НИКИТА КУЛЬТИН

Visual Basic

ОСВОЙ САМОСТОЯТЕЛЬНО

+CD

bhv

Никита Культин

Visual Basic

ОСВОЙ САМОСТОЯТЕЛЬНО

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.068+800.92VB
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Visual Basic. Освой самостоятельно. — СПб.: БХВ-Петербург, 2005. — 480 с.: ил.

ISBN 5-94157-516-5

Рассмотрен язык программирования Visual Basic. Изложены принципы визуального проектирования и событийного программирования. На конкретных примерах показаны возможности среды разработки, методика создания программы и тонкости процесса программирования. Большое внимание уделено программированию графики, баз данных, игр. Приведено описание базовых компонентов и наиболее часто используемых функций. На прилагаемом компакт-диске находятся исходные тексты программ.

Для начинающих программистов

УДК 681.3.068+80.92VB
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Римма Смоляк*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.05.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 30.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 5-94157-516-5

Оглавление

Предисловие

Visual Basic — что это?

В последнее время возрос интерес к программированию, что связано с развитием и внедрением в повседневную жизнь информационно-коммуникационных технологий. Если человек имеет дело с компьютером, то рано или поздно у него возникает желание, а иногда и необходимость программировать.

Среди пользователей персональных компьютеров в настоящее время наиболее популярно семейство операционных систем Windows и, естественно, что тот, кто собирается программировать, стремится писать программы, которые будут работать в этих системах.

Раньше рядовому программисту оставалось только мечтать о создании собственных программ, работающих в среде Windows, т. к. средства разработки были явно ориентированы на профессионалов, обладающих серьезными знаниями и опытом.

Бурное развитие вычислительной техники, потребность в эффективных средствах разработки программного обеспечения привели к появлению систем программирования, ориентированных на так называемую "быструю разработку", пионером среди которых был пакет Microsoft Visual Basic.

В основе систем быстрой разработки (RAD-систем, Rapid Application Development — среда быстрой разработки приложений) лежит технология визуального проектирования и событийного программирования. Суть этой технологии заключается в том, что среда разработки берет на себя бóльшую часть рутинной работы, оставляя программисту работу по конструированию диалоговых окон и функций обработки событий. Производительность программиста при использовании RAD-систем — фантастическая!

Microsoft Visual Basic — это среда быстрой разработки, в которой в качестве языка программирования используется Visual Basic. В основе идеологии Visual Basic лежит технология визуального проектирования и методология объектно-ориентированного событийного программирования.

В настоящее время наиболее широко используется шестая версия пакета — Microsoft Visual Basic 6.0, которая позволяет создавать самые различные программы: от простейших однооконных приложений до программ управления распределенными базами данных.

Microsoft Visual Basic может работать в среде операционных систем от Windows 98 до Windows XP. Особых требований, по современным меркам, к ресурсам компьютера пакет не предъявляет.

Об этой книге

В книге, которая посвящена программированию в конкретной среде, необходим баланс между тремя линиями:

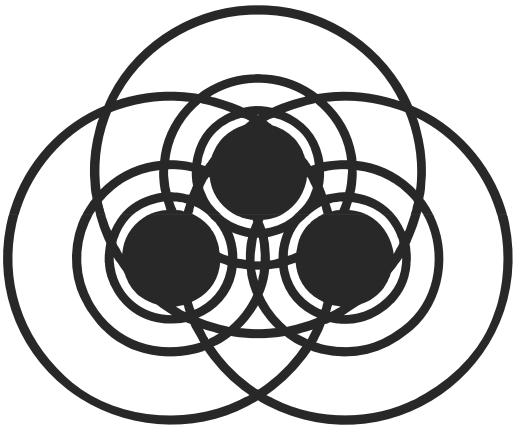
- язык программирования;
- техника и технология программирования (программирование как таковое);
- среда разработки.

Чтобы описать процесс разработки программы, объяснить, как она работает, нужно оперировать такими терминами, как *объект*, *событие*, *свойство*, понимание которых на начальном этапе изучения программирования весьма проблематично. Как поступить? Сначала дать описание языка, а затем приступить к описанию среды разработки и процесса программирования в Visual Basic? Очевидно, что это не лучший вариант. Поэтому при изложении материала принят подход (метод), который можно назвать "от задачи". Суть метода заключается в том, что берется конкретная задача и на ее примере рассматриваются определенная технология, возможности среды разработки и особенности языка программирования, необходимые для решения этой конкретной задачи.

Книга, которую вы держите в руках, — это не описание языка программирования и среды разработки Visual Basic. Это руководство (учебное пособие) по программированию в Microsoft Visual Basic. В нем рассмотрена вся цепочка, весь процесс создания программы: от разработки диалогового окна и процедур обработки событий до создания справочной системы.

Цель этой книги — научить программировать в Microsoft Visual Basic 6.0, т. е. создавать законченные программы различного назначения: от простых однооконных приложений до вполне профессиональных программ работы с базами данных.

Научиться программировать можно только в процессе решения конкретных задач. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Не занимайтесь просто чтением примеров, реализуйте их с помощью вашего компьютера. Не бойтесь экспериментировать, вносите изменения в программы — чем больше сделаете самостоятельно, тем большему вы научитесь!

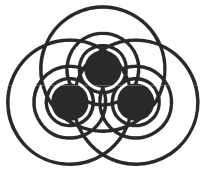


ЧАСТЬ I

СРЕДА РАЗРАБОТКИ MICROSOFT VISUAL BASIC

Первая часть книги знакомит читателя со средой разработки Microsoft Visual Basic. На примере программы **Конвертор**, которая позволяет пересчитать цену из долларов в рубли, демонстрируется процесс создания программы (технология визуального проектирования и событийного программирования), вводятся основные понятия и термины объектно-ориентированного программирования. Также в первой части приведено описание базовых и дополнительных компонентов.

Глава 1



Установка

Устанавливается Microsoft Visual Basic обычным образом. Программа установки стартует автоматически.

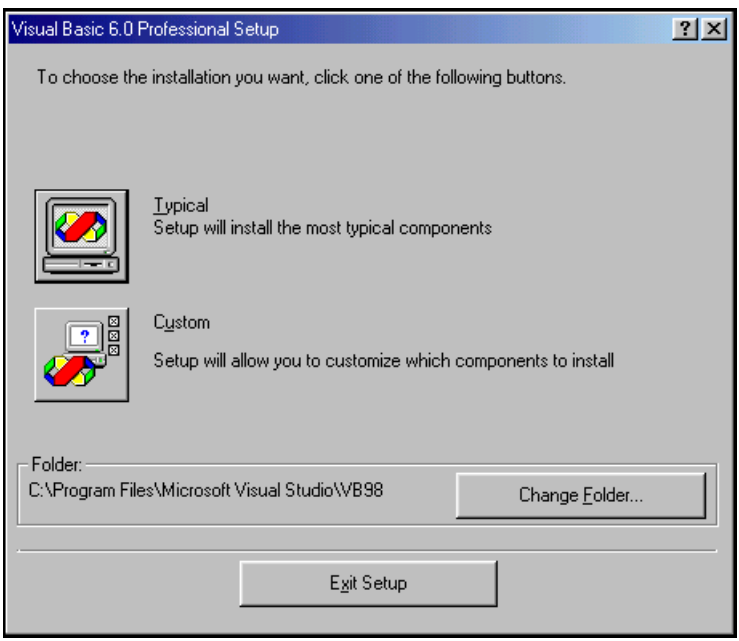


Рис. 1.1. Диалоговое окно выбора варианта установки

После того как установочный CD будет помещен в дисковод, надо ввести регистрационную информацию и выбрать каталог, куда предполагается установить Visual Basic, а затем выбрать вариант установки (рис. 1.1):

- **Typical** (Обычный). В этом случае с установочного CD на жесткий диск компьютера будут скопированы все основные компоненты Microsoft Visual Basic (лучше выбрать именно этот вариант установки);
- **Custom** (Выборочный, определяемый программистом). Этот вариант позволяет программисту самому выбрать компоненты, которые надо установить. Рекомендуется в том случае, когда на диске компьютера недостаточно свободного места для полной установки.

По окончании процесса копирования файлов программа установки предлагает перезапустить компьютер, а после перезагрузки поместить в CD-дисковод компакт-диск, на котором находится MSDN Library (Microsoft Developer Network — справочная система Microsoft Visual Studio). Возможны три варианта установки MSDN:

- **Typical** (Обычный). Предполагается, что на жесткий диск компьютера будет скопировано ядро справочной системы (порядка 60 Мбайт), а вся остальная информация будет загружаться по мере необходимости с компакт-диска;
- **Custom** (Выборочный). Этот вариант позволяет программисту выбрать разделы справочной информации, которые будут скопированы на диск компьютера, остальные разделы также будут доступны, но только тогда, когда компакт-диск с MSDN находится в дисководе;
- **Full** (Полный). В этом случае вся справочная информация (порядка 800 Мбайт) будет скопирована на жесткий диск.

Если программист предполагает работать только с Visual Basic, рекомендуется выбрать выборочный (**Custom**) вариант установки MSDN (рис. 1.2), а затем разделы: **VB Documentation** (Документация Visual Basic), **VB Product Samples** (Примеры Visual Basic) и **VS Shared Documentation** (Общая документация Visual Studio).

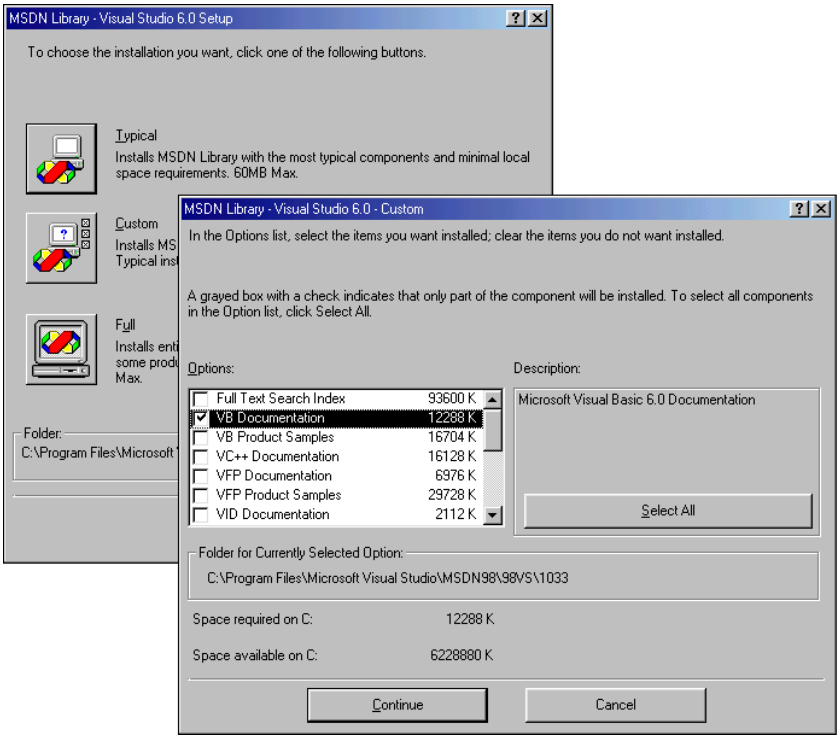


Рис. 1.2. Установка MSDN-справочной системы

Начало работы

Запускается Visual Basic обычным образом, т. е. выбором команды **Пуск ▸ Программы ▸ Microsoft Visual Basic 6.0 ▸ Microsoft Visual Basic 6.0** (рис. 1.3).

Если Visual Basic запущен сразу после установки, то на фоне главного окна отображается окно **New Project** (рис. 1.4). В этом окне на вкладке **New** перечислены виды программ (проектов), которые можно создать в Visual Basic. Чтобы приступить к работе над новым *Windows-приложением*, надо выбрать **Standard EXE** и щелкнуть на кнопке **Открыть**. Если после запуска Visual Basic окно **New Project** на экране не отображается, то для того чтобы оно стало доступным, надо в меню **File** выбрать команду **New Project**.

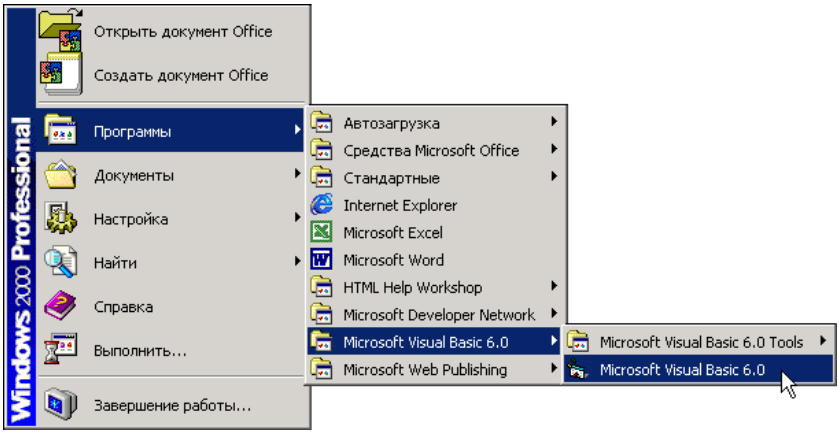


Рис. 1.3. Запуск Visual Basic

Рис. 1.4. Начало работы над новой программой (Windows-приложением)

Вид окна Visual Basic в начале работы над стандартным Windows-приложением приведен на рис. 1.5. В верхней части окна находится строка меню и стандартная панель инструментов, слева — *палитра компонентов*, в центре — окно *конструктора формы*, справа — окно *менеджера проекта*, окно *свойств* и окно *отображения положения формы*.

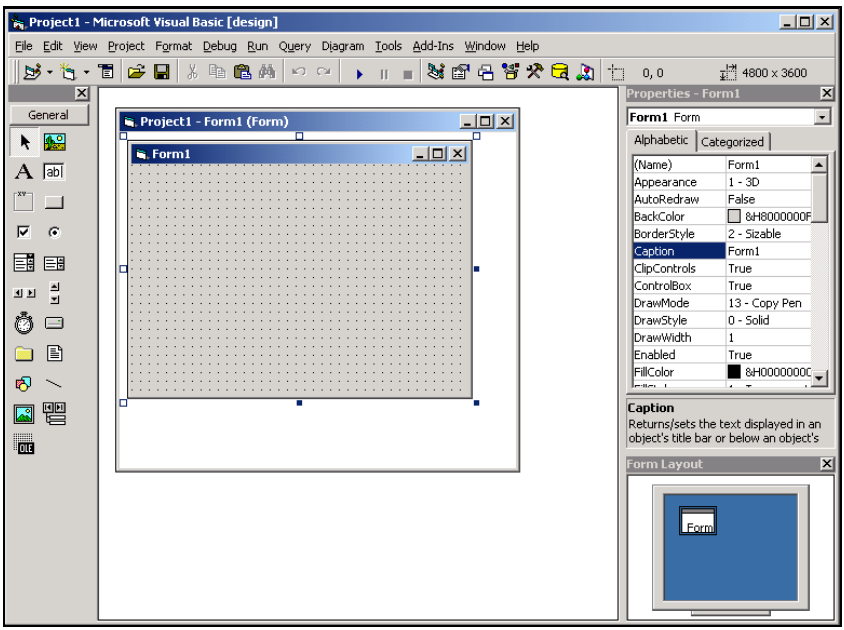


Рис. 1.5. Окно Visual Basic
в начале работы над новой программой

На стандартной панели инструментов (рис. 1.6) находятся кнопки, соответствующие наиболее часто используемым командам. Здесь же располагаются кнопки, с помощью которых можно быстро сделать доступными окна палитры компонентов, менеджера проектов, свойств и др.

В окне конструктора формы (рис. 1.7) находится *форма*, которая представляет собой заготовку окна разрабатываемого приложения.

Рис. 1.6. Стандартная панель инструментов

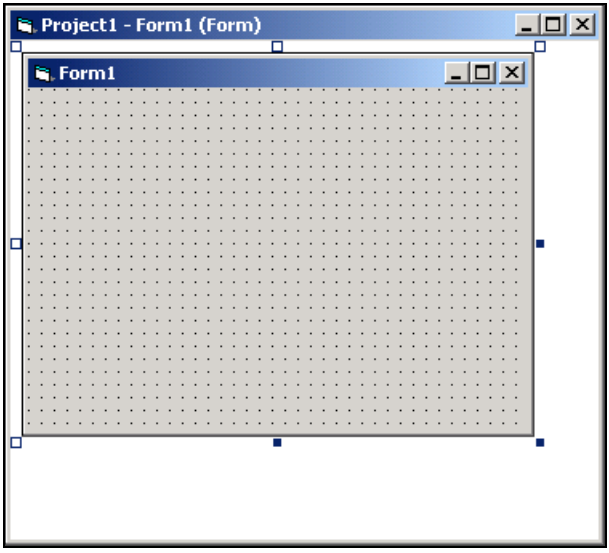


Рис. 1.7. Окно конструктора формы

В палитре компонентов (рис. 1.8) находятся значки, обозначающие *компоненты* — элементы интерфейса пользователя, которые программист может поместить на поверхность формы. Если палитра компонентов не отображается, то для того чтобы она стала доступной, надо выбрать команду **View ▸ Toolbox** или щелкнуть на соответствующей кнопке панели инструментов.

В окне менеджера проекта (рис. 1.9) отображается структура (состав) проекта, над которым в данный момент идет работа. Если окно менеджера проекта не отображается, то для того чтобы оно стало доступным, надо выбрать команду **View ▸ Project Explorer** или щелкнуть на соответствующей кнопке панели инструментов.

Рис. 1.8. Палитра компонентов (окно **Toolbox**)

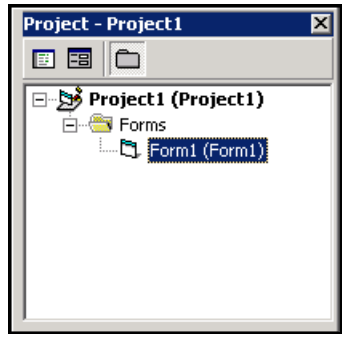


Рис. 1.9. В окне **Project** отображается структура (состав) проекта

Окно свойств (рис. 1.10) предназначено для отображения и редактирования значений свойств объектов. В терминологии визуального проектирования *объект* — это диалоговое окно или элемент интерфейса пользователя (поле ввода, командная кнопка, переключатель и др.).

