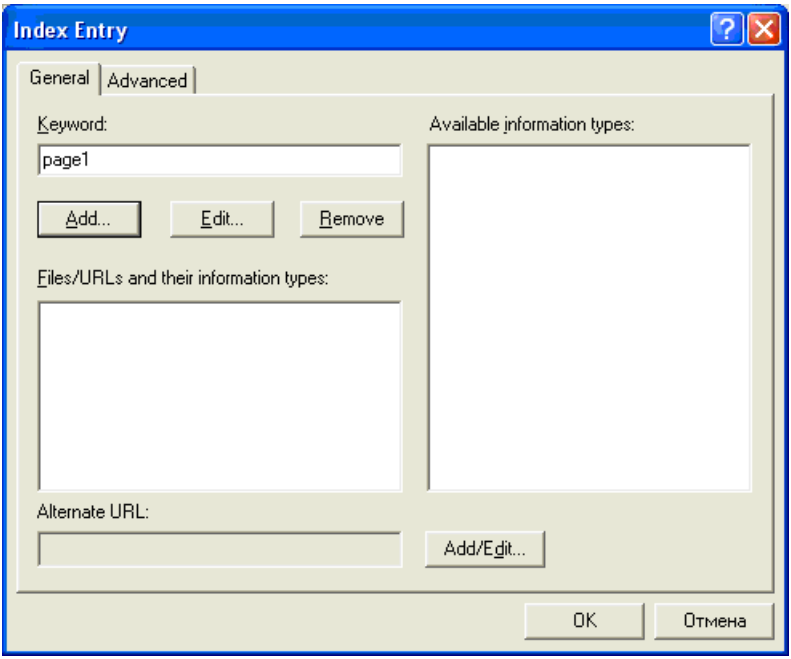


Рис. 15.40. Добавление нового ключевого слова

В окне **Path or URL** в списке **HTML titles** надо найти и выбрать `my_title1`, при этом поля **File or URL** и **Title** заполнятся автоматически (рис. 15.41), нажать кнопку **OK** (рис. 15.42).

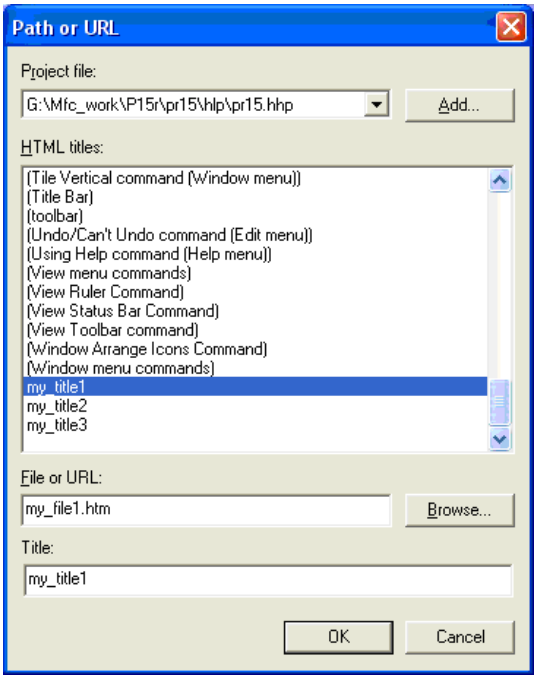
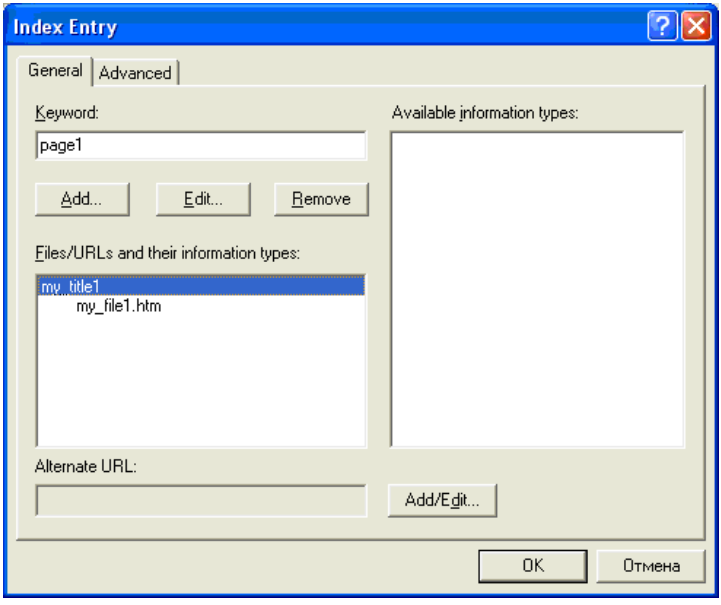
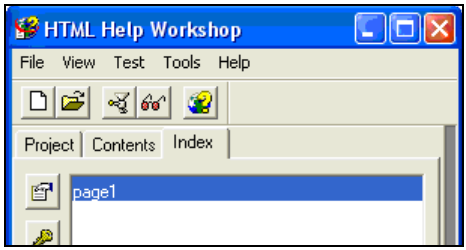


Рис. 15.41. Связывание нового ключевого слова с нужной страницей справки



а

Рис. 15.42. Добавление ключевого слова и связывание с ним файла (*а*)



б

Рис. 15.42. Измененный список ключевых слов справки (*б*)

Результат работы справки показан на рис. 15.43.

Можно добавить ключевое слово, по которому можно будет открывать не одну страницу справки, а выбирать нужную.

Добавим ключевое слово `page23`, чтобы по нему можно было выбирать **my_page2** или **my_page3**. Ключевое слово `page23` связывается со страницей **my_page2** аналогично `page1` (рис. 15.44). Теперь, *не закрывая* окно **Index Entry**, надо снова нажать кнопку **Add** и добавить страницу **my_page3** (рис. 15.45 и 15.46).

Если сделать двойной щелчок мышью на вкладке **Index** по **page23**, то откроется список подключенных к этому ключевому слову страниц (рис. 15.47).