Свойство — это характеристика, которая определяет (задает) внешний вид объекта. Например, значение свойства `Caption` задает заголовок формы, а свойств `Width` и `Height` — ее размер. Свойства на вкладке **Alphabetic** отображаются в алфавитном порядке, а на вкладке **Categorized** сгруппированы. Например, в группу **Appearance** объединены свойства, которые определяют вид формы (заголовок, фоновый рисунок, тип границы и т. д.), а в группу **Position** — свойства, определяющие размер и положение на экране.

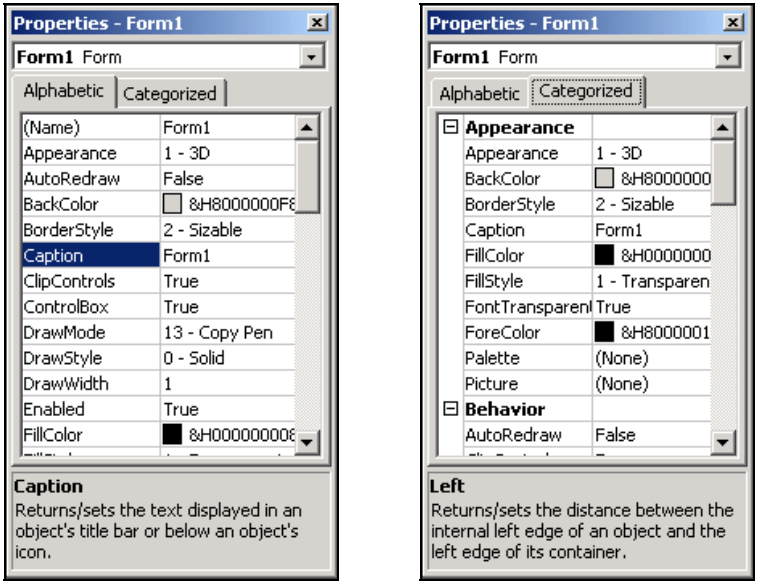
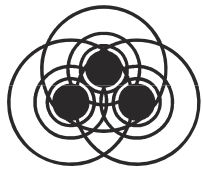


Рис. 1.10. В окне **Properties** перечислены свойства объекта и указаны их значения

Глава 2



Первый проект

Процесс создания программы (Windows-приложения) в Visual Basic рассмотрим на примере. Пусть надо создать программу, которая пересчитывает цену из долларов в рубли. Вид окна программы во время ее работы после ввода исходных данных и щелчка на кнопке **Пересчет** приведен на рис. 2.1.

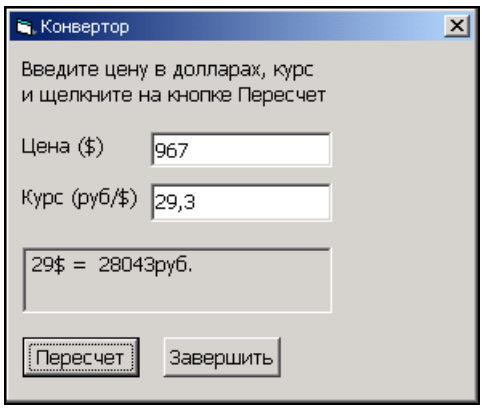


Рис. 2.1. Окно программы пересчета цены из долларов в рубли

Чтобы начать работу над новой программой или, как принято говорить, над новым проектом, запустите Visual Basic, на вкладке **New** окна **New Project** выберите **Standard EXE** и щелкните на кнопке **Открыть**. Если после запуска Visual Basic окно **New Project** не отображается, то выберите команду **File ▸ New Project**.

Форма

Работа над новым *проектом* (так принято называть программу (приложение), над которой идет работа) начинается с создания или, как иногда говорят, с настройки стартовой формы.

Стартовая форма создается путем изменения значений свойств, находящихся в окне дизайнера формы **Form1**, и добавления к этой форме необходимых компонентов (полей ввода, отображения текста, командных кнопок и т. д.).

Для просмотра и изменения значений свойств формы и ее компонентов используется окно **Properties**. В верхней части этого окна указано имя объекта, значения свойств которого отображаются в данный момент. В левой колонке окна перечислены свойства объекта, в правой — указаны значения свойств.

Некоторые свойства формы приведены в табл. 2.1.

Таблица 2.1. *Свойства формы (объекта* ***Form****)*

Свойство	Описание
`Name`	Имя формы. Используется для доступа к форме и ее компонентам
`Caption`	Текст заголовка
`Width`	Ширина формы. Задается в твипах
`Height`	Высота формы. Задается в твипах
`StartUpPosition`	Положение формы при ее появлении на экране. Форма может располагаться в центре экрана (`Center Screen`), в центре родительской формы (`Center Owner`). Положение формы могут определять значения свойств `Top` и `Left` (в этом случае значение свойства `StartUpPosition` должно быть равно `Manual`)
`Top`	Расстояние от верхней границы формы до верхней границы экрана или до верхней границы родительской формы

Таблица 2.1 *(продолжение)*

Свойство	Описание
`Left`	Расстояние от левой границы формы до левой границы экрана или до левой границы родительской формы
`Icon`	Значок (картинка) в заголовке, обозначающая системное меню
`MaxButton`	Признак наличия в заголовке окна кнопки **Развернуть**
`MinButton`	Признак наличия в заголовке окна кнопки **Свернуть**
`BorderStyle`	Стиль (вид) границы. Граница может быть: • обычной (`Sizable`); • тонкой (`Fixed Single`) (в этом случае изменить размер окна путем перемещения границы мышью нельзя); • вообще отсутствовать (`None`). Если значение свойства равно `Fixed Dialog`, то граница окна тонкая и кнопки **Развернуть** и **Свернуть** в заголовке не отображаются
`BackColor`	Цвет формы. Цвет можно задать, выбрав его из палитры или указав привязку к элементу цветовой схемы операционной системы. Во втором случае цвет определяется текущей цветовой схемой и выбранным компонентом привязки, и меняется при изменении цветовой схемы операционной системы
`ScaleMode`	Определяет (задает) единицу измерения размеров компонентов, которые находятся на поверхности формы. Размер компонентов может измеряться в твипах (Twip), пикселах (Pixel) и других единицах

Таблица 2.1 *(окончание)*

Свойство	Описание
`Font`	Шрифт, который по умолчанию используется находящимися на поверхности формы компонентами для отображения текста (например, надпись на командной кнопке, текст в поле редактирования или в поле отображения текста)

Сначала надо задать заголовок формы — изменить значение свойства `Caption` с `Form1` на `Конвертор`. Чтобы это сделать, нужно в окне **Properties** выбрать свойство `Caption` и щелкнуть мышью в поле значения свойства. В результате этих действий в поле значения свойства (после слова `Form1`) появится курсор и можно будет ввести значение свойства (рис. 2.2).

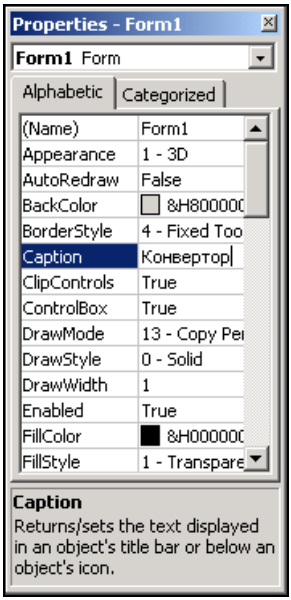


Рис. 2.2. Изменение значения свойства путем ввода строки

Следует обратить внимание, что ширина и высота формы измеряются в специальных единицах — твипах. Задавать значения свойств `Width` и `Height` в твипах неудобно. Гораздо проще захватить один из находящихся на границе формы черных квадратиков и переместить границу (вертикальную, горизонтальную или обе сразу) в нужном направлении (рис. 2.3). По окончании перемещения границы значения свойств `Width` и `Height` автоматически изменятся и будут соответствовать установленному размеру формы.

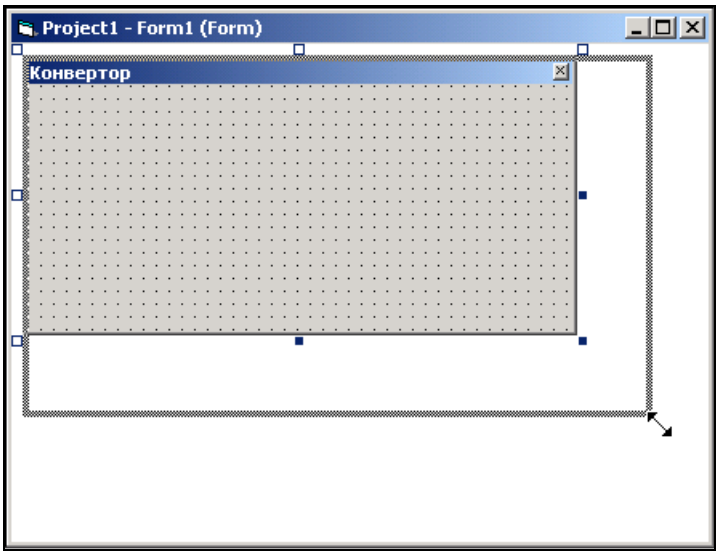


Рис. 2.3. Изменение размера формы путем перемещения границы

При выборе некоторых свойств (например, `BorderStyle`) справа от текущего значения свойства появляется значок раскрывающегося списка. Очевидно, что значение таких свойств можно задать путем выбора из списка (рис. 2.4). Здесь следует обратить внимание на то, что в списке сначала указывается числовое значение константы, а затем — ее название. При этом не следует путать название константы и ее обозначение. Например, чис-

ленное значение константы `Fixed Single` равно единице, а символьное значение равно `vbFixedSingle`.

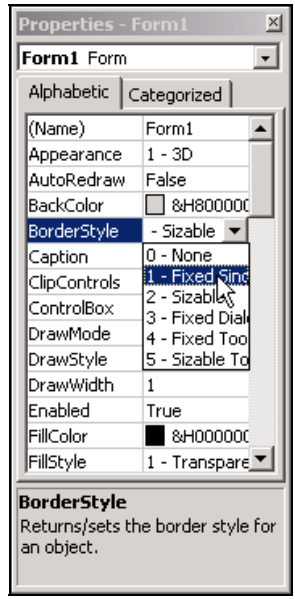


Рис. 2.4. Изменение значения свойства путем выбора из списка

Рядом со значениями некоторых свойств отображается командная кнопка с тремя точками. Это значит, что для изменения значения свойства используется дополнительное диалоговое окно. Например, в результате щелчка на кнопке с тремя точками в строке свойства `Icon` открывается окно **Load Icon**, в котором можно открыть один из каталогов компьютера и выбрать ico-файл (картинку, которая будет изображать системное меню в заголовке формы).

В табл. 2.2 приведены значения свойств стартовой формы разрабатываемой программы. Остальные свойства оставлены без изменения и в таблице не приведены.

Таблица 2.2. *Значения свойств стартовой формы*

Свойство	Значение
Caption	Конвертор
Width	4425
Height	3705
BorderStyle	Fixed Single
MaxButton	False
MinButton	False
StartUpPosition	CenterScreen
ScaleMode	Pixel
Font	Tahoma, обычный, 10

Компоненты

Программа пересчета цены из долларов в рубли должна получить от пользователя исходные данные — курс (соотношение рубля к доллару) и цену в долларах. В подобных программах данные с клавиатуры вводят в поля редактирования. Поэтому в форму разрабатываемого приложения надо добавить два поля редактирования — два компонента `TextBox`.

Для того чтобы добавить в форму компонент `TextBox` или другой *компонент*, необходимо:

1. В окне **ToolBox** выбрать этот компонент (щелкнуть на значке компонента) (рис. 2.5).
2. Установить указатель мыши в ту точку формы, в которой должен быть левый верхний угол компонента, и нажать левую кнопку мыши.
3. Переместить указатель мыши в ту точку, в которой должен быть правый нижний угол компонента, и отпустить кнопку мыши.

В результате этих действий на форме появится компонент.

Рис. 2.5. Значок компонента `TextBox` (поле редактирования)

Каждому добавленному в форму компоненту Visual Basic присваивает имя, которое состоит из стандартного имени компонента и порядкового номера. Например, если к форме добавить два компонента `TextBox`, то их имена будут `Text1` и `Text2`. Программист путем изменения значения свойства `Name` может изменить имя компонента. В простых программах имена компонентов, как правило, не изменяют.

На рис. 2.6 приведен вид формы после добавления двух компонентов `TextBox`, предназначенных для ввода исходных данных. Один из компонентов выделен. Свойства выделенного компонента отображаются в окне **Properties**. Чтобы увидеть свойства другого компонента, надо щелкнуть левой кнопкой мыши на его изображении или выбрать имя компонента в раскрывающемся списке, который находится в верхней части окна **Properties**.

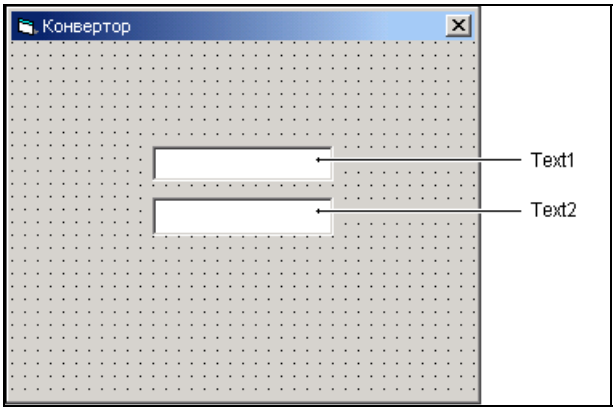


Рис. 2.6. Форма после добавления компонентов `TextBox`

В табл. 2.3 перечислены основные свойства компонента TextBox — поля редактирования.

Таблица 2.3. *Свойства компонента* `TextBox` *(поле ввода/редактирования)*

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам (в частности — для доступа к тексту, находящемуся в поле редактирования)
`Text`	Текст, находящийся в поле редактирования
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Width`	Ширина поля редактирования
`Height`	Высота поля редактирования
`Font`	Шрифт, используемый для отображения текста в поле редактирования
`ForeColor`	Цвет текста в поле компонента. Цвет можно задать, выбрав его из палитры или указав элемент цветовой схемы операционной системы
`Locked`	Используется для ограничения возможности изменения текста в поле редактирования. Если значение свойства равно `True`, то текст в поле редактирования изменить нельзя
`MultiLine`	Разрешает (при значении, равном `True`) отображение текста в несколько строк
`ScrollBars`	Управляет отображением полос прокрутки. У компонента может быть вертикальная полоса прокрутки (`Vertical`), горизонтальная (`Horizontal`) или обе полосы прокрутки (`Both`). Если значение свойства равно `None`, то полосы прокрутки не отображаются
`Visible`	Позволяет скрыть компонент (значение `False`) или сделать его видимым (значение `True`)

Дизайнер формы позволяет изменить размер и положение компонента при помощи мыши.

Чтобы изменить положение компонента, необходимо установить указатель мыши на его изображении, нажать левую кнопку мыши и, удерживая кнопку нажатой, переместить контур компонента в нужную точку формы, затем отпустить кнопку мыши. Во время перемещения компонента (рис. 2.7) отображаются текущие значения координат его левого верхнего угла (значения свойств `Left` и `Top`).

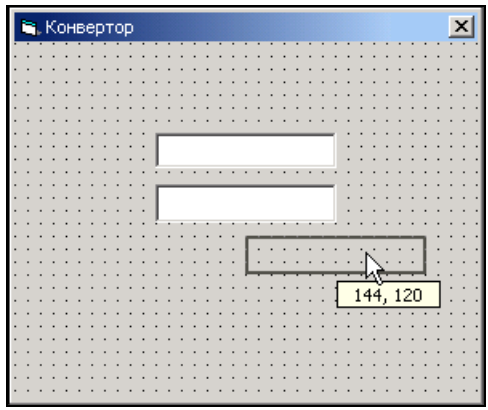


Рис. 2.7. Отображение текущих значений свойств `Left` и `Top` при изменении положения компонента

Чтобы изменить размер компонента, необходимо установить указатель мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить положение границы компонента, затем кнопку мыши надо отпустить. Во время изменения размера компонента отображаются текущие значения свойств `Height` и `Width` (рис. 2.8).

В табл. 2.4 приведены значения свойств полей редактирования `TextBox1` и `TextBox2`. Компонент `TextBox1` предназначен для ввода курса, `TextBox2` — для ввода цены в долларах. Обратите внимание на то, что значением свойства `Text` обоих компонен-

тов является пустая строка. Также следует обратить внимание и на то, что координаты и размеры компонентов указаны в пикселах.

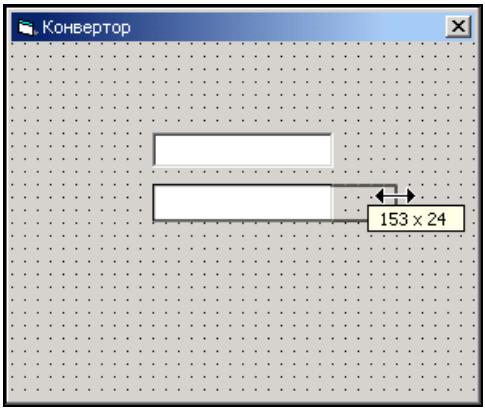


Рис. 2.8. Отображение текущих значений свойств `Width` и `Height` при изменении размера компонента

Таблица 2.4. *Значения свойств компонентов* `Text1` *и* `Text2`

Свойство	Значение	
	Text1	**Text2**
Text		
Left	88	88
Top	56	88
Width	113	113
Height	22	22

Помимо полей редактирования в окне программы должна находиться краткая информация о программе и назначении полей редактирования. Отображение текста на поверхности формы обеспечивает компонент `Label` (рис. 2.9). Добавляется компо-

нент Label в форму точно так же, как и компонент TextBox (поле редактирования).