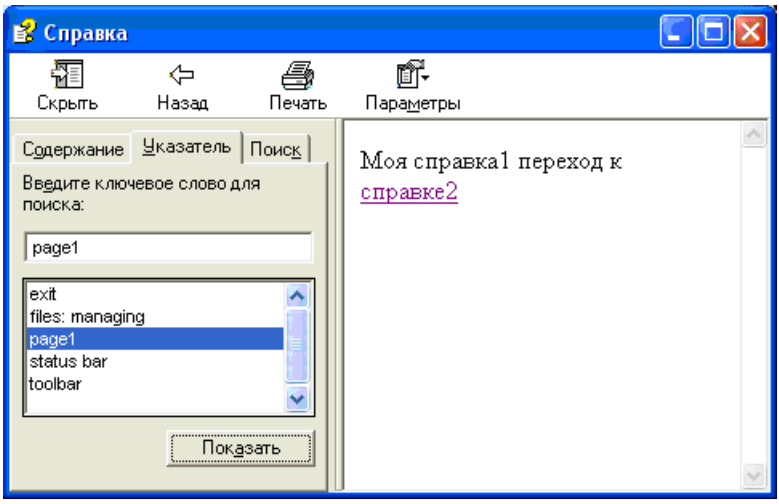


Рис. 15.43. Работа указателя справки

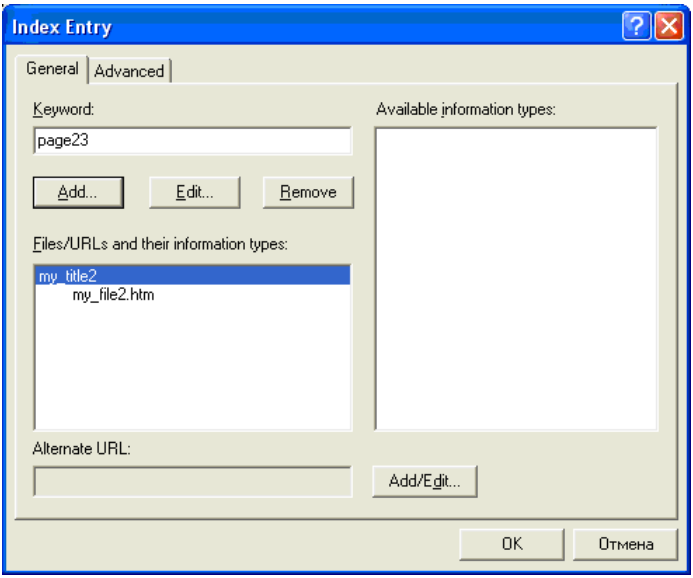


Рис. 15.44. Подготовка второго ключевого слова

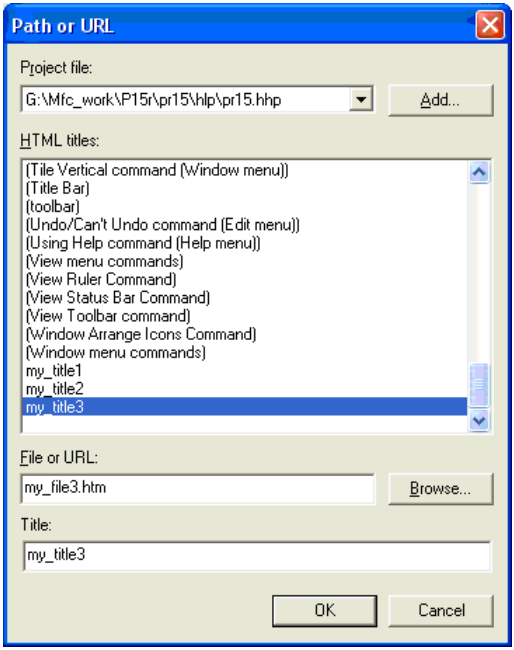


Рис. 15.45. Связывание ключевого слова со второй страницей справки

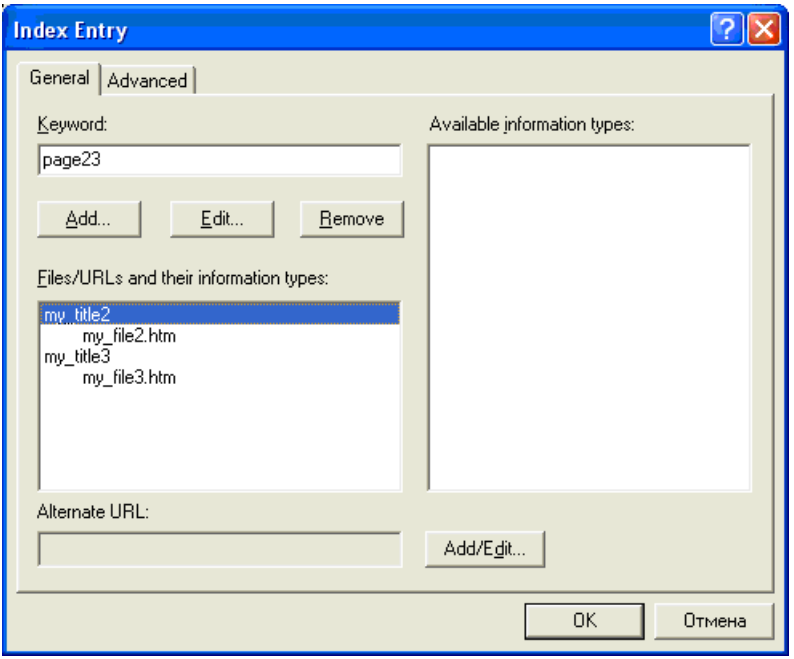
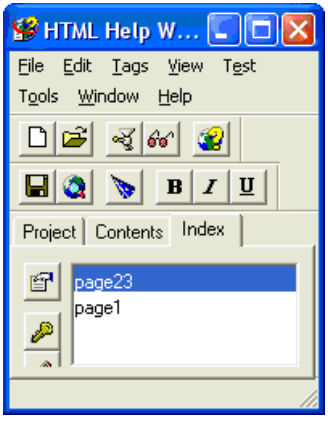
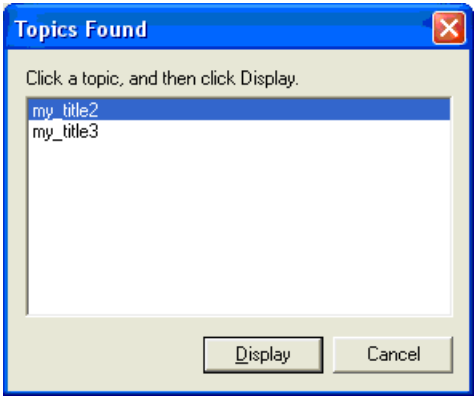


Рис. 15.46. Вид ключевого слова и связанных с ним двух файлов



а



б

Рис. 15.47. Выбор ключевого слова (*а*) и просмотр списка страниц для него (*б*)

Результат работы справки показан на рис. 15.48.

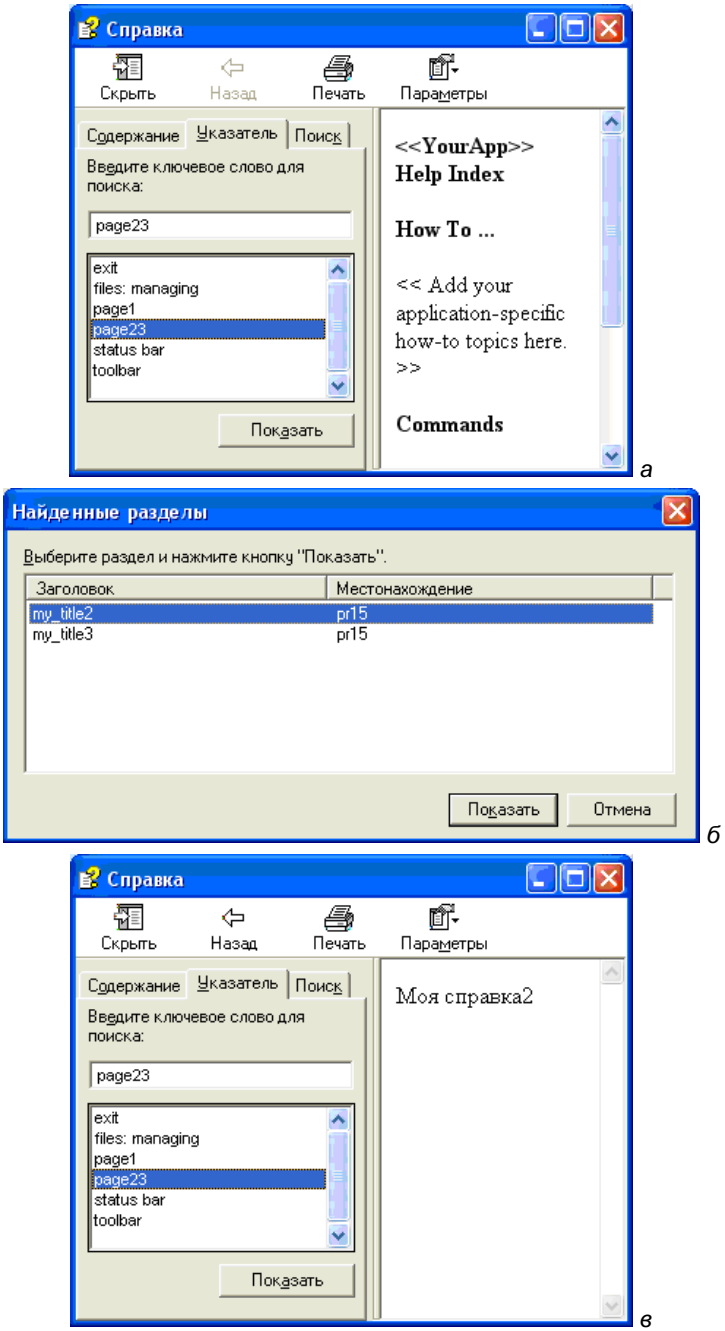


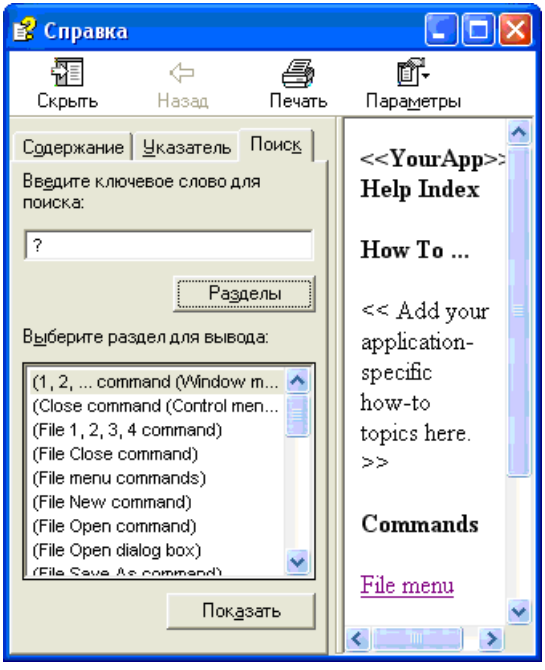
Рис. 15.48. Выбор ключевого слова, связанного с двумя страницами (*а*), выбор нужной страницы (*б*) и открытие выбранной страницы справки (*в*)

Работа с поиском справки

Для поиска используются заголовки страниц, заданные дескриптором `<Title>...</Title>`. При выборе ключевого слова для поиска можно пользоваться символами `'?'` (что эквивалентно любой одной букве) и `'*'` (любая последовательность букв). Результаты работы поиска показаны на рис. 15.49. Если в поле поиска задать символ `'?'`, то буден выведен список всех разделов (рис. 15.49, *а*). Для того чтобы найти наши разделы, надо задать в строке поиска `my*` или `my_title?` (рис. 15.49, *б* и *в*).

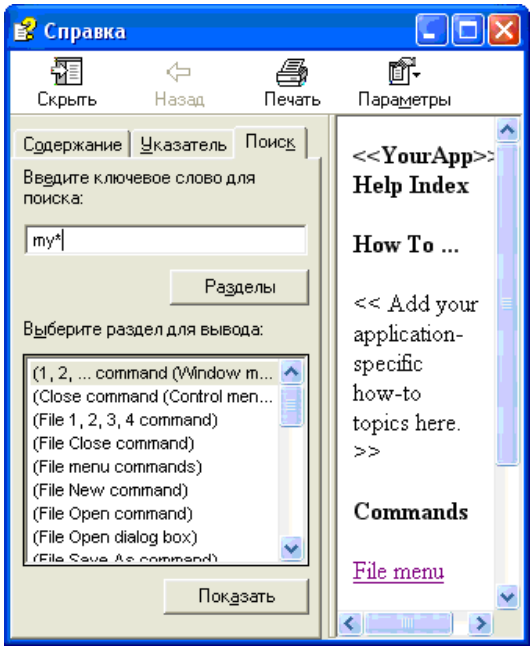
Создание контекстной справки

Так как при создании проекта мы отключили возможность печати документа (сняли флажок **Printing and print preview**, см. рис. 15.1), то в меню нет пунктов для работы с печатью, но на панели инструментов осталась соответствующая кнопка (без обработки и контекстной справки). Если запросить на нее контекстную справку, то выдается окно сообщения (рис. 15.50).

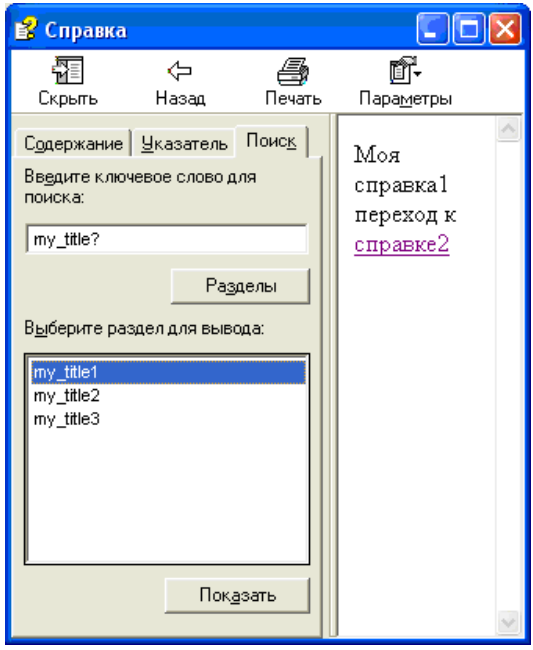


а

Рис. 15.49. Работа с поиском справки: выбор общей (*а*)



б



в

Рис. 15.49. Работа с поиском справки: выбор частной (*б*) маски для поиска справки, открытие нужной страницы справки (*в*)

Рис. 15.50. Попытка вызова контекстной справки для кнопки печати

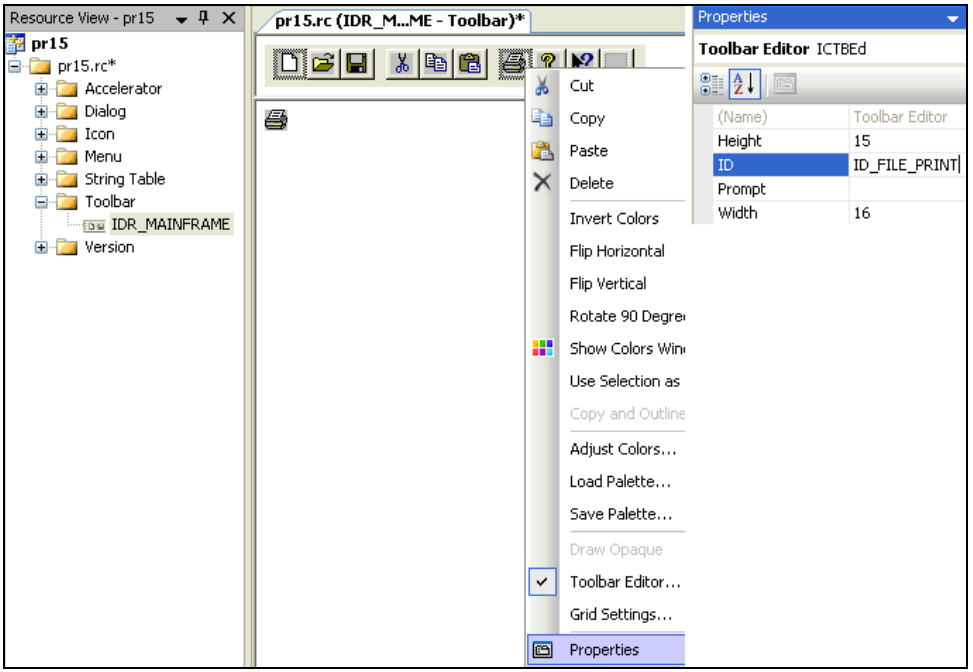
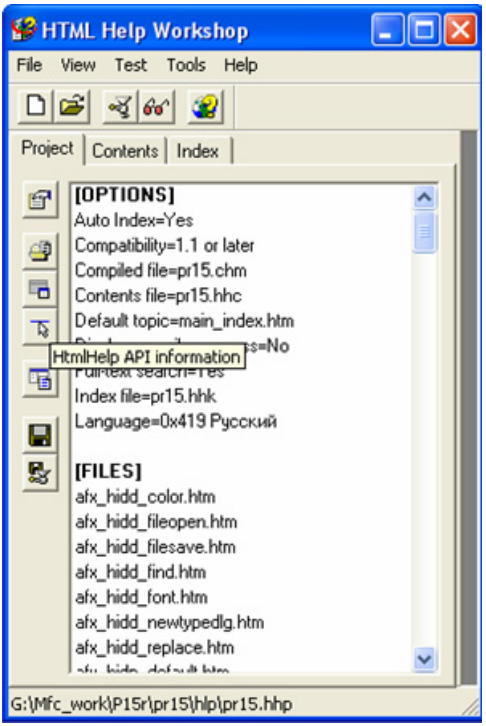
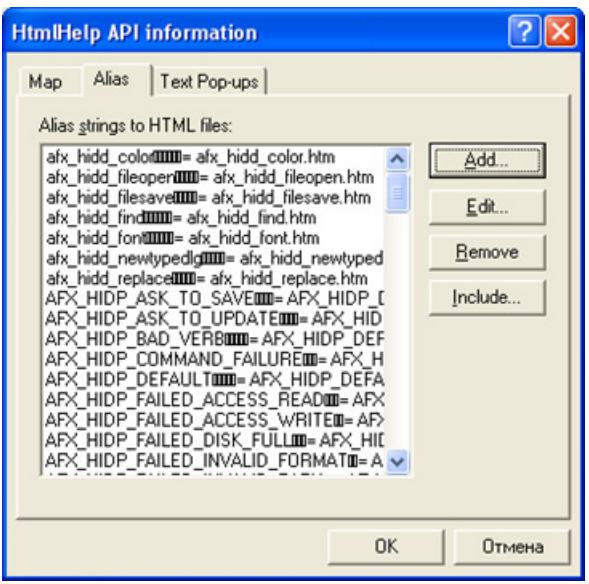