Рис. 2.9. Значок компонента Label (поле отображения текста)

Свойства компонента Label перечислены в табл. 2.5.

Таблица 2.5. *Свойства компонента Label (поле отображения текста)*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
Caption	Текст, отображаемый в поле компонента
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы
Width	Ширина компонента (поля отображения текста)
Height	Высота компонента (поля отображения текста)
AutoSize	Признак того, что ширина компонента определяется его содержимым
WordWrap	Признак того, что слова, которые не помещаются в текущей строке (при условии, что значение свойства AutoSize равно False), должны автоматически переноситься на следующую строку
Alignment	Задает способ выравнивания текста внутри поля компонента. Текст может быть выровнен по левому краю (LeftJustify), по центру (Center) или по правому краю (RightJustify)

Таблица 2.5 *(окончание)*

Свойство	Описание
`Font`	Шрифт, используемый для отображения текста. Если значение свойства оставлено без изменения, то для отображения текста используется шрифт, заданный для формы
`ForeColor`	Цвет текста в поле компонента. Цвет можно задать, выбрав его из палитры или указав элемент цветовой схемы операционной системы
`BorderStyle`	Задает вид границы вокруг компонента. По умолчанию граница отсутствует
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

В форму разрабатываемого приложения надо добавить четыре компонента `Label`. Первый компонент `Label` предназначен для вывода информационного сообщения, второй и третий — для вывода информации о назначении полей редактирования (ввода), четвертый — для вывода результата расчета.

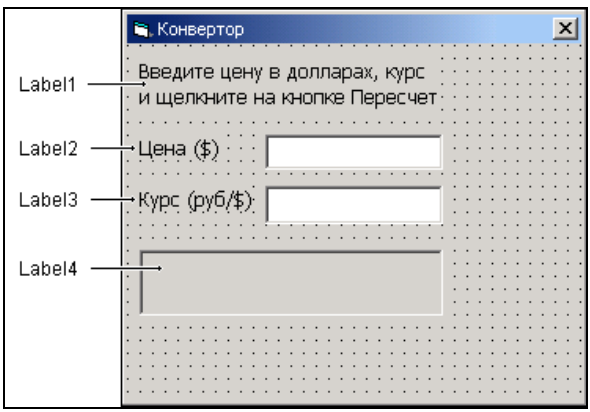


Рис. 2.10. Форма после добавления компонентов `Label` (полей отображения текста)

После того как компоненты Label будут добавлены в форму, надо выполнить их настройку — установить значения свойств в соответствии с табл. 2.6. Форма программы после настройки компонентов Label должна выглядеть так, как показано на рис. 2.10.

Обратите внимание: значение свойства Caption вводится как одна строка. Расположение текста в поле компонента зависит от характеристик используемого шрифта (свойство Font) и размеров компонента.

Таблица 2.6. *Значения свойств компонентов Label1, Label2, Label3 и Label4*

Компонент	Свойство	Значение
Label1	Caption	Введите цену в долларах, курс и щелкните на кнопке **Пересчет**
	Left	8
	Top	8
	Width	192
	Height	40
Label2	Caption	Цена ($)
	Left	8
	Top	56
	AutoSize	True
Label3	Caption	Курс (руб/$)
	Left	8
	Top	88
	AutoSize	True
Label4	Caption	
	Left	8
	Top	128
	Width	193
	Height	41
	BorderStyle	Fixed Single

Последнее, что надо сделать на этапе создания формы, — добавить две командные кнопки: **Вычислить** и **Завершить**. Назначение этих кнопок очевидно.

Командная кнопка — компонент `CommandButton` (рис. 2.11), добавляется в форму точно так же, как и другие компоненты. Свойства этого компонента приведены в табл. 2.7.

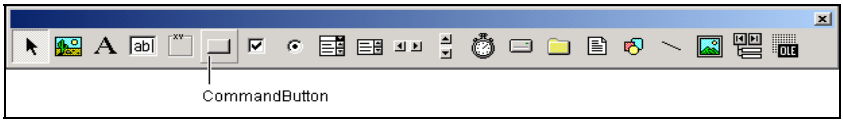


Рис. 2.11. Значок компонента `CommandButton` (командная кнопка)

Таблица 2.7. *Свойства компонента* `CommandButton` *(командная кнопка)*

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Width`	Ширина кнопки
`Height`	Высота кнопки
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна (пользователь может ее нажать, сделать щелчок на кнопке). Если значение свойства равно `False`, то кнопка не доступна

Таблица 2.7 *(окончание)*

Свойство	Описание
Visible	Позволяет скрыть кнопку (False) или сделать ее видимой (True).
Style	Тип кнопки. Копка может быть обычной (Standard) или "графической" (Graphical). На поверхности "графической" кнопки отображается картинка
Picture	Картинка, которая отображается на "графической" кнопке (значение свойства Style равно Graphical)
DisabledPicture	Для "графической" кнопки (значение свойства Style равно Graphical) задает картинку, которая отображается на кнопке, в том случае, если кнопка не доступна (значение свойства Enabled равно False)
DownPicture	Для "графической" кнопки (значение свойства Style равно Graphical) задает картинку, которая отображается на кнопке, в том случае, если кнопка нажата
ToolTipText	Задает текст подсказки, которая появляется при позиционировании указателя мыши на кнопке

После добавления к форме двух командных кнопок (компонентов CommandButton) нужно установить значения их свойств в соответствии с табл. 2.8.

Таблица 2.8. *Значения свойств компонентов* Button1 *и* Button2

Свойство	Значение	
	Button1	**Button2**
Caption	Пересчет	Завершить
Left	8	96
Top	184	184
Width	73	73
Height	25	25

Окончательный вид формы разрабатываемого приложения **Конвертор** приведен на рис. 2.12.

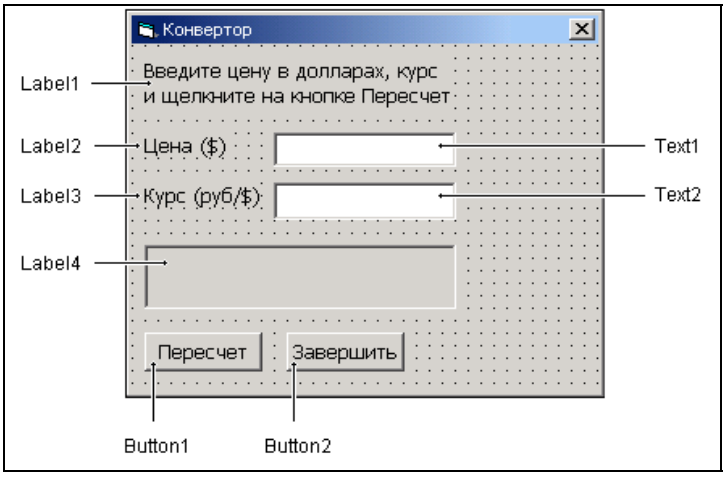


Рис. 2.12. Форма программы **Конвертор**

Завершив работу над формой приложения, можно приступить к программированию.

Событие и процедура обработки события

Вид созданной формы подсказывает, как работает приложение. Очевидно, что пользователь должен ввести в поля редактирования исходные данные и щелкнуть мышью на кнопке **Пересчет**. Щелчок на изображении командной кнопки — это пример того, что в Windows называется *событием*.

Событие (`Event`) — это то, что происходит во время работы программы. В VisualBasic каждому событию присвоено имя. Например, щелчок кнопкой мыши на изображении командной кнопки — это событие `Click`, нажатие клавиши в процессе ввода строки текста в поле компонента `TextBox` — событие `KeyPress`.

В табл. 2.9 приведены некоторые события Windows.

Таблица 2.9. *События*

Событие	Происходит
`Click`	При щелчке кнопкой мыши
`DblClick`	При двойном щелчке кнопкой мыши
`MouseDown`	При нажатии кнопки мыши
`MouseUp`	При отпускании кнопки мыши
`MouseMove`	При перемещении мыши
`KeyPress`	При нажатии клавиши клавиатуры
`KeyDown`	При нажатии клавиши клавиатуры. События `KeyDown` и `KeyPress` — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие `KeyUp`)
`KeyUp`	При отпускании нажатой клавиши клавиатуры
`Initialize`	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
`Activate`	Когда элемент управления (форма) становится активным окном
`Paint`	При появлении окна на экране в начале работы программы, после появления окна или его части, которая, например, была закрыта другим окном, а также и в других подобных случаях
`Resize`	При изменении размера элемента управления
`GotFocus`	При получении элементом управления фокуса (например, при перемещении курсора в поле ввода текста)
`LostFocus`	При потере элементом управления фокуса

Следует обратить внимание на то, что действия пользователя, как правило, приводят к возникновению последовательности

(цепочки) событий. Например, в начале работы программы возникает цепочка событий `Initialize - Load - Activate - Resize - Paint`, а в конце работы программы, когда пользователь сделает щелчок на кнопке **Закрыть —** цепочка `QueryUnload - Unload — Terminate`.

Реакцией на событие должно быть какое-либо действие. В Visual Basic реакция на событие реализуется как *процедура обработки события*. Таким образом, для того чтобы программа выполняла некоторую работу в ответ на действия пользователя, программист должен написать процедуру обработки соответствующего события. Следует обратить внимание на то, что значительную часть обработки событий берет на себя компонент. Поэтому программист должен разрабатывать процедуру обработки события только в том случае, если реакция на событие отличается от стандартной или не определена. Например, если по условию задачи ограничений на символы, вводимые в поле `TextBox`, нет, то процедуру обработки события `KeyPress` писать не надо, т. к. во время работы программы будет использована стандартная (скрытая от программиста) процедура обработки этого события.

Методику создания процедур обработки событий рассмотрим на примере процедуры обработки события `Click` для командной кнопки **Пересчет**. Эта процедура должна получить исходные данные из полей редактирования `Text1` (курс) и `Text2` (цена в долларах), затем выполнить расчет (пересчитать цену из долларов в рубли) и вывести результат в поле отображения текста (`Label4`).

Чтобы создать процедуру обработки события `Click` на командной кнопке, надо в окне дизайнера формы сделать двойной щелчок мышью на изображении этой кнопки, в результате чего станет доступным окно редактора кода, в которое будет добавлена процедура обработки события (рис. 2.13). Имя процедуры обработки события, сформированное средой разработки, состоит из двух частей. Первая часть имени идентифицирует объект, для которого создана процедура, вторая часть — событие. В нашем примере объект — это командная кнопка `Command1`, а событие — `Click`. После этого в окне редактора кода можно набирать инструкции, реализующие процедуру обработки события.

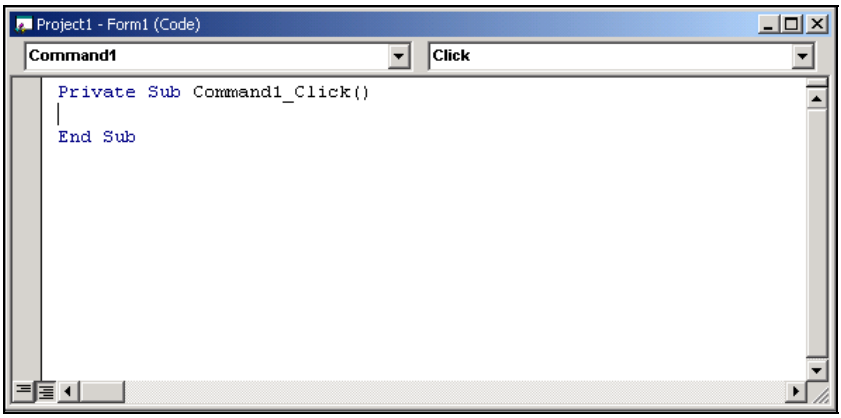


Рис. 2.13. Процедура обработки события

Чтобы создать процедуру обработки другого события, надо активизировать окно редактора кода (выбрать в меню **View** команду **Code**), выбрать объект (рис. 2.14), для которого надо создать процедуру обработки события, а затем из раскрывшегося списка событий, который находится в верхней части окна редактора кода (справа от списка объектов), выбрать событие (рис. 2.15).

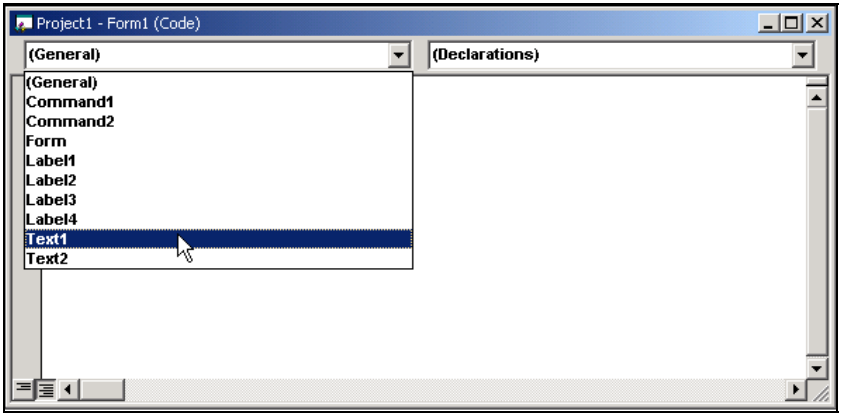


Рис. 2.14. Выбор объекта,
для которого надо создать процедуру обработки события

Например, чтобы создать процедуру обработки события KeyPress для поля редактирования Text1, надо сначала выбрать объект Text1, затем — событие KeyPress.

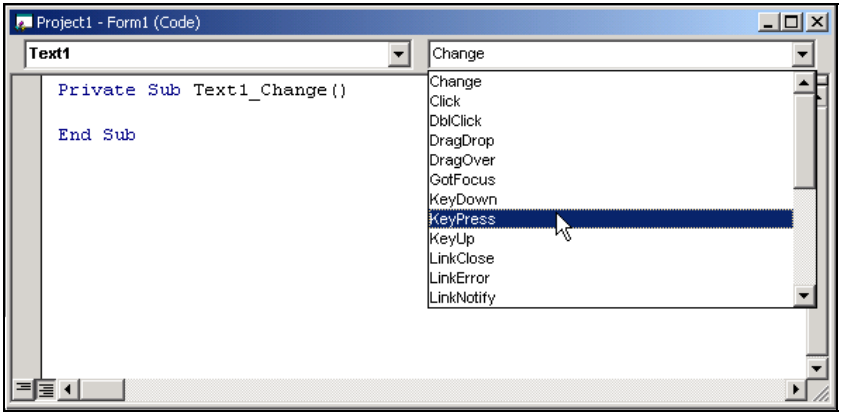


Рис. 2.15. Выбор события, обработка которого должна быть выполнена

В листинге 2.1 приведен текст процедуры обработки события Click для командной кнопки **Пересчет**. Обратите внимание на то, как представлена программа. Ее общий вид соответствует тому, как она выглядит в окне редактора кода: ключевые слова выделены полужирным, комментарии — курсивом (выделение выполняет редактор кода). Кроме того, инструкции программы набраны с отступами в соответствии с принятыми в среде программистов правилами хорошего стиля.

Листинг 2.1. Процедура обработки события Click на кнопке Пересчет

```
Private Sub Command1_Click()

    Dim k As Double     ' курс
    Dim usd As Double   ' цена в долларах
    Dim r As Double     ' цена в рублях
```

```
    ' ввод исходных данных
    usd = Val(Text1.Text)
    k = Val(Text2.Text)

    ' пересчет
    r = usd * k

    ' вывод результата
    Label4.Caption = Str(usd) + "$ = " + Str(r) + "руб."

End Sub
```

Процедура `Command1_Click` выполняет пересчет цены из долларов в рубли и выводит результат расчета в поле `Label4`. Исходные данные вводятся из полей редактирования `Text1` и `Text2` путем обращения к свойству `Text`. Доступ к свойству осуществляется путем указания имени объекта (`Text1` или `Text2`) и свойства (`Text`). Имя свойства от имени объекта отделяется точкой. Свойство `Text` (символьного типа) содержит строку символов, которую ввел пользователь. Для правильной работы программы строка должна представлять собой изображение целого или дробного числа, т. е. содержать только цифры и, возможно, разделитель целой и дробной частей числа — запятую. Преобразование строки символов в число выполняет функция `Val`, которой в качестве параметров передается значение свойства `Text` — строка символов, находящаяся в поле редактирования. Значение функции `Val` — это число, изображением которого является строка-параметр.

После того как исходные данные будут помещены в переменные *k* и *usd*, выполняется расчет.

Вычисленное значение цены в рублях выводится в поле `Label4` путем присваивания значения свойству `Caption`. Для преобразования числа в строку символов используется функция `Str`.

В результате нажатия кнопки **Завершить** программа должна завершить работу. Чтобы это произошло, надо закрыть главное окно программы. Делается это при помощи метода `Close`. Про-

цедура обработки события `Click` для кнопки **Завершить** приведена в листинге 2.2.

Листинг 2.2. Процедура обработки события `Click` на кнопке Завершить

```
Private Sub Command2_Click()
    End ' завершить работу программы
End Sub
```

Редактор кода

Редактор кода выделяет ключевые слова языка программирования (**Sub**, **Function**, **Dim**, **If**, **Else** и др.) полужирным шрифтом, что делает текст программы более выразительным и облегчает восприятие структуры программы.

Помимо ключевых слов редактор кода выделяет цветом комментарии.

В процессе разработки программы часто возникает необходимость переключения между окном редактора кода и окном дизайнера формы. Выбрать нужное окно можно при помощи командных кнопок **View Object** и **View Code**, которые находятся в верхней части окна менеджера проекта (рис. 2.16).

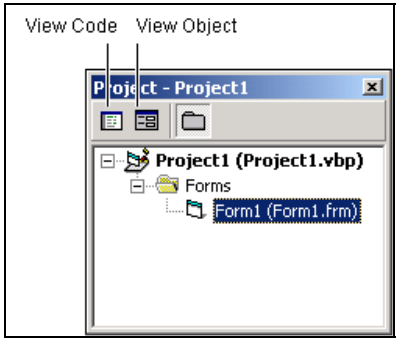


Рис. 2.16. Командные кнопки **View Object** и **View Code**

В процессе набора текста программы редактор кода выводит справочную информацию о параметрах процедур и функций, о свойствах и методах объектов.