Рис. 15.51. Просмотр идентификатора кнопки печати

Чтобы не создавать новое меню (или кнопку) на панель инструментов, но научиться создавать контекстную справку, добавим ее для кнопки печати (подготовленную ранее страницу **my_page1** из файла my_file1.htm). Сначала надо посмотреть идентификатор этой кнопки `ID_FILE_PRINT` и, желательно, скопировать его в буфер (рис. 15.51).



а



б

Рис. 15.52. Создание контекстной справки с помощью кнопки на панели инструментов (*а*) и окно для добавления контекстной справки (*б*)

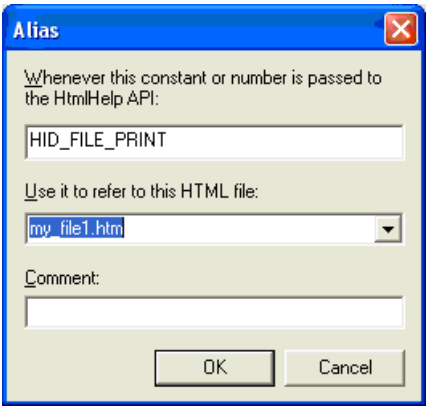


Рис. 15.53. Задание параметров контекстной справки

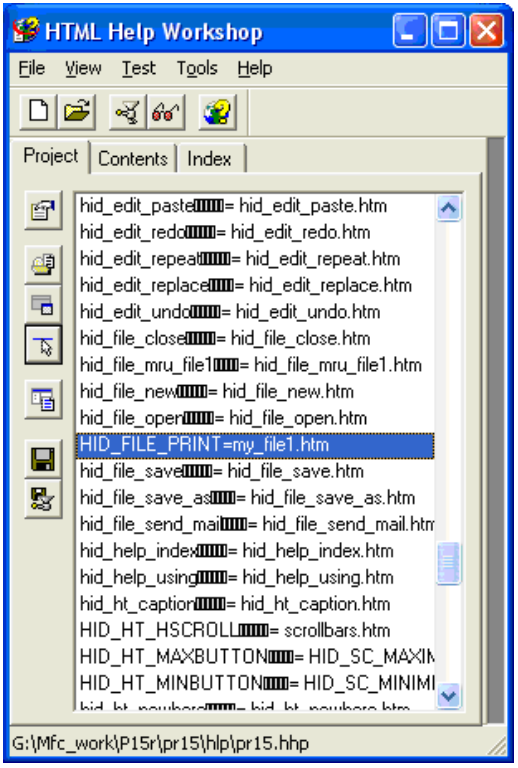


Рис. 15.54. Просмотр добавленной в проект контекстной справки

В редакторе HTML-справки на вкладке **Project** на левой панели инструментов надо выбрать четвертую сверху кнопку **HtmlHelp Api information** (рис. 15.52, *а*). В открывшемся окне **HtmlHelp API information** выберите вкладку **Alias** и нажмите кнопку **Add** (рис. 15.52, *б*). В окне **Alias** в первом поле нужно ввести идентификатор объекта, для которого будет выдаваться контекстная справка (можно вставить из буфера) и перед ним (без пробелов) добавить символ `'H'` (или `'h'`): `HID_FILE_PRINT` (рис. 15.53). В списке **Use it to refer to this HTML file** выберите файл, соответствующий странице справки: my_file1.htm, и нажмите кнопку **OK**. Проверьте (на вкладке **Project**), что в секции `[ALIAS]` появилась новая запись (рис. 15.54). Сохраните проект справки (**File | Save Project**), выйдите из редактора справки, заново *постройте* проект **pr15**, выполнив команду **Build | Rebuild Solution** (рис. 15.55), и запустите приложение. Теперь при выборе контекстной справки (с помощью кнопки (\?)) для кнопки печати будет сразу открываться справка со страницей **my_page1** (рис. 15.56).

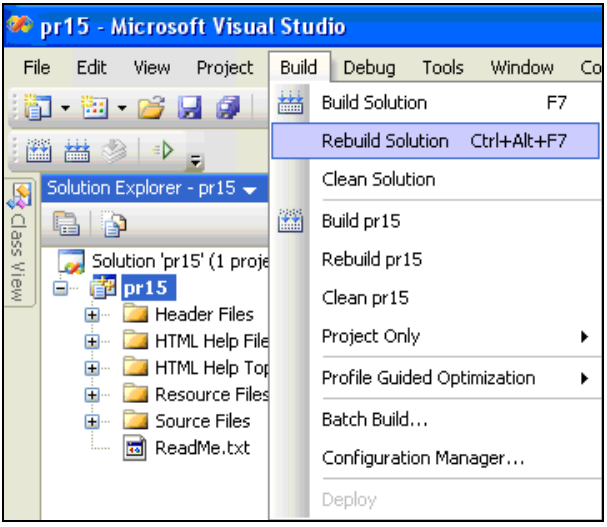


Рис. 15.55. Повторное построение проекта для правильного подключения справки

Примечание

Если открыть файл pr15.hpp в простом текстовом редакторе (можно прямо в редакторе Visual Studio, дважды щелкнув по файлу в окне **Solution**

Explorer), то будет видно, что секция `[ALIAS]` выровнена с помощью табуляторов:

```
[ALIAS]
afx_hidd_color          = afx_hidd_color.htm
afx_hidd_fileopen       = afx_hidd_fileopen.htm
afx_hidd_filesave       = afx_hidd_filesave.htm
```

Эти табуляторы отображаются в окне **HtmlHelp Api information** как символы '□' (рис. 15.52).

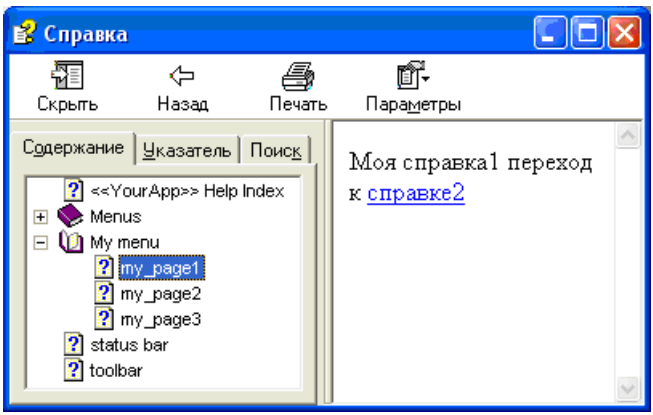


Рис. 15.56. Результат работы контекстной справки

15.1.4. Подготовка справки с помощью Microsoft Word

Если вы не владеете в достаточной мере HTML, то можно заполнять файлы справки в стандартном редакторе Microsoft Word. Для этого нужно подготовить файлы htm и пункты справки, как было рассказано в *разд. 15.1.3*. Сохраните проект справки и выйдите из редактора HTML. Откройте файл my_file1.htm в редакторе Microsoft Word. Выберите меню **Файл | Сохранить как**. Лучше сохраните его не как **Веб-страницу**, а как **Веб-страницу с фильтром** (если в вашей версии Microsoft Word есть такой тип файлов). Тогда файл справки будет меньше и попроще, но все, что надо для работы со справкой, там останется (рис. 15.57 и 15.58).

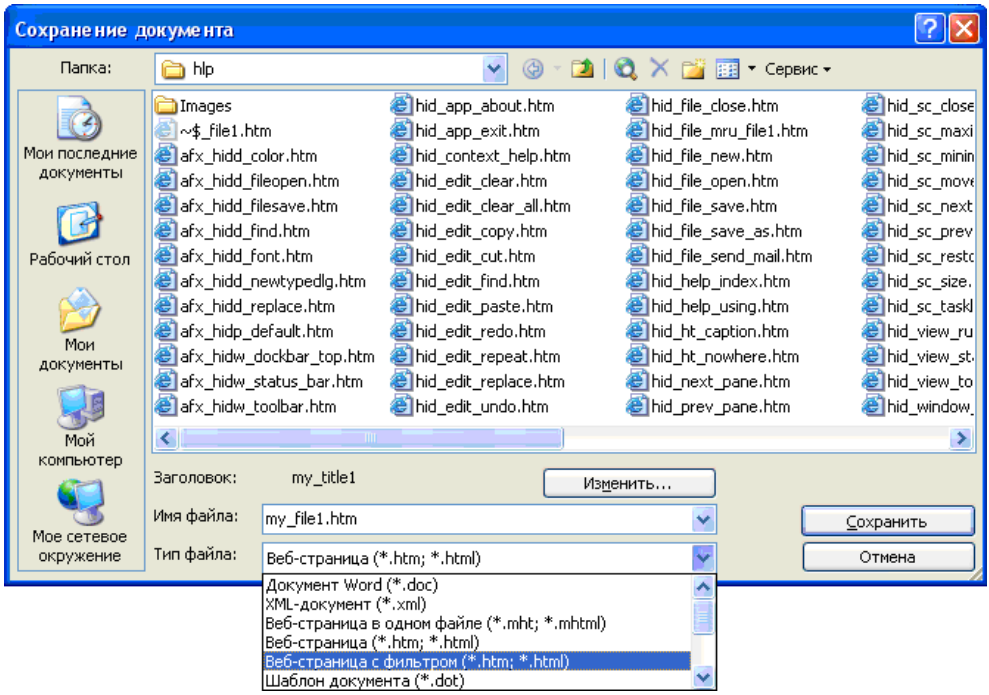


Рис. 15.57. Сохранение файла справки в редакторе Microsoft Word

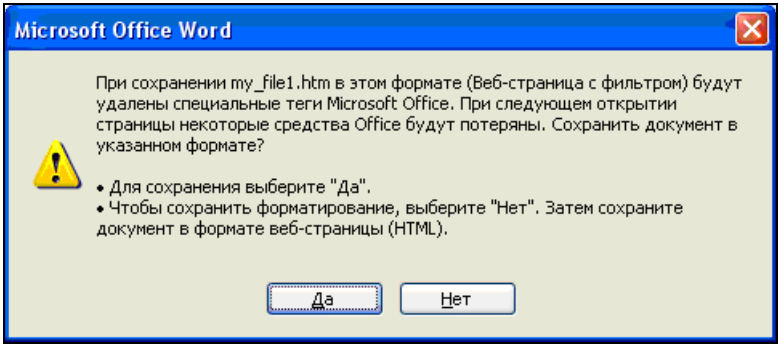


Рис. 15.58. Предупреждение при сохранении файла справки в редакторе Microsoft Word

Теперь можно набирать текст справки. Для того чтобы добавить гиперссылку, надо поставить курсор в нужное место текста, вызвать контекстное меню правой кнопкой мыши и выполнить команду **Гиперссылка** (рис. 15.59).

В открывшемся окне **Добавление гиперссылки** надо (рис. 15.60):

- в поле **Текст** ввести `к странице2` (текст, который будет выделен синим цветом и подчеркнут);
- выбрать файл со страницей, куда надо переходить по этой ссылке: my_file2.htm;
- если нажать кнопку **Подсказка**, то можно ввести текст всплывающей подсказки (рис. 15.61), который будет появляться, когда пользователь наведет курсор на гиперссылку, и нажать кнопку **OK**;
- в окне **Добавление гиперссылки** нажать кнопку **OK**.

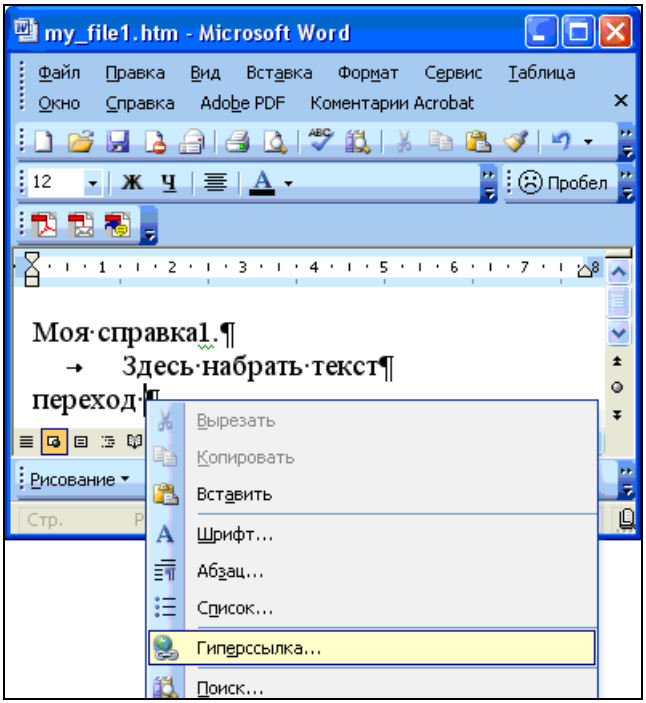


Рис. 15.59. Добавление гиперссылки в текст справки

Сохраните файл подготовленной страницы (рис. 15.62) и выйдите из редактора. Постройте весь проект заново и запустите программу (рис. 15.63).

Если при создании проекта вы забыли включить создание справки, то надо создать аналогичный проект со справкой (например, с именем **pr0**, тогда бу-

дет использоваться файл pr0.chm). Подготовьте и откомпилируйте в нем справку. Затем скопируйте каталог hlp в свой проект. В своем проекте создайте новый пункт меню для вызова справки и добавьте его обработку в класс приложения, как показано в листинге 15.11.