Например, если в окне редактора кода набрать `MsgBox` (имя функции, которая выводит на экран окно сообщения) и открывающую скобку, то на экране появится окно подсказки, в котором будут перечислены параметры функции `MsgBox` с указанием их типа (рис. 2.17). Один из параметров выделен полужирным. Так редактор подсказывает программисту, какой параметр он должен вводить. После набора параметра и запятой в окне подсказки будет выделен следующий параметр. И так до тех пор, пока не будут указаны все параметры.

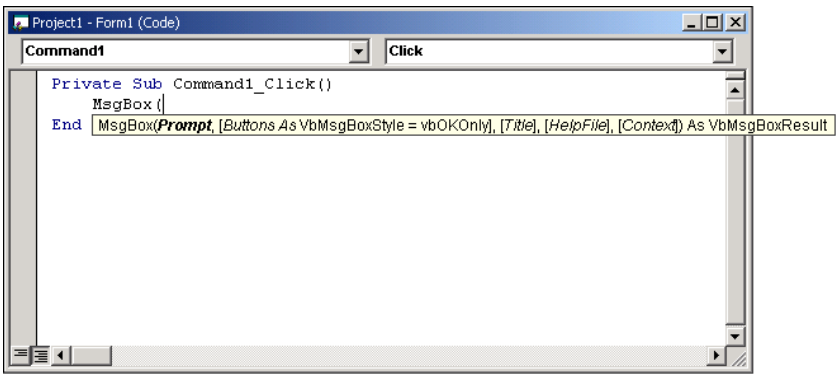


Рис. 2.17. Пример подсказки

Для объектов редактор кода выводит список свойств и методов. Как только программист наберет имя объекта (компонента) и точку, так сразу на экране появляется окно подсказки — список свойств и методов этого объекта (рис. 2.18). Перейти к нужному элементу списка можно при помощи клавиш перемещения курсора, либо набрав на клавиатуре несколько первых букв имени нужного свойства или метода. После того как будет выбран нужный элемент списка и нажата клавиша <Enter>, выбранное свойство или метод будут вставлены в текст программы.

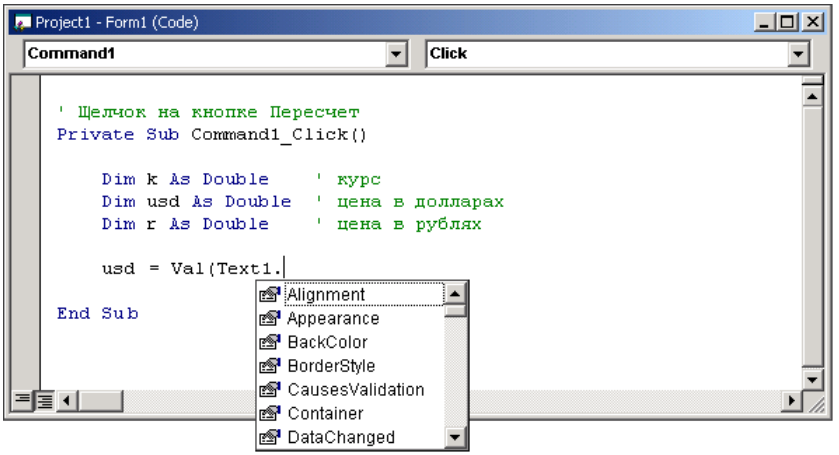


Рис. 2.18. Редактор кода автоматически выводит список свойств и методов объекта (компонента)

Система подсказок существенно облегчает процесс подготовки текста программы, избавляет от рутины. Кроме того, если во время набора программы подсказка не появилась, то это означает, что программист допустил ошибку (скорее всего, неверно набрал имя процедуры или функции).

В процессе набора текста программы Visual Basic проверяет программу на отсутствие *грубых синтаксических* ошибок. Контроль активизируется сразу после набора очередной строки текста программы. Если в набранной строке есть ошибка, то компилятор выделяет цветом эту строку и выводит сообщение. Окончательная проверка программы на отсутствие синтаксических ошибок выполняется во время ее работы.

Запись инструкций

Программа на языке Visual Basic представляет собой последовательность инструкций, каждую из которых, за исключением инструкций **If**, **Select**, **For**, **While** и некоторых других, обычно записывают в отдельной строке. Например:

```
Dim k As Double
Dim i As Integer
```

```
k = Val(Text1.Text)
usd = Val(Text2.Text)
rub = k * usd
```

Несколько инструкций можно записать в одной строке, отделив одну инструкцию от другой двоеточием:

```
Dim k As Double : Dim usd As Double
k = Val(Text1.Text) : usd = Val(Text2.Text) : rub = k * usd
```

Длинные инструкции (например, инструкции вызова функций или процедур, у которых много параметров) лучше записывать в несколько строк. Чтобы инструкцию записать в несколько строк, необходимо в конец текущей строки поместить символ подчеркивания (_), отделив его от последнего набранного символа инструкции пробелом, а затем нажать клавишу <Enter>, чтобы продолжить набор текста в следующей строке. Например:

```
MsgBox("Надо задать цену и курс.", _
       "Конвертор", vbInformation + vbOk)
```

Инструкции **If**, **Select**, **For** и некоторые другие необходимо записывать в несколько строк, причем именно так этого требует синтаксис Visual Basic. Например, слово **Then** инструкции **If** должно следовать сразу за условием, а не в следующей строке (если в процессе набора программы нажать клавишу <Enter> сразу после того, как будет набрано условие, Visual Basic выведет сообщение об ошибке).

В процессе набора текста программы рекомендуется следовать правилам хорошего стиля программирования, что предполагает использование отступов при записи инструкций выбора и циклов, комментариев, а также пустых строк. Формально отступы, пустые строки и комментарии в тексте программы не нужны, но их наличие существенно облегчает восприятие программы, делает более понятной ее структуру. Ниже приведен пример записи инструкции **Select**.

```
Select Case KeyAscii
    Case 8, 48 To 57 ' <Backspace> и цифры

    Case 44 ' запятая
        If InStr(1, Text1.Text, ",") <> 0 Then
```

```
            KeyAscii = 0
        End If

    Case 46 ' точка
        If InStr(1, Text1.Text, ",") <> 0 Then
            KeyAscii = 0
          Else
            KeyAscii = 44
        End If

    Case 13 ' клавиша <Enter>
        Text2.SetFocus

    Case Else
        KeyAscii = 0
End Select
```

Следует обратить внимание на то, что слова **Case** записаны с отступом относительно слова **Select**. Инструкции, которые должны быть выполнены в случае совпадения значения переменной-селектора KeyAscii со значением одной из констант, указанной после слова **Case**, также смещены, но уже относительно слова **Case**. Также обратите внимание, что слово **End**, отмечающее конец инструкции **If**, находится строго под словом **If**, а инструкции, которые должны быть выполнены (в случае если условие истино или ложно), записаны одна под другой и смещены относительно **If**.

Приведенную выше инструкцию **Select** можно записать и так:

```
Select Case KeyAscii
Case 8, 48 To 57 ' <Backspace> и цифры
Case 44 ' запятая
If InStr(1, Text1.Text, ",") <> 0 Then
KeyAscii = 0
End If
Case 46 ' точка
If InStr(1, Text1.Text, ",") <> 0 Then
KeyAscii = 0
```

```
Else
KeyAscii = 44
End If
Case 13 ' клавиша <Enter>
Text2.SetFocus
Case Else
KeyAscii = 0
End Select
```

Для облегчения понимания логики работы программы в текст программы нужно включать поясняющий текст — комментарии. В Visual Basic комментарий — это текст, который следует за одинарной кавычкой до конца строки. Комментировать рекомендуется назначение основных переменных, действия, выполняемые процедурами и функциями, ключевые операции и точки программы.

Справочная информация

В процессе набора программы можно получить справку (например, о конструкции языка программирования, процедуре или функции). Для этого надо в окне редактора кода набрать слово (инструкцию языка программирования, имя процедуры или функции и т. д.), о котором надо получить справку, после чего нажать клавишу <F1>.

Получить доступ к справочной информации можно, выбрав в меню **Help** команду **Contents** или **Index**. Команда **Contents** обеспечивает доступ к вкладке **Содержание**, а команда **Index** — к вкладке **Указатель** окна справочной системы. На вкладке **Содержание** надо выбрать раздел **Visual Basic Documentation**, а в этом разделе — нужный подраздел. Вкладка **Указатель** обеспечивает доступ к нужному разделу справочной информации по ключевому слову. Как правило, в качестве ключевого слова используют несколько первых букв имени функции, процедуры, свойства или метода (рис. 2.19).

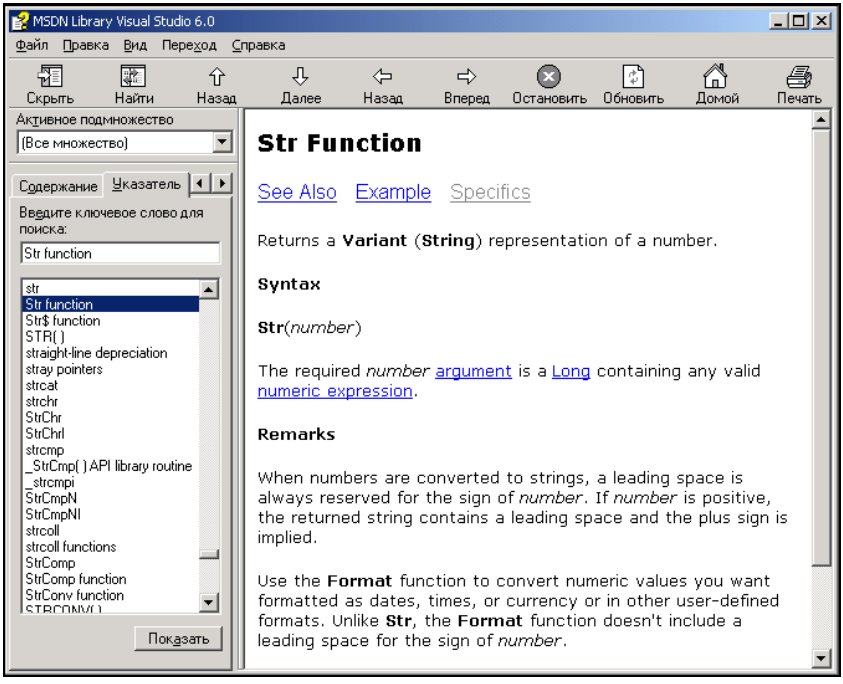


Рис. 2.19. Поиск справочной информации по ключевому слову

Сохранение проекта

Проект Visual Basic — это совокупность файлов (форм, модулей и других компонентов), которые и образуют приложение (программу) и которые компилятор использует для создания исполняемого (exe) файла. В простейшем случае проект образуют файл описания проекта (vbp) и файл формы (frm).

Перед тем как сохранить проект, рекомендуется задать ему имя (это имя отображается в заголовке главного окна, окна дизайнера формы, проекта и др.). Чтобы задать имя проекта, надо в окне **Project** сделать щелчок на названии проекта и в окне **Properties** изменить значение свойства **Name** (рис. 2.20).

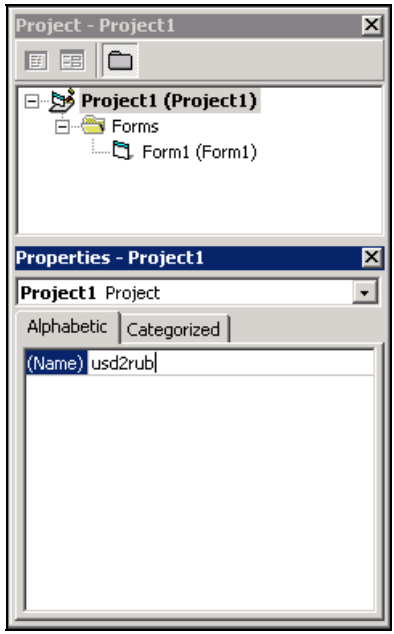


Рис. 2.20. Перед тем как сохранить проект, надо задать его имя

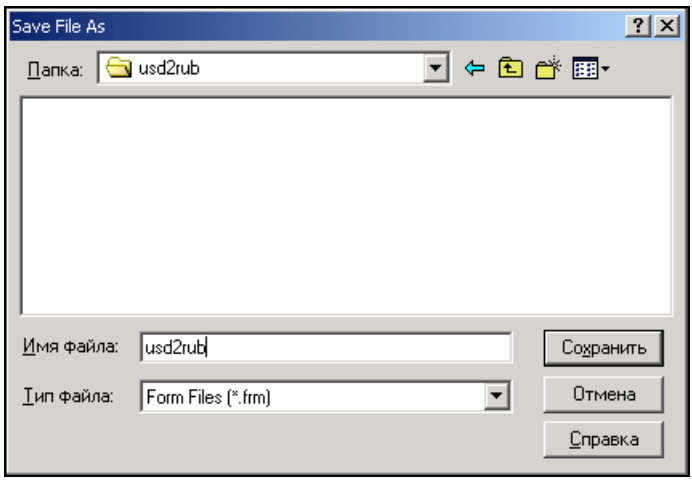


Рис. 2.21. Сохранение модуля формы

Для того чтобы сохранить проект (программу, над которой в данный момент работает программист), надо в меню **File** выбрать команду **Save Project** или сделать щелчок на соответствующей командной кнопке. Если проект еще ни разу не был сохранен, то на экране появляется окно **Save File As**, в котором надо выбрать папку, предназначенную для проектов Visual Basic, создать в папке проектов новую папку для сохраняемого проекта, открыть эту папку и задать имя файла модуля формы (рис. 2.21). Затем в следующем окне **Save Project As** надо задать имя файла проекта (рис. 2.22).

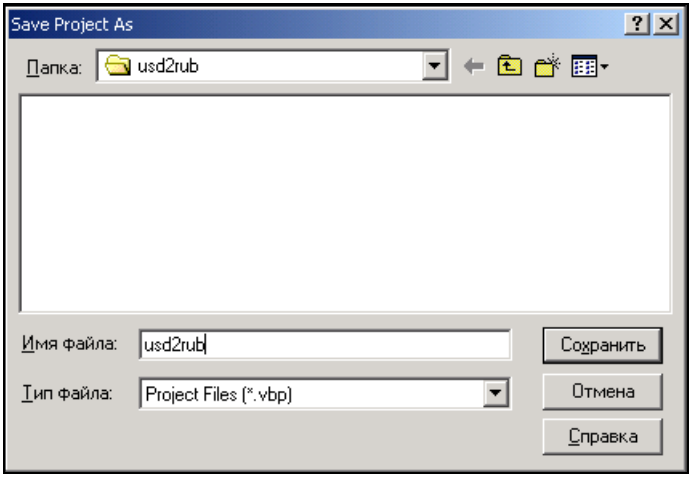


Рис. 2.22. Сохранение файла проекта

Запуск программы

После того как программа будет сохранена, можно выполнить ее запуск. Запустить программу можно несколькими способами:

- в меню **Run** выбрать команду **Start**;
- нажать клавишу <F5>;
- сделать щелчок на командной кнопке **Start**, которая находится на стандартной панели инструментов (рис. 2.23). На этой

же панели инструментов находится кнопка **End**, щелчок на которой останавливает запущенную программу.

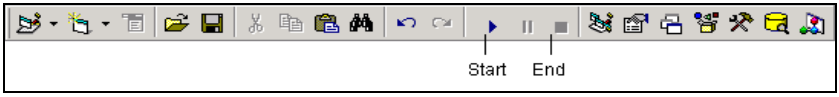


Рис. 2.23. Кнопки управления процессом выполнения программы

Если в процессе компиляции или выполнения программы, запущенной из среды разработки, обнаруживается ошибка, то выполнение программы приостанавливается и на экране появляется окно с сообщением об ошибке (рис. 2.24). Чтобы исправить обнаруженную ошибку, надо завершить выполнение программы: сначала закрыть окно сообщения об ошибке, затем щелкнуть на кнопке **End**.

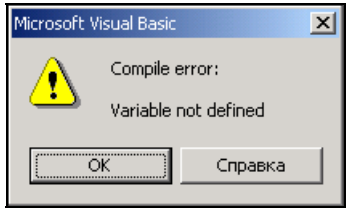


Рис. 2.24. Пример сообщения об ошибке

Следует обратить внимание на то, что окончательная проверка программы происходит во время ее выполнения и, следовательно, возможна ситуация, когда в какой-либо части программы, которая еще ни разу не была активизирована, содержатся ошибки, хотя сама программа вроде бы и работает. Чтобы избежать подобных неприятностей, надо перед запуском программы выполнить ее компиляцию: выбрать в меню **Run** команду **Start With Full Compile**. При выборе этой команды программа будет запущена только в том случае, если в ней нет синтаксических ошибок.

Также следует обратить внимание и на то, что по умолчанию в Visual Basic переменные можно не объявлять. Эта особенность языка является источником ошибок, которые иногда довольно трудно обнаружить. Поэтому настоятельно рекомендуется в начало текста программы поместить директиву `Option Explicit`, которая устанавливает, что все переменные должны быть объявлены явно в инструкции `Dim`, или на вкладке **Editor** окна **Options** (рис. 2.25) надо установить переключатель **Require Variable Declaration** (Требовать объявления переменных). Окно **Options** становится доступным в результате выбора команды **Tools ▸ Options**.

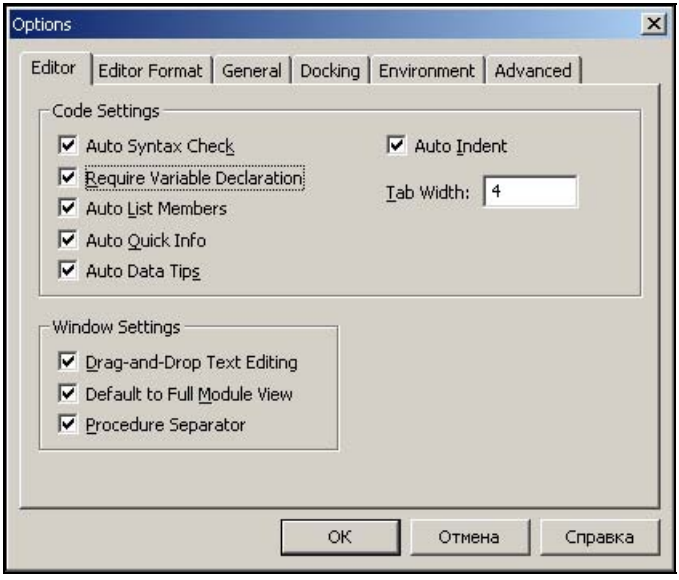


Рис. 2.25. Выберите переключатель **Require Variable Declaration**

Создание exe-файла

Чтобы иметь возможность запустить программу из операционной системы, а не только из среды разработки Visual Basic, надо создать выполняемый файл.