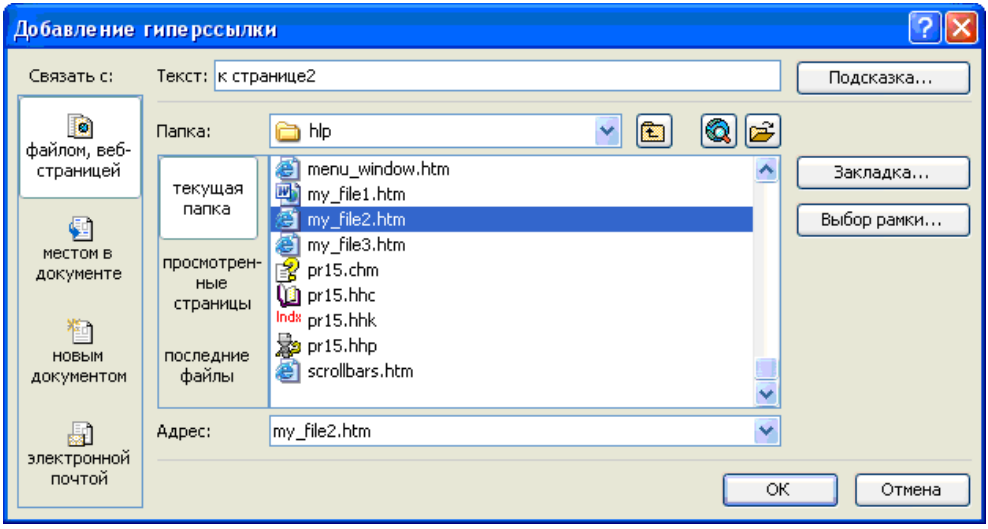


Рис. 15.60. Выбор параметров гиперссылки

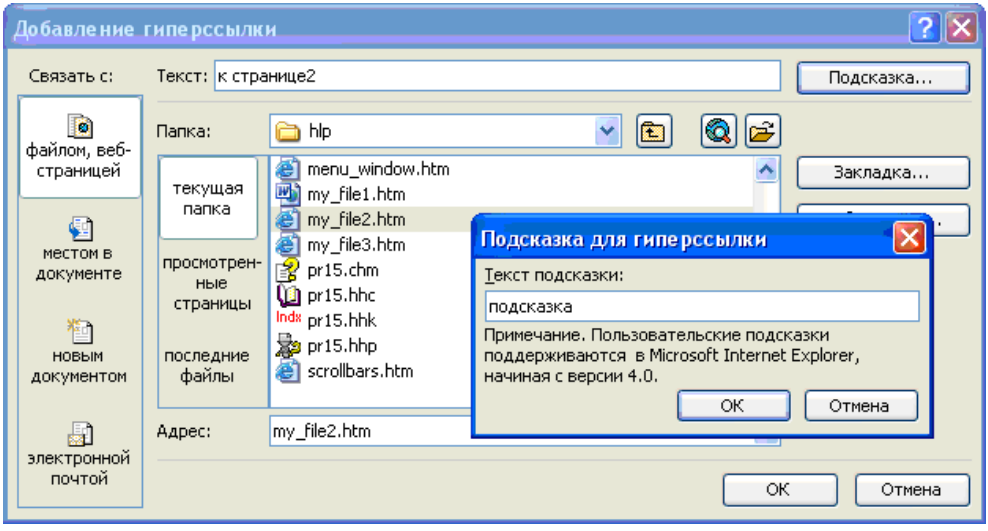


Рис. 15.61. Добавление всплывающей подсказки для гиперссылки

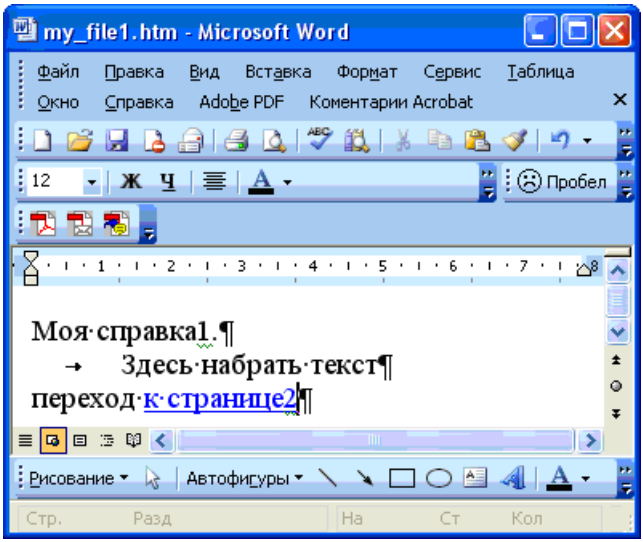


Рис. 15.62. Подготовленная в Microsoft Word страница справки

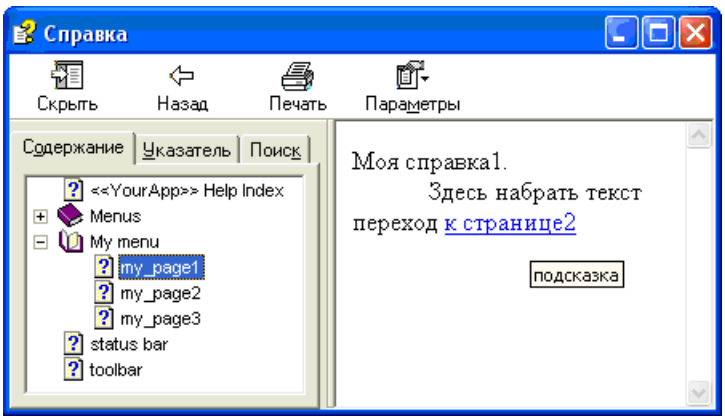


Рис. 15.63. Работа справки, подготовленной в редакторе Microsoft Word

Листинг 15.11. Самостоятельное подключение справки к проекту

```
// ...
void Cpr100App::OnHelpMyHlp()
{
    // TODO: Add your command handler code here
```

```
    ShellExecute(this->m_pMainWnd->operator HWND(), // Дескриптор
                                                    // родительского окна
                 "open",                            // Действие с файлом
                 "pr0.chm",                         // С каким файлом
                  0,                                // Дополнительные параметры
                  "hlp",                            // Каталог, где находится файл
                  SW_SHOWNORMAL);                   // Режим отображения
}
```

15.1.5. Некоторые полезные сведения о языке HTML

Далее приведем некоторые полезные дескрипторы, которые могут пригодиться при создании файлов справки.

Работа с абзацем

<P>...</P> — создание нового абзаца.

Align — выравнивание абзаца.

**
** — переход на новую строку.

<NOBR>...</NOBR> — запрещение перехода на следующую строку.

<PRE>...</PRE> — сохранение разметок табуляции (иначе они не работают, заменяются простым пробелом).

<CENTER>...</CENTER> — центрирование текста.

<BLOCKQUOTE>...</BLOCKQUOTE> — визуальное выделение абзаца отступами слева и справа.

Пример работы с абзацем приведен в листинге 15.12 и показан на рис. 15.64.

Листинг 15.12. Работа с абзацем

```
<BODY>
   <P>Новый абзац</P>
   <P Align="left">Выравнивание абзаца по левой границе</P>
   <P Align="right">Выравнивание абзаца по правой границе</P>
```

```
    <P Align="center">Выравнивание абзаца по центру</P>
    <P Align="djustify">Выравнивание абзаца по ширине</P>
    <P>Переход на <BR>новую строку</P>
    <P><NOBR><PRE>Новый абзац                    длинный, но его нельзя
       переносить</PRE></NOBR></P>
    <P>Текст простой<CENTER>центр </CENTER>продолжение
       текста</P>
    <P><BLOCKQUOTE>Этот абзац с отступом слева и справа,
       для наглядности</BLOCKQUOTE></P>
</BODY>
```

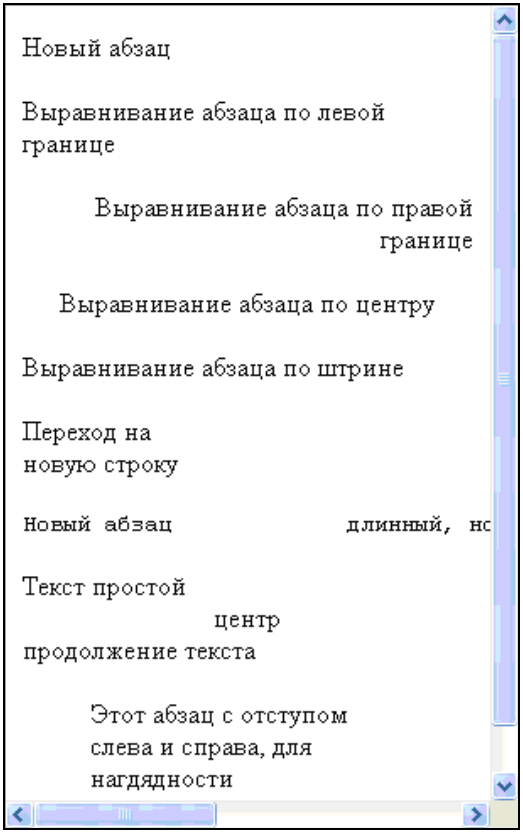


Рис. 15.64. Работа с абзацем

Работа со шрифтом

<I>...</I> — выделение курсивом.

<B>...</B> — полужирный шрифт.

<U>...</U> — подчеркивание.