Чтобы создать выполняемый exe-файл, надо в меню **File** выбрать команду **Make**. На экране появится окно **Make Project** (рис. 2.26), в котором можно задать/изменить имя выполняемого файла программы (по умолчанию имя создаваемого компилятором файла совпадает с именем проекта).

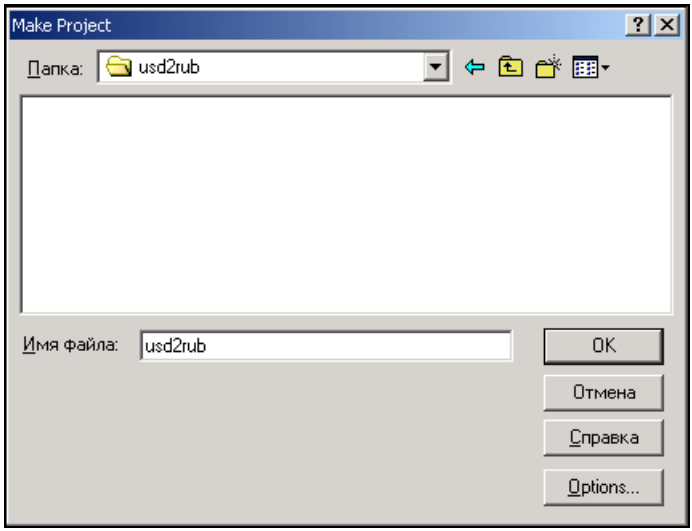


Рис. 2.26. Щелчок на кнопке **OK** активизирует процесс создания exe-файла

Завершение работы

Для того чтобы завершить работу с Visual Basic, надо в меню **File** выбрать команду **Exit**. Если с момента последнего сохранения проекта в программу были внесены какие-либо изменения, то на экране появится окно с запросом о необходимости сохранить внесенные изменения (рис. 2.27).

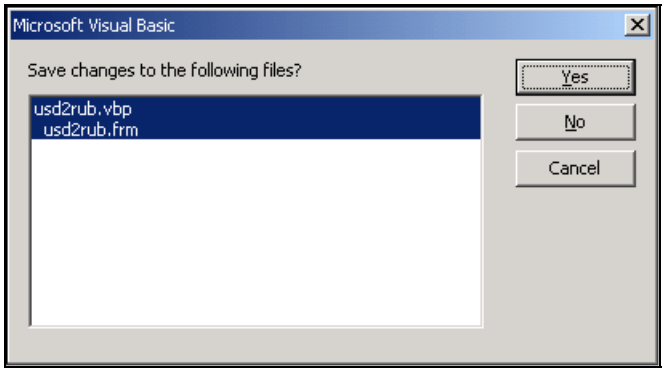


Рис. 2.27. Запрос о необходимости сохранения изменения в проекте

Внесение изменений

После нескольких запусков программы **Конвертор** возникает желание усовершенствовать ее. Например, так, чтобы в поля **Цена** и **Курс** пользователь мог вводить только цифры, и чтобы после ввода цены (в результате нажатия клавиши <Enter>) курсор переходил в поле **Курс**, а нажатие клавиши <Enter> в поле **Курс** активизировало бы процесс расчета.

Чтобы внести какие-либо изменения в программу, нужно запустить Visual Basic и открыть соответствующий проект. Для этого надо из меню **File** выбрать команду **Open Project**. В результате становится доступным окно **Open Project**. Вкладка **Existing** этого окна позволяет открыть проект обычным образом. На вкладке **Resent** перечислены проекты, над которыми программист работал в последнее время. Чтобы внести изменения в программу **Конвертор**, надо на вкладке **Resent** выбрать проект **usd2rub** (рис. 2.28).

В листинге 2.3 приведена программа **Конвертор**, в которую добавлены процедуры обработки события `KeyPress` для компонентов `Text1` и `Text2`. Вспомните: для того чтобы добавить в программу процедуру обработки события, нужно в окне редактора кода выбрать компонент, для которого создается процедура, а

также и событие (Visual Basic сформирует шаблон процедуры обработки события, после чего можно вводить инструкции, реализующие процедуру обработки).

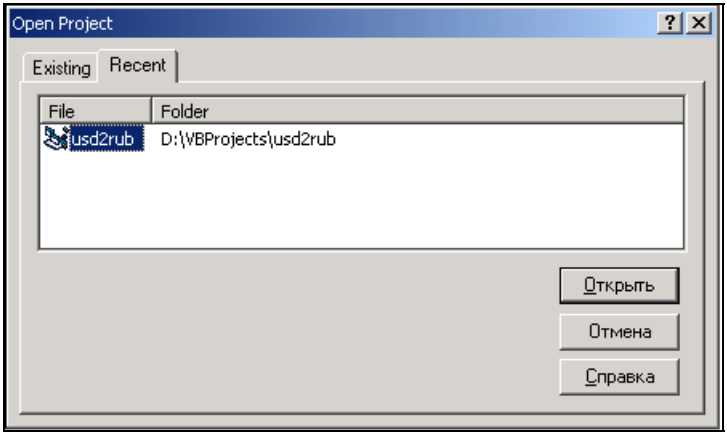


Рис. 2.28. На вкладке **Resent** перечислены проекты, над которыми программист работал в последнее время

Листинг 2.3. Модуль формы программы Конвертор после внесения изменений

```
Option Explicit

' Щелчок на кнопке Пересчет
Private Sub Command1_Click()

    Dim k As Double     ' курс
    Dim usd As Double   ' цена в долларах
    Dim r As Double     ' цена в рублях

    ' ввод исходных данных
    k = Val(Text1.Text)
    usd = Val(Text2.Text)
```

```
    If k = 0 Or usd = 0 Then
        MsgBox "Надо задать цену и курс", , "Конвертор"
    Else
        ' пересчет
        r = usd * k
        ' вывод результата
        Label4.Caption = Str(usd) + "$ = " + Str(r) + "руб."
    End If
End Sub

' Щелчок на кнопке Завершить
Private Sub Command2_Click()
    End
End Sub

' нажатие клавиши в поле Цена
'(обработка события KeyPress в поле компонента Text1)
Private Sub Text1_KeyPress(KeyAscii As Integer)

    ' Параметр KeyAscii содержит код нажатой клавиши.
    ' Если параметру присвоить значение 0, то символ,
    ' соответствующий нажатой клавише в поле редактирования,
    ' не появится и у пользователя будет впечатление,
    ' что программа не реагирует на нажатие некоторых клавиш.
    ' В данном случае допустимы символы:
    ' цифры (код клавиш от 48 до 57)
    ' запятая (код 44),
    ' а также клавиша Backspace.
    ' Точку процедура меняет на запятую.

    Select Case KeyAscii
        Case 8, 48 To 57
            ' <Backspace> и цифры
```

```
        Case 44 ' запятая
            If InStr(1, Text1.Text, ",") <> 0 Then
                KeyAscii = 0
            End If

        Case 46 ' точка
            If InStr(1, Text1.Text, ",") <> 0 Then
                KeyAscii = 0
            Else
                KeyAscii = 44
            End If

        Case 13 ' клавиша <Enter>
            Text2.SetFocus

        Case Else
            KeyAscii = 0
    End Select
End Sub

' нажатие клавиши в поле Курс
'(обработка события KeyPress в поле компонента Text2)
Private Sub Text2_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 8, 48 To 57
            ' <Backspace> и цифры

        Case 44 ' запятая
            If InStr(1, Text1.Text, ",") <> 0 Then
                KeyAscii = 0
            End If

        Case 46 ' точка
            If InStr(1, Text1.Text, ",") <> 0 Then
```

```
            KeyAscii = 0
        Else
            KeyAscii = 44
        End If

    Case 13 ' клавиша <Enter>
        Command1.SetFocus

    Case Else
        ' не отображать символ
        KeyAscii = 0

End Select
End Sub
```

Процедура обработки события `KeyPress` для компонента `Text1` получает в качестве параметра (`KeyAscii`) код клавиши, нажатие которой вызвало событие. Если в процедуре параметру `KeyAscii` присвоить значение 0, то символ, соответствующий нажатой клавише, в поле редактирования не появится, и у пользователя будет впечатление, что программа не реагирует на нажатие клавиши. Таким образом можно обеспечить фильтрацию символов. В рассматриваемой программе поле `Text1` должно содержать дробное число, т. е. допустимыми символами являются цифры и запятая.

Если нажата допустимая клавиша, то процедура ничего не делает и соответствующий символ появляется в поле редактирования.

Если нажата недопустимая клавиша, то параметру `KeyAscii` присваивается значение `0`.

В случае нажатия клавиши <Enter> процедура путем вызова метода `SetFocus` компонента `Text2` переводит курсор в поле редактирования `Text2`.

Если пользователь нажимает запятую, то процедура проверяет, есть ли в поле редактирования запятая (это делает функция `InStr`, которая возвращает номер позиции искомого символа в строке

или ноль, если символа в строке нет). Если в поле редактирования запятая уже есть, то параметру `KeyAscii` присваивается значение `0`, и вторая запятая не появляется. Если нажата точка, то программа проверяет, есть ли в поле редактирования запятая, и если нет, то заменяет код точки на код запятой. Таким образом, пользователь нажимает точку, а появляется запятая.

Процедура обработки события `KeyPress` для компонента `Text2` работает аналогичным образом.

После внесения изменений проект следует откомпилировать (команда **Run ▸ Start With Full Compile**), создать выполняемый файл (команда **File ▸ Make**) и сохранить (команда **File ▸ Save Project**).

Значок приложения

Свойство `Icon` определяет значок, который отображается в заголовке окна программы. Этот же значок отображается на панели задач во время работы программы, а также изображает программу (exe-файл) в папке. По умолчанию это стандартный значок (рис. 2.29).

Рис. 2.29. Стандартный значок Visual Basic

Программист может изменить значок формы. Для этого надо в строке свойства `Icon` щелкнуть на кнопке с тремя точками и в появившемся окне **Load Icon** (рис. 2.30) выбрать файл значка (ico-файл). Следует обратить внимание на то, что в процессе установки Visual Basic в каталог Microsoft Visual Studio\Common\Graphics\Icons копируется библиотека значков.

Также программист может создать свой собственный уникальный, значок. Сделать это можно при помощи утилиты Image Editor (Imageedit.exe), которая входит в состав Visual Basic и находится в папке Microsoft Visual Studio\Common\Tools\Vb\Imagedit.

Перед тем как рассмотреть процесс создания файла значка, необходимо сказать несколько слов о структуре ico-файла.

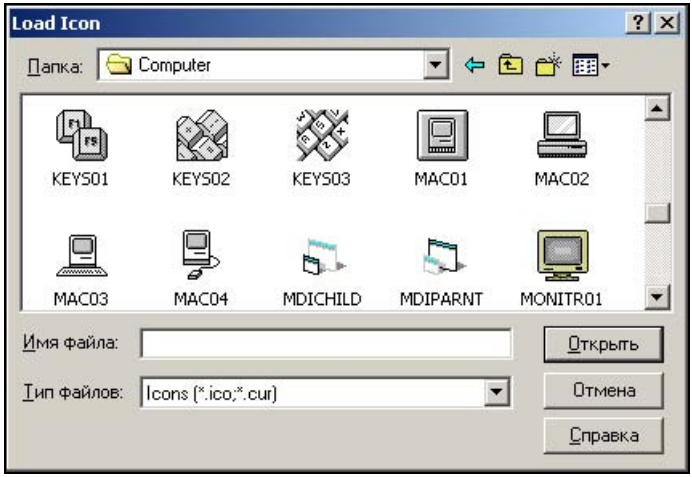


Рис. 2.30. Выбор значка для формы

В простейшем случае в ico-файле может находиться один-единственный значок. Размер значка может быть 16×16 или 32×32. Значок 16×16 отображается в заголовке диалогового окна без искажения. В папке в режиме отображения **Крупные значки** значок 16×16 масштабируется до размера 32×32 и поэтому выглядит "размытым". Значок 32×32, наоборот, в папке выглядит хорошо, а в заголовке окна нечетко, т. к. масштабируется до размера 16×16. Таким образом, чтобы значок отображался четко и в заголовке окна и в папке, ico-файл должен содержать две картинки: 16×16 и 32×32.

Значки могут отличаться и глубиной цвета (количеством цветов палитры). До недавнего времени стандартной считалась 16-цветная палитра. Сейчас палитра большинства значков насчитывает 256 цветов. К сожалению, Image Editor использует 16-цветную палитру.

Процесс создания ico-файла рассмотрим на примере: создадим файл, который будет содержать два 16-цветных значка (16×16 и 32×32).

Чтобы приступить к созданию значка, надо запустить **Image Editor**, в меню **File** выбрать команду **New**, в окне **Resource Type** выбрать **Icon** и щелкнуть на кнопке **OK** (рис. 2.31). Затем в поя-

вившемся окне **New Icon Image** (рис. 2.32) надо выбрать **EGA/VGA 16-Color 32x32**. В результате этих действий в ico-файл, над которым идет работа, будет добавлен новый *ресурс* (значок) и станет доступным окно редактирования картинки (рис. 2.33).

Рис. 2.31. Начало работы над новым значком

Рис. 2.32. Выбор характеристик значка в диалоговом окне **New Icon Image**

Процесс рисования в Image Editor практически ничем не отличается от соответствующего процесса в обычном графическом редакторе. Вместе с тем следует обратить внимание, что помимо обычных цветов в палитре есть цвет экрана (**Screen**). Точки картинки, закрашенные этим цветом, при отображении значка окрашиваются в цвет фона поверхности, на которой значок отображается.

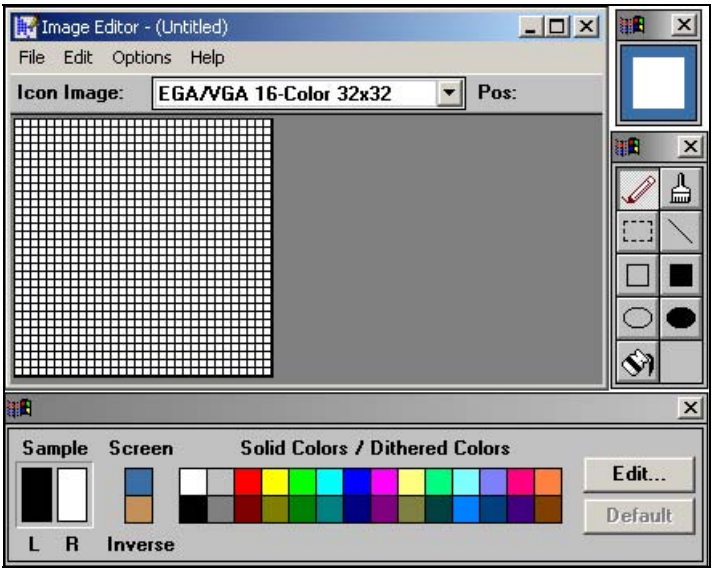


Рис. 2.33. Окно редактирования картинки значка

После того как картинка 32 ×32 будет готова, можно приступить к работе над картинкой размера 16×16. Для этого надо сначала в меню **Edit** выбрать команду **New Image**, затем в появившемся окне **New Icon Image** выбрать позицию **Small Icon 16-Color 16x16**.

Чтобы сохранить созданные картинки, надо в меню **File** выбрать команду **Save** и задать имя файла. Следует обратить внимание на то, что обе картинки будут сохранены в одном файле.

Окончательная настройка приложения

После того как программа будет отлажена, можно выполнить ее окончательную настройку: задать название программы и значок, который будет изображать программу (исполняемый файл приложения) в папке или на рабочем столе. Также в выполняемый файл можно поместить сведения о разработчике программы, о

ее версии, а кроме того, дать краткое описание. Эта информация отображается на вкладках окна **Свойства** (рис. 2.34).

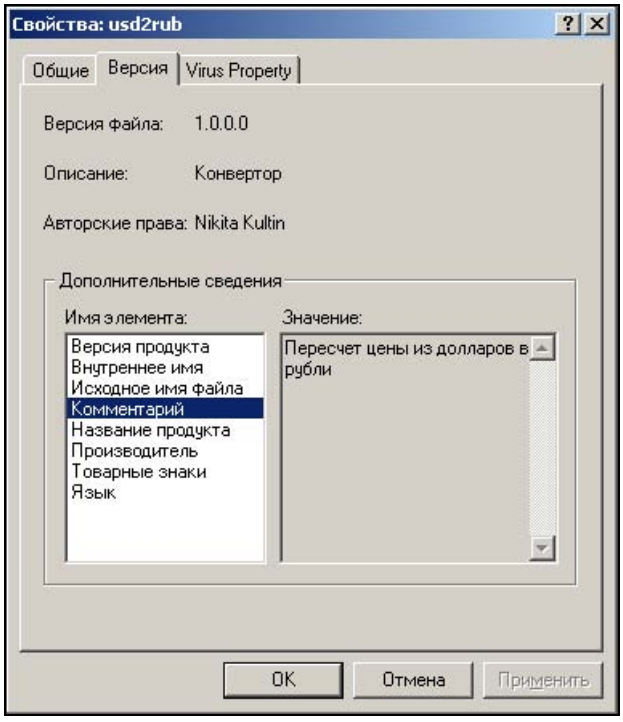


Рис. 2.34. Подробная информация о программе отображается во вкладках окна **Свойства**

Чтобы выполнить настройку приложения, надо в меню **Project** выбрать команду **Properties**. В поле **Application Title** вкладки **Make** надо ввести название приложения (рис. 2.35). На этой же вкладке можно задать значок, который будет изображать файл программы в папках компьютера (для этого надо раскрыть список **Icon** и выбрать форму, значок которой будет использоваться в качестве значка приложения). Чтобы задать информацию о программе (краткое описание, имя разработчика и т. д.) надо в группе **Version Information**, в списке **Type**, выбрать характеристи-

ку (свойство) программы и в поле **Value** ввести текст. Свойства программы приведены в табл. 2.10.

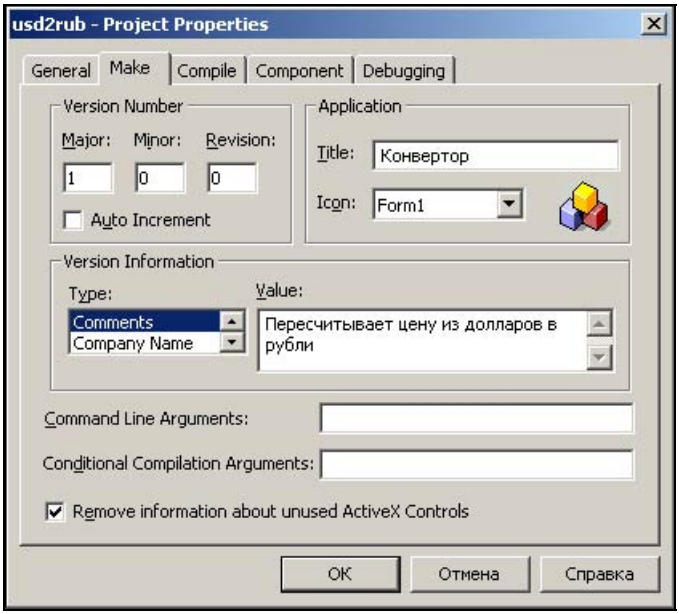


Рис. 2.35. Окончательная настройка приложения

Таблица 2.10. *Свойства (характеристики) программы*

Свойство	Содержание
`Comments`	Краткая информация о программе (назначение программы)
`Company Name`	Информация о разработчике
`File Description`	Описание файла (для программы Приложение)
`Legal Copyright`	Информация об авторских правах на программу
`Product Name`	Название программы

После того как будет выполнена настройка приложения, надо активизировать процесс создания выполняемого файла — выбрать команду **File ▸ Make**.

Перенос приложения на другой компьютер

Программа, созданная в Visual Basic, использует *динамические библиотеки*, которые устанавливаются на компьютер в процессе инсталяции Visual Basic. Поэтому чтобы программа, созданная в Visual Basic, могла работать на другом компьютере (у пользователя), там должны быть установлены соответствующие динамические библиотеки (если в процессе запуска программы обнаружится, что какая-либо из необходимых библиотек недоступна, то программа работать не сможет). Однако это не значит, что для успешного запуска программы на компьютере пользователя обязательно должен быть установлен Visual Basic — достаточно только библиотек.

В принципе, программу (exe-файл, файлы данных и справки), а также необходимые для ее работы динамические библиотеки можно вручную поместить на установочный диск и в таком виде передать пользователю, чтобы он сам скопировал файлы на диск своего компьютера и установил (зарегистрировал в системе) библиотеки. Но лучше все-таки создать программу установки, которая в автоматическом режиме выполнит необходимые действия. Программу установки можно поместить, например, на CD, и в таком виде передать пользователю.

Создать программу, обеспечивающую установку на компьютер пользователя приложения, созданного в Visual Basic, можно, например, при помощи утилиты Iexpress. Процесс создания программы установки описан в *главе 9*.

Глава 3

Базовые компоненты

В этой главе демонстрируются назначение и использование базовых компонентов. Основное внимание уделяется тому компоненту, название которого вынесено в заголовок раздела (вспомогательные компоненты при этом подробно не рассматриваются). В описании уделено внимание тем свойствам и методам, которые отражают специфику рассматриваемого компонента и представляют для программиста наибольший интерес (описание других свойств можно найти в справочной системе).

К базовым можно отнести компоненты: `Label`, `TextBox`, `CommandButton`, `CheckBox`, `OptionButton`, `ComboBox`, `ListBox`, `PictoureBox`, т. е. те, которые используются практически в любом приложении, они доступны (подключаются) автоматически, и, что немаловажно, программе, которая использует только эти компоненты, не нужны никакие дополнительные библиотеки (кроме стандартных).

Компонент *Label*

Компонент `Label` (рис. 3.1) предназначен для отображения текстовой информации.

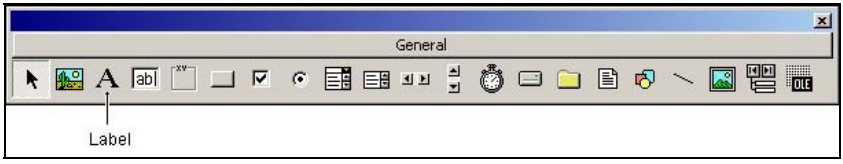


Рис. 3.1. Компонент `Label`

Задать текст, отображаемый в поле компонента, можно как во время разработки формы (задав значение свойства `Caption` в окне свойств), так и во время работы программы, присвоив значение свойству `Caption`.

Свойства компонента `Label` приведены в табл. 3.1.

Таблица 3.1. *Свойства компонента* `Label` *(поле отображения текста)*

Свойство	Описание
`Caption`	Отображаемый текст
`Font`	Шрифт, используемый для отображения текста
`AutoSize`	Признак того, что размер поля определяется его содержимым (значением свойства `Caption`)
`WordWrap`	Признак того, что слова, которые не помещаются в текущей строке, должны быть перенесены на следующую строку
`Alignment`	Задает способ выравнивания текста внутри поля. Текст может быть выровнен по левому краю (0), по центру (2) или по правому краю (1)
`BorderStyle`	Стиль границы поля компонента. Граница может быть (1) или отсутствовать (0). Вид границы определяет значение свойства `Appearance`
`Appearance`	Вид границы компонента (если значение свойства `BorderStyle` равно 1). Граница может представлять собой тонкую (0) или "объемную" (1) линию
`BackColor`	Цвет фона поля компонента. Цвет можно задать выбором из палитры или указав привязку к текущей цветовой схеме операционной системы
`BackStyle`	Управляет отображением фона области вывода текста. Область вывода текста может быть закрашена цветом, заданным свойством `BackColor` (в этом случае значение свойства должно быть равно 1), или быть прозрачной (0)

Таблица 3.1 *(окончание)*

Свойство	Описание
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

Если длина текста такова, что он не может быть выведен в одну строку (при заданном значении свойства `Width`), то для того чтобы текст был выведен в несколько строк, свойствам `AutoSize` и `WordWrap` надо присвоить соответственно значения `False` и `True`.

Чтобы в поле компонента `Label` вывести числовое значение, это значение надо при помощи функции `Str` или `Format` преобразовать в строку.

Программа Компонент Label (ее форма приведена на рис. 3.2, а текст — в листинге 3.1) демонстрирует возможности компонента `Label`. Следует обратить внимание на то, что компоненты `OptionButton` объединены в два массива.

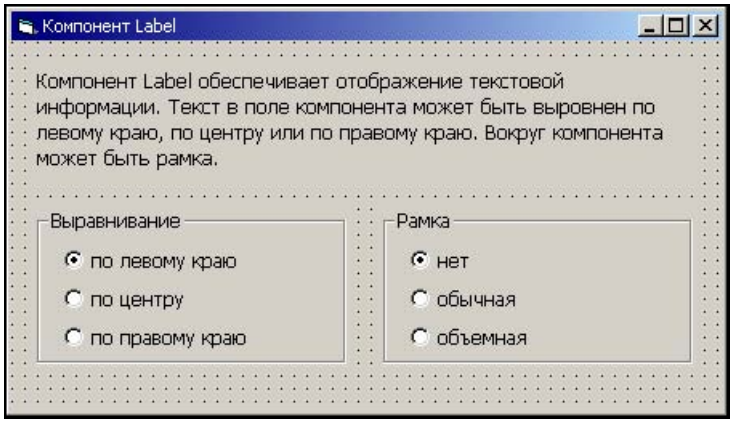


Рис. 3.2. Форма программы Компонент Label

Листинг 3.1. Изменение свойств компонента `Label`

```
' щелчок на переключателе группы "Выравнивание"
Private Sub Option1_Click(Index As Integer)
    Select Case Index
        Case 0 ' по левому краю
            Label1.Alignment = vbLeftJustify
        Case 1 ' по центру
            Label1.Alignment = vbCenter
        Case 2 ' по правому краю
            Label1.Alignment = vbRightJustify
     End Select


End Sub

' щелчок на переключателе группы "Рамка"
Private Sub Option2_Click(Index As Integer)
    Select Case Index
        Case 0 ' рамки
            Label1.BorderStyle = vbBSNone
            Label1.Appearance = 1
        Case 1 ' обычная рамка
            Label1.BorderStyle = vbFixedSingle
            Label1.Appearance = 0
        Case 2 ' объемная рамка
            Label1.BorderStyle = vbFixedSingle
            Label1.Appearance = 1
    End Select
End Sub
```

Компонент *TextBox*

Компонент `TextBox` (рис. 3.3) предназначен для ввода данных (строки символов) с клавиатуры. Свойства компонента приведены в табл. 3.2.

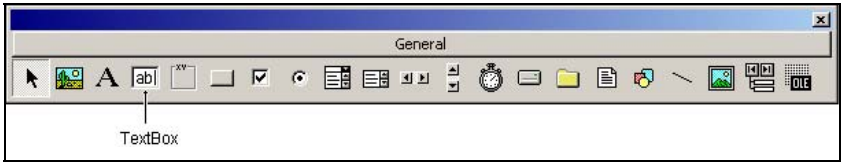


Рис. 3.3. Компонент `TextBox`

Таблица 3.2. *Свойства компонента* `TextBox` *(поле редактирования)*

Свойство	Описание
`Text`	Текст, находящийся в поле компонента
`MaxLength`	Задает максимальное количество символов, которое можно ввести в поле редактирования. Если значение свойства равно 0, то ограничения на количество символов нет
`Font`	Шрифт, используемый для отображения содержимого поля
`MultiLine`	Позволяет (значение — `True`) ввести в поле редактирования несколько строк текста
`ScrollBars`	Управляет отображением полос прокрутки: • 0 — полосы прокрутки не отображать; • 1 — отображать горизонтальную полосу прокрутки; • 2 — отображать вертикальную полосу прокрутки; • 3 — отображать обе полосы прокрутки;

Таблица 3.2 *(окончание)*

Свойство	Описание
`Locked`	Используется для ограничения возможности изменения текста в поле редактирования (если значение свойства равно `True`, то текст в поле редактирования изменить нельзя)
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

Наибольший интерес для программиста представляют события `KeyPress` и `Change`. Событие `KeyPress` возникает в момент нажатия клавиши в поле компонента (до того, как символ появится в поле редактирования), событие `Change` — в момент изменения содержимого поля редактирования.

В поле компонента `TexBox` отображаются все символы, которые пользователь набирает на клавиатуре, что не всегда удобно. Программа, точнее процедура обработки события `KeyPress`, может обеспечить фильтрацию вводимых символов путем замены кода запрещенного символа нулем.

Программа Компонент TextBox (ее форма приведена на рис. 3.4, а текст — в листинге 3.2) демонстрирует использование компонента `TextBox` для ввода данных различного типа. Программа спроектирована таким образом, что в режиме ввода текста в поле редактирования отображаются все символы, в режиме ввода целого числа — только цифры и знак "минус" (если это первый символ). В режиме ввода дробного числа кроме цифр и знака "минус" в поле компонента отображается символ-разделитель (запятая или точка, в зависимости от настройки операционной системы). В режиме ввода пароля программа реагирует только на алфавитно-цифровые клавиши, но вместо символов, которые набирает пользователь, отображаются звездочки. Также в процессе ввода информации в нижней части формы отображается количество введенных символов.

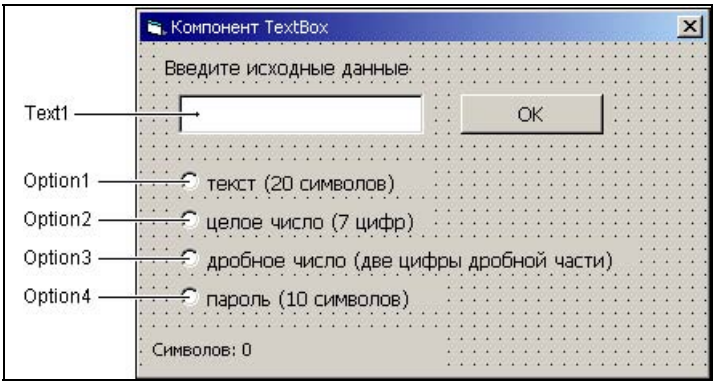


Рис. 3.4. Форма программы Компонент TextBox

Листинг 3.2. Компонент `TextBox`

```
Private Sub Form_Load()
    ' ввод строки любых символов
    Option1.Value = True
    Text1.MaxLength = 20 ' кол-во символов, которое
                         ' можно ввести
End Sub

' нажатие клавиши в поле редактирования
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If Option1.Value Then
        ' в поле редактирования можно вводить любые символы
        Exit Sub
    End If

    If Option2.Value Then
        ' в поле редактирования можно ввести только
        ' целое число
```

```
    Select Case KeyAscii
        Case 48 To 57     ' цифры

        Case 45           ' минус
            If Len(Text1.Text) > 0 Then
                ' в поле уже введены символы
                KeyAscii = 0
            End If

        Case 8            ' клавиша <Backspace>

        Case 13           ' клавиша <Enter>

        Case Else         ' остальные символы запрещены
            KeyAscii = 0
    End Select
End If

If Option3.Value Then
    ' в поле редактирования можно ввести только
    ' дробное число
    Select Case KeyAscii
        Case 48 To 57     ' цифры
        Dim m As Integer ' кол-во цифр дробной части
        If InStr(1, Text1.Text, ",") <> 0 Then
              ' в поле редактирования есть запятая
        m = Len(Text1.Text) - InStr(1, Text1.Text, ",")
              If m = 2 Then KeyAscii = 0
        End If

        Case 45           ' минус
            If Len(Text1.Text) > 0 Then
                ' в поле уже введены символы
```

```
                KeyAscii = 0
            End If

        Case 44, 46       ' точка или запятая
            If KeyAscii = 46 Then
                KeyAscii = 44 ' точку заменим запятой
            End If
            If InStr(1, Text1.Text, ",") <> 0 Then
                ' в поле редактирования уже есть запятая
                KeyAscii = 0
            End If

        Case 8            ' клавиша <Backspace>

        Case 13           ' клавиша <Enter>

        Case Else         ' остальные символы запрещены
            KeyAscii = 0
    End Select
End If

If Option4.Value Then
   ' в поле редактирования предназначено для ввода пароля
     Select Case KeyAscii
        Case 48 To 57     ' цифры
            KeyAscii = 42

        Case 65 To 122    ' буквы латинского алфавита
            KeyAscii = 42

        Case 192 To 255   ' буквы русского алфавита
            KeyAscii = 42
```

```
            Case 8              ' клавиша <Backspace>

            Case 13             ' клавиша <Enter>

            Case Else           ' остальные символы запрещены
                KeyAscii = 0
        End Select

    End If

End Sub

' изменилось содержимое поля Text1
Private Sub Text1_Change()
    Label2.Caption = "Символов: " & Str(Len(Text1.Text))
    If Len(Text1.Text) = 0 Then
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If
End Sub

' текст
Private Sub Option1_Click()
    Text1.Text = ""
    Text1.MaxLength = 20
    Text1.SetFocus
End Sub

' целое число
Private Sub Option2_Click()
    Text1.Text = ""
    Text1.MaxLength = 7
    Text1.SetFocus
End Sub
```

```
' дробное число
Private Sub Option3_Click()
    Text1.Text = ""
    Text1.MaxLength = 11
    Text1.SetFocus
End Sub

' пароль
Private Sub Option4_Click()
    Text1.Text = ""
    Text1.MaxLength = 10
    Text1.SetFocus
End Sub
```

Простой редактор текста

В поле компонента `TextBox` можно разместить несколько строк текста, для этого свойству `MultiLine` надо присвоить значение `True`.

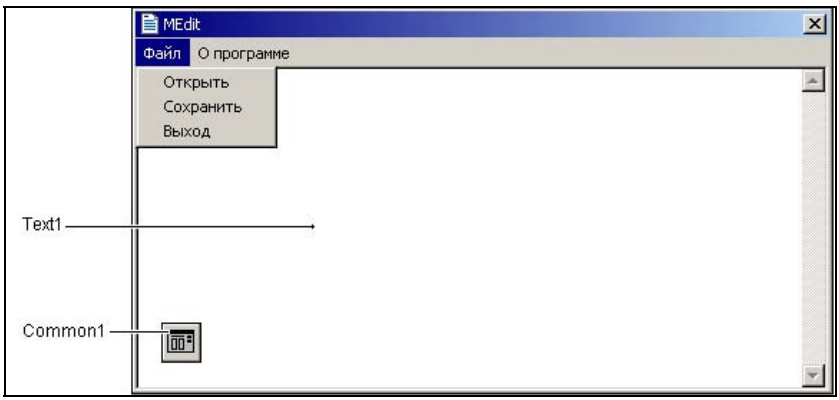


Рис. 3.5. Форма программы Medit

Следующая программа (MEdit) демонстрирует, как на базе компонента TexBox можно создать простой, но вполне работоспособный редактор текста. Редактор MEdit (Micro Editor) позволяет просматривать и вносить изменения в текстовые файлы. Форма программы приведена на рис. 3.5. Значения свойств компонена TextBox1 приведены в табл. 3.3, текст программы — в листинге 3.3.