Заголовки выделяются следующими дескрипторами:

- **<H1>...</H1>** — самый большой;
- **<H2>...</H2>**;
- **<H3>...</H3>**;
- **<H4>...</H4>**;
- **<H5>...</H5>**;
- **<H6>...</H6>** — самый маленький.

Размер шрифта (может принимать значения от 1 до 7):

<FONT Size="*Размер шрифта***">...</FONT>**

Специальные символы и их коды приведены в табл. 15.1.

Таблица 15.1. *Специальные символы*

Символ	Цифровой код	Именной код
"	&#34;	&quot;
&	&#38;	&amp;
<	&#60;	&lt;
>	&#62;	&gt;
§	&#167;	&sect;
©	&#169;	&copy;
«	&#171;	&laquo;
®	&#174;	&reg;
±	&#177;	&plusm;
»	&#187;	&raquo;
÷	&#247;	&divide;
неразрывный пробел		

Пример работы со шрифтами приведен в листинге 15.13 и показан на рис. 15.65.

Листинг 15.13. Работа со шрифтами

```
<BODY>
    Текст <I>курсив</I>, <B>полужирный</B>, <U>подчеркивание</U>
    <H1>Заголовок 1</H1>
    <H3>Заголовок 3</H3>
    <H6 Align="center">Заголовок 6 по центру</H6>
    <FONT Size="7">Шрифт 7</FONT><BR>
    Текст &#34;в кавычках&quot; простой
</BODY>
```

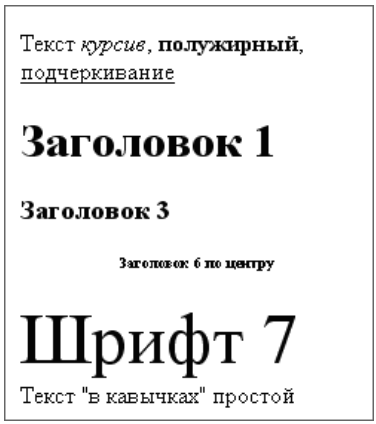


Рис. 15.65. Работа со шрифтами

Добавление рисунка

<IMG SRC="*имя файла***">**

Имя файла может задаваться следующим образом:

- *диск*:*каталог**имя_файла* — полный путь к файлу (например, C:\Images\image.gif);
- *имя_файла* — файл находится в каталоге hlp (C:\pr15\hlp\image.gif);

- *каталог/имя_файла* — файл находится в каталоге, вложенном в hlp (`C:\pr15\hlp\Images\image.gif`);
- *../имя_файла* — файл находится в каталоге выше hlp (`C:\pr15\image.gif`).

Если надо добавить рисунок несколько раз, то на диске должно быть соответствующее количество его копий.

Width — задание размера рисунка в процентах относительно размера окна. При изменении размера окна рисунок будет уменьшаться или увеличиваться.

Align — расположение рисунка.

Пример работы с рисунком приведен в листинге 15.14 и показан на рис. 15.66.

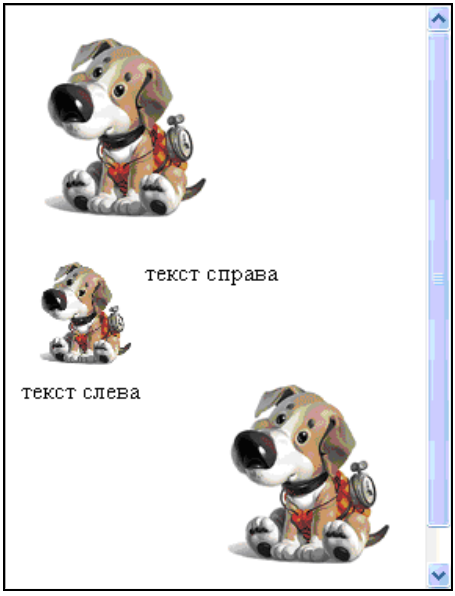


Рис. 15.66. Работа с рисунком

Листинг 15.14. Работа с рисунком

```
<BODY>
   <IMG SRC="Images/dog.gif"><BR>
   <P><IMG SRC="Images/dog_copy1.gif" Align="left" Width=30%">
```

```
      текст справа<P>
   <P> </P>
   <P><IMG SRC="Images/dog_copy2.gif" Align="right" >
      текст слева<P>
</BODY>
```

Добавление гиперссылки

<A Href="*имя файла***">**текст ссылки**</A>** — создание гиперссылки.

Title — текст всплывающей подсказки.

Пример работы с гиперссылками приведен в листинге 15.15 и показан на рис. 15.67.

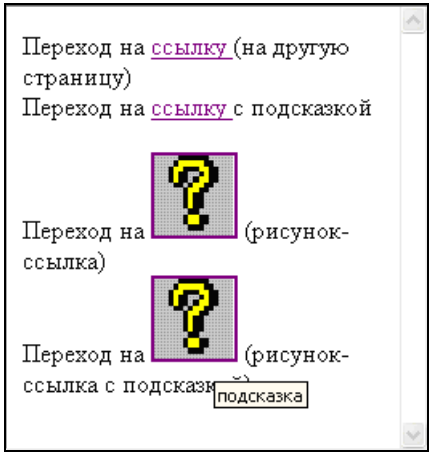


Рис. 15.67. Работа с гиперссылкой

Листинг 15.15. Работа с гиперссылками

```
<BODY>
   <P>
      Переход на <A Href="pfile2.htm">ссылку </A>(на другую страницу)<BR>
      Переход на <A Href="pfile2.htm" Title="подсказка" > ссылку </A>с
         подсказкой
   </P>
```

```
   <P>
      Переход на <A Href="pfile2.htm" > <IMG SRC="Images/quest.gif"></A>
         (рисунок-ссылка)<BR>
      Переход на <A Href="pfile2.htm" Title="подсказка" >
         <IMG SRC="Images/quest.gif"></A>
        (рисунок-ссылка с подсказкой)<BR>
   </P>
</BODY>
```

Бегущая строка

<MARQUEE>...</MARQUEE> — создание бегущей строки.

Bgcolor — цвет страницы или цвет фона бегущей строки.

Text — цвет текста страницы.

Color — цвет текста.

Возможные цвета (цвет может задаваться шестнадцатеричным кодом или именем):

- белый — **white** (#FFFFFF);
- бирюзовый — **teal** (#008080)
- желтый — **yellow** (#FFFF00);
- зеленый — **green** (#008000);
- золотой — **gold** (#FFD700);
- красно-коричневый — **maroon** (#800000);
- красный — **red** (#FF0000);
- лимонный — **lime** (#00FF00);
- морской волны — **aqua** (#00FFFF);
- оливковый — **olive** (#808000);
- пурпурный — **purple** (#800080);
- серебряный — **silver** (#C0C0C0);
- серый — **gray** (#808080);
- синий — **blue** (#0000FF);

- темно-голубой — **navy** (#000080);
- фуксиновый — **fuchsia** (#FF00FF);
- черный — **black** (#000000).

Пример работы с бегущей строкой приведен в листинге 15.16 и показан на рис. 15.68.

Листинг 15.16. Работа с бегущей строкой

```
<BODY Bgcolor="gray" Text="yellow">
   Примеры бегущей строки <BR><BR>
   <P>
      <MARQUEE>бегущая строка</MARQUEE>
   </P>
   <P>
      <MARQUEE ><FONT Size="7">шрифт</FONT></MARQUEE>
   </P>
   <P>
      <MARQUEE Bgcolor="aqua" ><FONT Size="7" Color="purple">
         цвет</FONT></MARQUEE>
   </P>
</BODY>
```

Рис. 15.68. Работа с бегущей строкой

Нумерованный список

<OL>...</OL> — нумерованный список.

<LI>...</LI> — элемент списка.

Start — начальное значение для нумерации списка.

Type — вид нумерации, может принимать следующие значения:

- "1" — 1, 2, 3, 4, ... (по умолчанию);
- "I" — I, II, III, IV, ...;
- "a" — a, b, c, d, ...;
- "A" — A, B, C, D, ...

Value — номер текущего элемента.