Таблица 3.3. *Значения свойства компонента* `Text1`

Свойство	Значение
MultiLine	True
Scroll	2 - Vertical

Для выбора имени файла, который надо загрузить в редактор текста (в поле компонента Text1), используется стандартное окно **Открыть**, отображение которого обеспечивает компонент Common1 (CommonDialog). Этот же компонент обеспечивает отображение стандартного окна **Сохранить**. Чтение текста из файла выполняет оператор **Input**, запись текста, находящегося в поле редактирования, — оператор **Write**. В качестве параметра этим операциям передается идентификатор файла (целое число). В рассматриваемой программе идентификатор — это значение функции FreeFile. В принципе, идентификатор можно задать в виде константы, но существует вероятность, что файл с указанным номером уже открыт. Поэтому в качестве идентификатора файла лучше все-таки использовать значение функции FreeFile.

В результате выбора в меню команды **О программе** на экране отображается соответствующее окно (рис. 3.6) Следует обратить внимание на то, что окно **О программе Medit** отображается как *модальный* диалог (см. процедуру About_Click). Это значит, что до тех пор, пока окно о программе не будет закрыто, окно программы, которая активизировала процесс отображения модального диалога, не будет реагировать на действия пользователя.

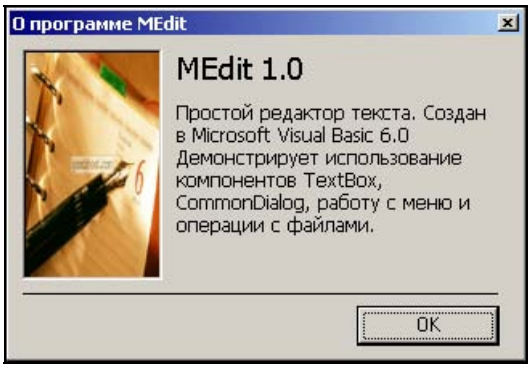


Рис. 3.6. Окно **О программе MEdit** отображается как модальный диалог

Листинг. 3.3. Простейший редактор текста на базе компонента TextBox

```
Dim fName As String      ' имя файла
Dim fChanged As Boolean ' True - текст изменен

' содержимое компонента TextBox изменилось
Private Sub Text1_Change()
    fChanged = True
End Sub

' команда Файл/Открыть
Private Sub FileOpen_Click()
    Dim fn As Integer
    Dim buf As String

    CommonDialog1.FileName = "*.txt"
    CommonDialog1.ShowOpen

    If CommonDialog1.FileName <> "*.txt" Then
        ' пользователь выбрал файл
```

```
        fName = CommonDialog1.FileName
        fn = FreeFile
        ' прочитать текст из файла
        Open fName For Input As fn
        Input #fn, buf
        Close #fn

        Text1.Text = buf
        Text1.SelStart = Len(Text1.Text)
                    ' курсор в конец текста
        frmMEdit.Caption = "MEdit - " & fName
        fChanged = False ' изменений нет

    End If
End Sub

' команда Файл/Сохранить
Private Sub FileSave_Click()
    Dim fn As Integer
    Dim i As Integer

    If fName = "" Then
        ' получить у пользователя имя файла
        CommonDialog1.FileName = ""
        CommonDialog1.DefaultExt = "txt"
        CommonDialog1.Filter = "*.txt"
        CommonDialog1.ShowSave
        If CommonDialog1.FileName <> "" Then
            fName = CommonDialog1.FileName
        Else
            Exit Sub
        End If
    End If
    ' записать текст в файл
    fn = FreeFile
```

```
    Open fName For Output As fn
    Write #fn, Text1.Text
    frmMEdit.Caption = "MEdit - " & fName
    Close #fn
    fChanged = False

End Sub

' команда Файл/Выход
Private Sub FileExit_Click()
    Dim r As Integer
    Dim fn As Integer

    If fChanged Then
        r = MsgBox("Текст изменен._
                    Сохранить измененый текст?", _
                    vbExclamation + vbYesNo, "MEdit")
        If r = vbYes Then
            If fName = "" Then
                ' получить у пользователя имя файла
                CommonDialog1.FileName = ""
                CommonDialog1.DefaultExt = "txt"
                CommonDialog1.Filter = "*.txt"
                CommonDialog1.ShowSave
                MsgBox CommonDialog1.FileName
                fName = CommonDialog1.FileName
            End If
            ' сохранить текст
            fn = FreeFile
            fName = CommonDialog1.FileName
            Open fName For Output As fn
            Write #fn, Text1.Text
            Close #fn
        End If
    End If
```

```
    Unload frmMEdit ' завершить работу программы
End Sub

' Выбор в меню команды "О программе"
Private Sub About_Click()
    frmAbout.Show vbModal
End Sub
```

Компонент *CommandButton*

Компонент `CommandButton` (рис. 3.7) представляет собой командную кнопку. Обычно на кнопке находится текст, но может быть и картинка (в этом случае кнопку называют графической). Свойства компонента `CommandButton` приведены в табл. 3.4.

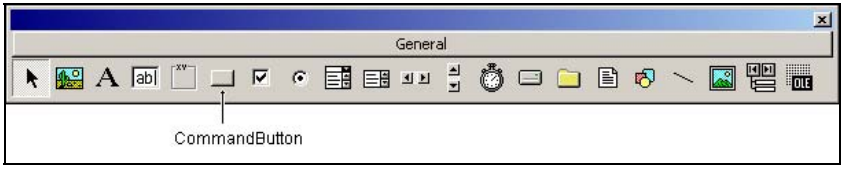


Рис. 3.7. Компонент `CommandButton`

Таблица 3.4. *Свойства компонента* `CommandButton` *(командная кнопка)*

Свойство	Описание
`Caption`	Текст на кнопке
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна. Если значение свойства равно `False`, то кнопка не доступна, т. е. при щелчке на кнопке никаких событий не происходит

Таблица 3.4 *(окончание)*

Свойство	Описание
`Visible`	Позволяет скрыть кнопку (значение — `False`) или сделать ее видимой (значение — `True`)
`Style`	Вид кнопки: • "обычная" (0 — `Standard`); • "графическая" (1 — `Graphical`) (кнопка, на поверхности которой есть картинка)
`Picture`	Свойство задает картинку для "графической" кнопки. Картинка отображается на поверхности формы, если значение свойства `Style` равно `1` (`Graphical`)
`MascColor`	Задает "прозрачный" цвет. Точки картинки, окрашенные этим цветом, на поверхности кнопки не отображаются (при условии, что значение свойства `UseMaskColor` равно `True`)
`UseMaskColor`	При значении `True` точки картинки, окрашенные цветом `MaskColor`, на поверхности кнопки не отображаются
`DisabledPicture`	Задает картинку для недоступной "графической" кнопки. Картинка отображается, если значение свойства `Style` равно `Graphical`, а значение свойства `Enabled` равно `False`
`DownPicture`	Задает картинку для нажатой "графической" кнопки. Картинка отображается в момент нажатия кнопки (если значение свойства `Style` равно `Graphical`)
`ToolTipText`	Задает текст подсказки, которая появляется при позиционировании указателя мыши на кнопке

Следующая программа (Графическая кнопка) (ее форма приведена на рис. 3.8, а текст в листинге 3.4) демонстрирует использование компонента `CommandButton` в качестве графической кнопки.

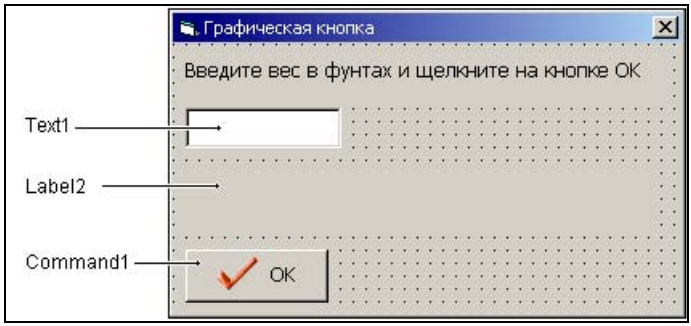


Рис. 3.8. Форма программы Графическая кнопка

Значения свойств компонента `Command` приведены в табл. 3.5, а картинки, которые отображаются на кнопке, — на рис. 3.9 (на одной из них цвет фона — пурпурный).

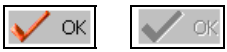


Рис. 3.9. Картинки для доступной (ok.bmp) и для недоступной (ok_d.btn) кнопки `Command1`

Таблица 3.5. *Значения свойств компонента* `Command1`

Свойство	Значение
`Style`	1 – `Graphical`
`Picture`	`ok.bmp`
`MaskColor`	`&H00FF00FF` (пурпурный)
`UseMaskColor`	`True`
`DisabledPicture`	`ok_d.bmp`

Листинг 3.4. Графическая кнопка

```
' изменилось содержимое поля редактирования
Private Sub Text1_Change()
    If Len(Text1.Text) = 0 Then
        ' в поле редактирования нет ни одной цифры
        Command1.Enabled = False ' кнопка Command1 недоступна
    Else
        Command1.Enabled = True  ' кнопка Command1 доступна
    End If
End Sub

' нажатие клавиши в поле редактирования
Private Sub Text1_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
       Case 8
           Label2.Caption = ""
       Case 48 To 57  ' цифра
       Case 44, 46    ' точка или запятая
          If KeyAscii = 46 Then KeyAscii = 44
          If InStr(1, Text1.Text, ",") <> 0 Then KeyAscii = 0
       Case 13 ' <Enter>
           Command1.SetFocus
       Case Else
           KeyAscii = 0
    End Select
End Sub

' щелчок на кнопке OK
Private Sub Command1_Click()
    Dim f As Double  ' вес в фунтах
    Dim kg As Double ' вес в килограммах
```

```
    f = Val(Text1.Text)
    kg = f * 0.495

    Label2.Caption = Str(f) & " ф. - это " & Str(kg) & " кг"

End Sub
```

Компонент *CheckBox*

Компонент `CheckBox` (рис. 3.10) представляет собой переключатель, который может быть в установленном (включенном), сброшенном (выключенном) или в промежуточном состоянии. Когда переключатель включен, то в поле переключателя находится "галочка", когда выключен — "галочки" нет. Свойства компонента `CheckBox` приведены в табл. 3.6.

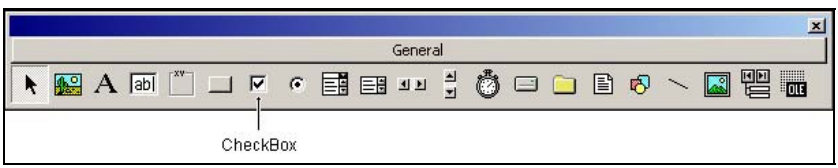


Рис. 3.10. Компонент `CheckBox`

Таблица 3.6. *Свойства компонента* `CheckBox`

Свойство	Описание
`Value`	Состояние переключателя: • `Checked` — переключатель включен (в квадратике есть "галочка"); • `Unchecked` — переключатель выключен (нет "галочки")
`Caption`	Текст, который находится справа от переключателя
`Font`	Шрифт, используемый для отображения пояснительного текста

Таблица 3.6 *(окончание)*

Свойство	Описание
`Enabled`	Признак доступности переключателя. Если значение свойства равно `True`, то переключатель доступен. Если значение свойства равно `False`, то переключатель не доступен
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Состояние переключателя изменяется в результате щелчка на его изображении (если значение свойства `Enabled` равно `True`). При этом возникает событие `Click`.

Следующая программа (Кафе) (ее форма приведена на рис. 3.11, а текст — в листинге 3.5) демонстрирует использование компонента `CheckBox`. Программа вычисляет стоимость заказа в кафе. В начале работы все переключатели доступны (за исключением переключателя **соус**, который становится доступным только в результате выбора переключателя **Биг-Мак**).

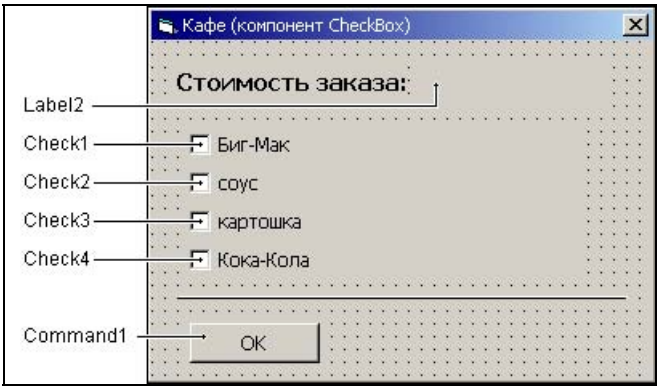


Рис. 3.11. Форма программы Кафе

Стоимость заказа отображается в поле компонента Label2 и изменяется при каждом изменении состояния какого-либо переключателя. Следует обратить внимание на процедуру обработки события Change компонента Label2 (поля отображения суммы заказа). Событие Change возникает всякий раз, когда какая-либо из процедур обработки события Click на переключателе изменяет текст в поле компонента Label2 (выводит стоимость заказа). Процедура обработки события Change контролирует сумму заказа и, если она равна нулю, делает кнопку **OK** недоступной.

Листинг 3.5. Кафе (компонент CheckBox)

```
Const BIGMAC = 54
Const SAUCE = 7.5
Const CHIPS = 24.5
Const COCACOLA = 18.5

Dim sum As Double ' сумма заказа

' Биг-Мак
Private Sub Check1_Click()
    If Check1.Value = vbChecked Then
        sum = sum + BIGMAC
        Check2.Enabled = True
              ' теперь переключатель "соус" доступен
    Else
        sum = sum - BIGMAC
        If Check2.Value = vbChecked Then
            Check2.Value = vbUnchecked
            ' сбросить переключатель "соус"
            ' цену соуса вычитать не надо,
            ' так как в результате выполнения предыдущей
            ' инструкции возникает событие Click для Check2,
            ' процедура обработки которого и отнимет
            ' цену соуса
```

```
        End If
        Check2.Enabled = False
            ' теперь переключатель "соус" доступен
    End If
    Label2.Caption = Str(sum) & " руб."
End Sub

' соус
Private Sub Check2_Click()
    If Check2.Value = vbChecked Then
        sum = sum + SAUCE
    Else
        sum = sum - SAUCE
    End If
    Label2.Caption = Str(sum) & " руб."
End Sub

' картошка
Private Sub Check3_Click()
    If Check3.Value = vbChecked Then
        sum = sum + CHIPS
    Else
        sum = sum - CHIPS
    End If
    Label2.Caption = Str(sum) & " руб."
End Sub

' Кока-Кола
Private Sub Check4_Click()
    If Check4.Value = vbChecked Then
        sum = sum + COCACOLA
    Else
        sum = sum - COCACOLA
    End If
```

```
    Label2.Caption = Str(sum) & " руб."
End Sub

' изменился текст в поле Label2
Private Sub Label2_Change()
    If sum <> 0 Then
        Command1.Enabled = True
    Else
        Command1.Enabled = False
    End If
End Sub
```

Компонент *OptionButton*

Компонент `OptionButton` (рис. 3.12) представляет собой кнопку (переключатель), которая может находиться в выбранном (включенном) или в невыбранном (выключенном) состоянии. Следует обратить внимание на то, что состояние кнопки зависит от состояния других кнопок (компонентов `OptionButton`), которые находятся на форме. В каждый момент времени только одна из кнопок может находиться в выбранном состоянии, однако возможна ситуация, когда ни одна из кнопок не выбрана.

Несколько компонентов `OptionButton` можно объединить в группу, разместив их в поле компонента `Frame`. Состояние компонентов, принадлежащих к одной группе, не зависит от состояния компонентов, принадлежащих другой группе. Свойства компонента `OptionButton` приведены в табл. 3.7.

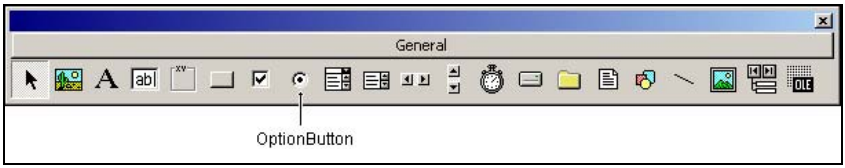


Рис. 3.12. Компонент `OptionButton`

Таблица 3.7. *Свойства компонента* `OptionButton`

Свойство	Описание
`Caption`	Текст, который находится справа от кнопки
`Value`	Состояние кнопки. Если кнопка выбрана, то значение свойства равно `True`, если кнопка не выбрана, значение равно `False`
`Font`	Шрифт, используемый для отображения поясняющего текста
`Enabled`	Признак доступности компонента. Если значение свойства равно `True`, то переключатель доступен. Если значение свойства равно `False`, то переключатель не доступен
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Состояние кнопки изменяется в результате щелчка на ее изображении. При этом возникает событие `Click`.

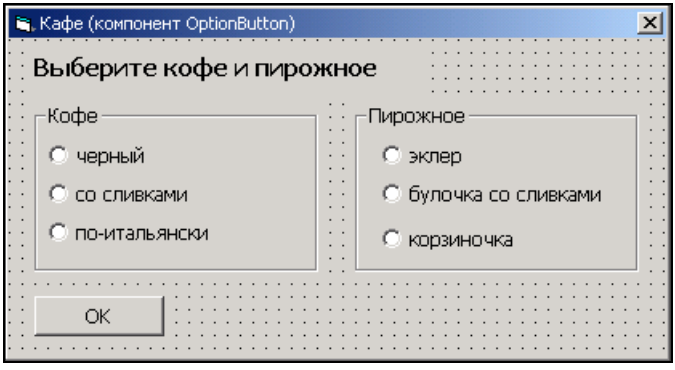


Рис. 3.13. Форма программы Кафе (компонент `OptionButton`)

Следующая программа (ее форма приведена на рис. 3.13, а текст — в листинге 3.6) демонстрирует использование компонента `OptionButton`. Компоненты `OptionButton` объединены в две группы (размещены в поле двух компонентов `Frame`). Это позволяет установить в выбранное состояние два переключателя одновременно — по одному в каждой группе. Для каждого компонента `OptionButton` в программе есть своя процедура обработки события `Click`, которая фиксирует вид выбранного кофе и пирожного в переменных `coffee` и `cake`. Процедура обработки события `Click` на кнопке **OK** проверяет состояние переключателей и, если в какой-либо группе нет выбранного переключателя, предлагает сделать выбор (выводит сообщение).