Пример работы с нумерованным списком приведен в листинге 15.17 и показан на рис. 15.69.

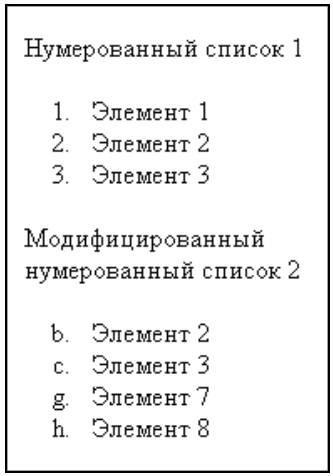


Рис. 15.69. Работа с нумерованным списком

Листинг 15.17. Работа с нумерованным списком

```
<BODY>
   Нумерованный список 1
   <OL>
```

```
      <LI>Элемент 1</LI>
      <LI>Элемент 2</LI>
      <LI>Элемент 3</LI>
   </OL>
   Модифицированный нумерованный список 2
   <OL Start="2" Type="a">
      <LI>Элемент 2 </LI>
      <LI>Элемент 3</LI>
      <LI Value="7">Элемент 7</LI>
      <LI>Элемент 8</LI>
   </OL>
</BODY>
```

Маркированный список

<UL>...</UL> — маркированный список.

Type — тип маркера всего списка или текущего элемента:

- "disk" — черный кружок (по умолчанию);
- "circle" — белый кружок;
- "square" — черный квадратик.

Пример работы с маркированным списком приведен в листинге 15.18 и показан на рис. 15.70.

Листинг 15.18. Работа с маркированным списком

```
<BODY >
   Маркированный список 1
   <UL>
      <LI>Элемент 1</LI>
      <LI>Элемент 2</LI>
      <LI>Элемент 3</LI>
   </UL>
   Модифицированный маркированный список 2
   <UL Type="circle">
```

```
      <LI>Элемент 1 </LI>
      <LI>Элемент 2</LI>
      <LI Type="square">Элемент 3</LI>
   </UL>
</BODY>
```

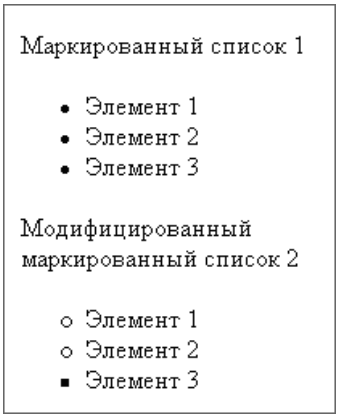


Рис. 15.70. Работа с маркированным списком

Пример работы с многоуровневым списком приведен в листинге 15.19 и показан на рис. 15.71.

Листинг 15.19. Работа с многоуровневым списком

```
<BODY >
   Многоуровневый список
   <OL>
      <LI>Нумерованный список</LI>
         <OL>
            <LI>Элемент 1</LI>
            <LI>Элемент 2</LI>
            <LI>Элемент 3</LI>
         </OL>
      <LI>Маркированный список</LI>
         <UL>
```

```
            <LI>Элемент 1</LI>
            <LI>Элемент 2</LI>
            <LI>Элемент 3</LI>
        </UL>
    </OL>
</BODY>
```

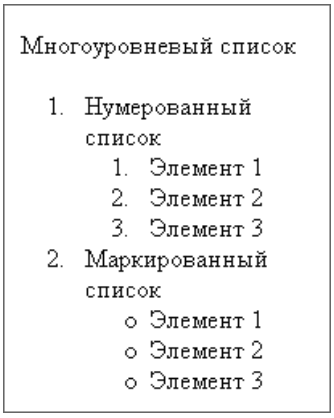


Рис. 15.71. Работа с многоуровневым списком

Приложение

Описание компакт-диска

Компакт-диск включает в себя примеры приложений (табл. П1.1), описанных в книге. Для запуска проекта в Visual Studio надо запустить файл *имя_проекта*.sln. Имена проектов соответствуют главам (*глава 1* — проект **pr1**, *глава 2* — **pr2** и т. д.). Исключение составляет только проект к *главе 1*. Так как эта глава является начальной, то, чтобы не вносить путаницу в освоение материала, к ней прилагаются два проекта: pr1\pr1.sln (первоначальный шаблон, построенный мастером) и pr1_new\pr1\pr1.sln (конечный проект, со всеми внесенными изменениями, описанными в *главе 1*). Для каждого проекта в папке Debug есть исполняемый файл *имя_проекта*.exe. Исполняемый файл был построен с использованием динамически подключаемых библиотек и будет запускаться только на компьютерах, где установлен пакет Microsoft Visual Studio 2005 или Microsoft Visual Studio 2008. Для тех, у кого нет Microsoft Visual Studio, в папке all_exe есть все исполняемые файлы проектов, запускаемые с любого компьютера с системой Windows XP.

Таблица П1.1. *Состав компакт-диска*

Папка на компакт-диске	Глава	Описание
pr1	1	Шаблон простого SDI-приложения, построенного мастером MFC AppWizard
pr1_new	1	Изменение заголовка, размера и цвета окон, вида курсора, иконки и меню. Использование горячих клавиш (акселератов) и клавиш альтернативного выбора пунктов меню
pr2	2	Вывод текста и графических объектов в окне представления. Создание и выбор шрифтов. Рисование геометрических фигур. Работа с цветом

Таблица П1.1 *(продолжение)*

Папка на компакт-диске	Глава	Описание
pr3	3	Использование дочерних объектов (кнопок и готовых рисунков) в окне представления. Динамическое изменение вида курсора
pr4	4	Работа с меню приложения. Добавление новых пунктов меню. Динамическое изменение меню. Обработка сообщений мыши. Создание и работа с контекстным меню
pr5	5	Использование виртуального окна для перерисовки и масштабирования изображения. Использование линейки прокрутки. Обработка нажатия клавиш. Создание дочернего окна
pr6	6	Создание с диалогового окна и работа с элементами управления **Button**, **Check Box**, **Edit Control**, **Combo Box**, **List Box**, **Radio Button**, **Static Text**, **Group Box**
pr7	7	Работа с элементами управления **Picture Control**, **Horizontal Scroll Bar**, **Slider Control**, **Spin Control**, **Progress Control**, **Hot Key**, **List Control**, **Tree Control**
pr8	8	Работа с элементами управления **Tab Control**, **Animation Control**, **Rich Edit 2.0 Control**, **Date Time Picker**, **Month Calendar Control**, **IP Address Control**, **Extended Combo box**. Использование страницы свойств **PropertySheet**
pr9	9	Работа с панелью инструментов и строкой статуса. Динамическое изменение панели инструментов. Добавление новой панели инструментов. Добавление новых полей в строку статуса. Динамическое изменение полей строки статуса. Подсказки для меню в строке статуса и всплывающие подсказки для кнопок панели инструментов
pr10	10	Работа простого SDI-приложения с использованием архитектуры документ/представления. Возможные варианты связи между объектами приложения
pr11	11	Рисование графических изображений в окне представления с помощью мыши. Загрузка и сохранение графических изображений на диске с использованием метафайла

Таблица П1.1 *(окончание)*

Папка на компакт-диске	Глава	Описание
pr12	12	Рисование графических изображений в окне представления с помощью мыши. Загрузка и сохранение графических изображений на диске с использованием архива. Использование стандартного окна диалога для выбора файла
pr13	13	Разделение окна представления на две области: верхняя — для работы с графическими данными; нижняя — для работы с текстовыми данными. Обмен данными между областями
pr14	14	Работа простого MDI-приложения с использованием архитектуры документ/представления. Работа с несколькими шаблонами документов: графическим и текстовым. Обмен данными между документами разных шаблонов
pr15	15	Работа со справочной системой приложения. Добавление новых разделов справки и ключевых слов. Использование гиперссылок на страницах справки. Работа контекстной справки
all_exe	—	Исполняемые файлы для всех проектов книги со статически подключенными библиотеками MFC

Предметный указатель

C

H

M

V

W

B

Г

Д

Ж

З

И

К

M

H

O

П

Р

C

T

Ф