Листинг 3.6. Кафе (компонент `OptionButton`)

```
Dim coffee As String  ' кофе
Dim cake As String    ' пирожное

' процедуры обработки события Ckick

' черный кофе
Private Sub Option1_Click()
    coffee = Option1.Caption
End Sub

' кофе со сливками
Private Sub Option2_Click()
    coffee = Option2.Caption
End Sub

' кофе по-итальянски
Private Sub Option3_Click()
    coffee = Option3.Caption
End Sub
```

```
' эклер
Private Sub Option4_Click()
    cake = Option4.Caption
End Sub

' булочка со сливками
Private Sub Option5_Click()
    cake = Option5.Caption
End Sub

' корзиночка
Private Sub Option6_Click()
    cake = Option6.Caption
End Sub

' щелчок на кнопке OK
Private Sub Command1_Click()
    If coffee = "" Or cake = "" Then
        MsgBox "Надо выбрать кофе и пирожное",_
                vbInformation, "Кафе"
    Else
        MsgBox "Вы выбрали кофе " & coffee & vbCr & _
               " и пирожное " & cake & vbCr & _
               "Правильный выбор!", vbOKOnly, "Кафе"
    End If
End Sub
```

Компонент *ComboBox*

Компонент `ComboBox` (рис. 3.14) представляет собой комбинацию поля ввода и списка, что позволяет ввести данные в поле редактирования путем набора на клавиатуре или путем выбора из списка. Свойства компонента приведены в табл. 3.8.

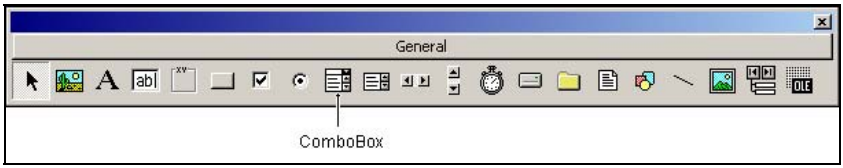


Рис. 3.14. Компонент `ComboBox`

Таблица 3.8. *Свойства компонента* `ComboBox`

Свойство	Описание
`Style`	Вид списка: • поле ввода и раскрывающийся список (0 — `Drop-down Combo`); • поле ввода со списком (1 — `Simple Combo`); • раскрывающийся список (`Dropdown List`)
`Text`	Содержимое поля редактирования (данные, введенные пользователем с клавиатуры или выбранные из списка)
`List`	Элементы списка — массив строк
`ListCount`	Количество элементов списка
`ListIndex`	Номер выбранного элемента списка. Если ни один из элементов списка не был выбран, то значение свойства равно `-1`. Элементы списка нумеруются с нуля
`Sorted`	Признак необходимости выполнить сортировку списка после добавления очередного элемента (значение свойства можно задать только во время разработки формы)
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

Список, отображаемый в поле компонента `ComboBox`, можно сформировать как во время создания формы, так и во время ра-

боты программы. Чтобы сформировать список во время создания формы, надо раскрыть список **List** и ввести элементы списка (рис. 3.15).

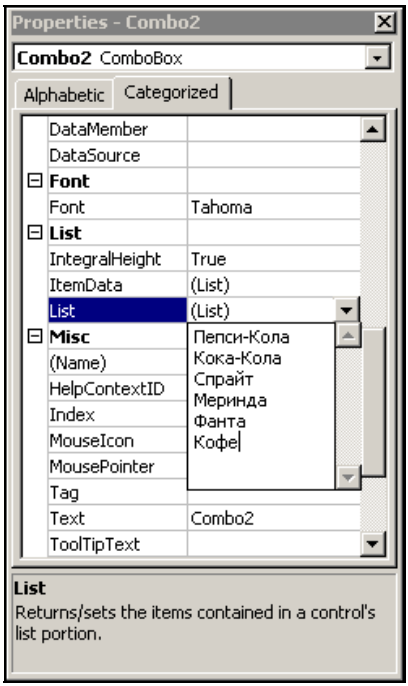


Рис. 3.15. Формирование списка компонента `ComboBox` во время создания формы

Формирование списка во время работы программы обеспечивает метод `AddItem`. В качестве параметра метода надо указать добавляемый элемент списка и (необязательно) позицию, куда надо этот элемент поместить. Например, следующий фрагмент кода формирует список компонента `Combo2`.

```
Combo2.AddItem ("Пепси-Кола")
Combo2.AddItem ("Кока-Кола")
Combo2.AddItem ("Спрайт")
```

```
Combo2.AddItem ("Меринда")
Combo2.AddItem ("Фанта")
Combo2.AddItem ("Кофе")
Combo2.AddItem ("Чай")
Combo2.AddItem ("Минеральная вода")
Combo2.AddItem ("Сок")
```

Если свойству `Sorted` компонента `Combo2` во время разработки формы было присвоено значение `True`, то список будет упорядочен по алфавиту.

Следующая программа (Любимый напиток) демонстрирует использование компонента `ComboBox` для ввода данных. Форма программы приведена на рис. 3.16, текст — в листинге 3.7. Значения свойств компонентов приведены в табл. 3.9. Программа спроектирована таким образом, что выбранный в списке элемент отображается в соответствующем поле `Label`. Кроме того, если пользователь введет в поле редактирования списка `Combo1` тот напиток, которого нет в списке, то он будет добавлен в список.

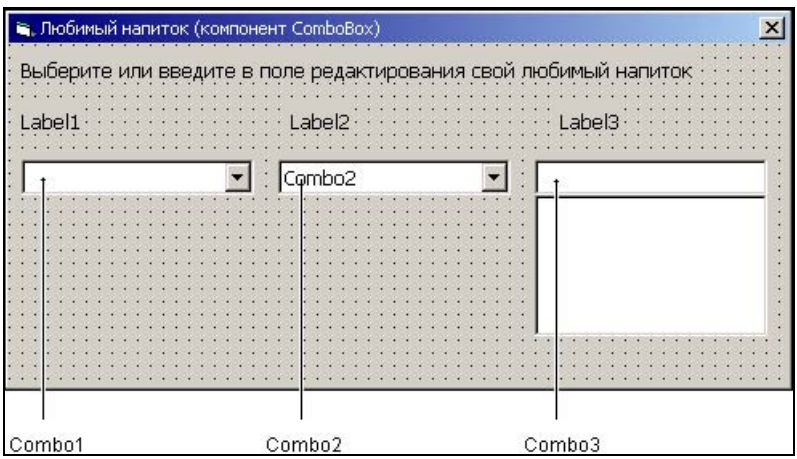


Рис. 3.16. Форма программы Любимый напиток (компонент `ComboBox`)

Таблица 3.9. *Значения свойств компонента* `ComboBox`

Свойство	Значение
Combo1.Style	0 – Dropdown Combo
Combo1.Sorted	True
Combo1.List	Пепси-Кола
	Кока-Кола
	Спрайт
	Меринда
	Кофе
Combo2.Style	1 – Dropdown List
Combo2.Sorted	True
Combo3.Style	2 – Simple Combo
Combo3.Sorted	True

Листинг. 3.7. Любимый напиток (ComboBox)

```
Private Sub Form_Load()

    ' если во время создания формы свойству Sorded было
    ' присвоено значение True, то список будет упорядочен

    ' сформировать список компонента Combo2
    Combo2.AddItem ("Пепси-Кола")
    Combo2.AddItem ("Кока-Кола")
    Combo2.AddItem ("Спрайт")
    Combo2.AddItem ("Меринда")
    Combo2.AddItem ("Фанта")
    Combo2.AddItem ("Кофе")
    Combo2.AddItem ("Чай")
    Combo2.AddItem ("Минеральная вода")
```

```
    ' сформировать список компонента Combo3
    Combo3.AddItem ("Пепси-Кола")
    Combo3.AddItem ("Кока-Кола")
    Combo3.AddItem ("Спрайт")
    Combo3.AddItem ("Меринда")
    Combo3.AddItem ("Фанта")
    Combo3.AddItem ("Кофе")
    Combo3.AddItem ("Чай")
    Combo3.AddItem ("Минеральная вода")

    Label1.Caption = ""
    Label2.Caption = ""
    Label3.Caption = ""
End Sub

' выбор элемента в списке Combo1
Private Sub Combo1_Click()
    Label1.Caption = Combo1.List(Combo1.ListIndex)
End Sub

' нажатие клавиши в поле редактирования компонента Combo1
Private Sub Combo1_KeyPress(KeyAscii As Integer)
    Dim found As Boolean ' True - название напитка, которое
                         ' ввел пользователь, уже есть в
                         ' списке
    Dim i As Integer

    If KeyAscii = 13 Then ' пользователь нажал <Enter>
        Combo1.Text = LTrim(RTrim(Combo1.Text))
        ' проверить, есть ли в списке List строка, которую
        ' пользователь ввел в поле редактирования
        found = False ' пусть нет
        i = 0
        Do While (i < Combo1.ListCount) And Not found
```

```
            If Combo1.List(i) = Combo1.Text Then
                found = True
            Else
                i = i + 1
            End If
        Loop
        If Not found Then
            ' строки в списке нет
            Combo1.AddItem (Combo1.Text) ' добавим
            MsgBox "В список добавлен элемент " & _
                   Chr(34) & Combo1.Text & Chr(34)
            Label1.Caption = Combo1.Text
        End If
    End If

End Sub

' выбор элемента в списке Combo2
Private Sub Combo2_Click()
    Label2.Caption = Combo2.List(Combo2.ListIndex)
End Sub

' выбор элемента в списке Combo3
Private Sub Combo3_Click()
    Label3.Caption = Combo3.List(Combo3.ListIndex)
End Sub
```

Компонент *Timer*

Компонент `Timer` (рис. 3.17) обеспечивает генерацию последовательности событий `Timer`. Свойства компонента приведены в табл. 3.10.

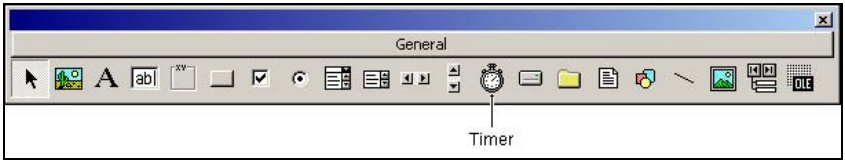


Рис. 3.17. Компонент `Timer`

Таблица 3.10. *Свойства компонента* `Timer`

Свойство	Описание
`Interval`	Период генерации события `Timer`.
	Задается в миллисекундах
`Enabled`	Разрешение работы.
	Если значение свойства равно `True`, то таймер генерирует событие `Timer` с периодом `Interval`

Использование компонента `Timer` демонстрирует программа Будильник (ее форма приведена на рис. 3.18, а текст — в листинге 3.8). Значения свойств компонента `Timer1` главной формы программы приведены в табл. 3.11.

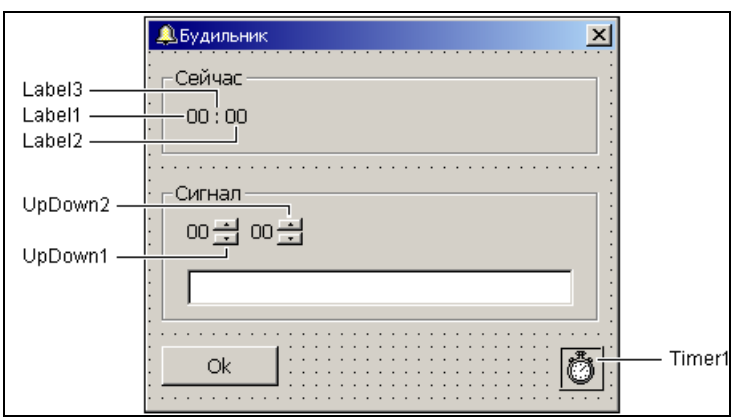


Рис. 3.18. Форма программы Будильник

Таблица 3.11. *Значения свойств компонента* `Timer1`

Свойство	Значение
Interval	500
Enabled	True

Листинг 3.8. Будильник (компонент `Timer`)

```
Private Declare Function PlaySound Lib "winmm.dll"_
        Alias "PlaySoundA" (ByVal lpszName As String,_
        ByVal hModule As Long, ByVal dwFlags As Long) As Long

' время на индикаторе
Dim h As Integer ' часы
Dim m As Integer ' минуты

Dim Alarm As Boolean ' True - будильник "установлен"

Private Sub Form_Load()
    h = Hour(Time)
    m = Minute(Time)
    Label1.Caption = Format(h, "00")
    Label2.Caption = Format(m, "00")
End Sub

' сигнал от таймера
Private Sub Timer1_Timer()

    ' проверим, надо ли обновить индикатор "часы"
    If h <> Hour(Time) Then
            ' надо
        h = Hour(Time)
        Label1.Caption = Format(h, "00")
    End If
```

```
    ' проверим, надо ли обновить индикатор "минуты"
    If m <> Minute(Time) Then
        ' надо
        m = Minute(Time)
        Label2.Caption = Format(m, "00")
    End If

    ' скрыть/отобразить двоеточие на индикаторе
    Label3.Visible = Not Label3.Visible

    If Alarm Then
        ' будильник установлен
        If UpDown1.Value = Hour(Time) _
                And UpDown2.Value = Minute(Time) Then
          Alarm = False
          Form2.Label1 = Format(UpDown1.Value, "00") & ":" & _
                       Format(UpDown2.Value, "00")
          Form2.Label2 = Form1.Text1
          PlaySound "ring.wav", 0, &H1
          Form2.Show ' отобразить окно с сообщением
        End If
    End If

End Sub

' щелчок на одной из кнопок компонента UpDown изменяет
' значение свойства Value, в результате возникает
' событие Change
Private Sub UpDown1_Change()
    Label4.Caption = Format(UpDown1.Value, "00")
End Sub
```

```
Private Sub UpDown2_Change()
    Label5.Caption = Format(UpDown2.Value, "00")
End Sub

' щелчок на кнопке Ok
Private Sub Command1_Click()
    Dim r As Integer
    Dim h, m As Integer    ' текущее время
    Dim ah, am As Integer ' время, на которое установлен
                          ' будильник

    h = Hour(Time)
    m = Minute(Time)
    ah = UpDown1.Value
    am = UpDown2.Value

    ' проверить, правильно ли пользователь установил время
    If (ah = 0 And am = 0) Or _
       (ah < h) Or (ah = h And am < m) Then
        r = MsgBox("Вы установили будильник на"_
            & Format(ah, "00") & ":" & Format(am, "00")_
            & vbCr & "Это действительно так?", _
            vbYesNo + vbExclamation, "Будильник")
        If r = vbNo Then Exit Sub
    End If
    Form1.Caption = "Будильник - " + Format(ah, "00")_
                    & ":" & Format(am, "00")
    Form1.WindowState = vbMinimized
    Alarm = True

End Sub

' пользователь щелкнул на кнопке "Закрыть окно"
Private Sub Form_QueryUnload(Cancel As Integer,_
                        UnloadMode As Integer)
```

```
    If MsgBox("Завершить работу программы?", vbYesNo,_
              "Будильник") = vbNo Then
        Cancel = True
    End If

End Sub

' фокус переместился в поле Text1, например
' в результате нажатия клавиши <Tab>
Private Sub Text1_GotFocus()
    Text1.SelStart = Len(Text1.Text) ' курсор в конец строки
End Sub
```

Компонент `Timer1` обеспечивает отображение текущего времени в полях компонентов `Label1` (часы), `Label2` (минуты) и `Label3` (двоеточие). Непосредственный вывод времени осуществляет процедура `Timer1_Timer`. Она сравнивает текущее время с отображаемым и, если надо, обновляет показания индикатора. Кроме того, она изменяет значение свойства `Visible` компонента `Label3` на противоположное. В результате двоеточие мигает с периодом, равным удвоенному значению свойства `Interval` таймера. Во время работы программы (после того как пользователь задаст время срабатывания будильника, текст сообщения и щелкнет на кнопке **OK**) окно программы на экране не отображается (процедура обработки события `Click` на кнопке `Command1` сворачивает окно). Однако таймер продолжает работать, т. е. периодически вызывается процедура `Timer1_Timer`. В тот момент, когда текущее время совпадает со временем срабатывания будильника, на экране появляется окно сообщения. В рассматриваемом примере в форме окна сообщения также есть таймер (рис. 3.19). Процедура обработки сигнала от таймера инвертирует значение свойства `Visible` компонента `Label1`, в результате чего сообщение мигает. Модуль формы сообщения приведен в листинге 3.9.

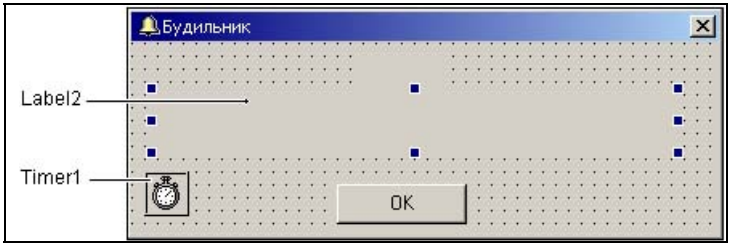


Рис. 3.19. Форма сообщения программы Будильник

Листинг 3.9. Модуль формы сообщения программы Будильник

```
' щелчок на кнопке OK
Private Sub Command1_Click()
    Unload Form2 ' закрыть окно сообщения
End Sub

' событие Unload возникает в момент завершения
' работы программы
Private Sub Form_Unload(Cancel As Integer)
    ' сейчас главное окно программы (в котором пользователь
    ' задал время сигнала будильника) закрыто
    Form1.Caption = "Будильник"
    Form1.WindowState = vbNormal ' раскрыть главное
                                 ' окно программы
End Sub

' сигнал от таймера
Private Sub Timer1_Timer()
    ' скрыть/показать сообщение
    Label2.Visible = Not Label2.Visible
End Sub
```

Компонент *PictureBox*

Компонент `PictureBox` (рис. 3.20) предназначен для отображения иллюстраций (чтобы иллюстрация отображалась без искажений, размер компонента должен быть пропорционален размеру иллюстрации). Иллюстрацию можно задать во время создания формы (присвоив значение свойству `Picture`) или загрузить из файла. На поверхности компонента можно рисовать. Свойства компонента `PictureBox` приведены в табл. 3.12.

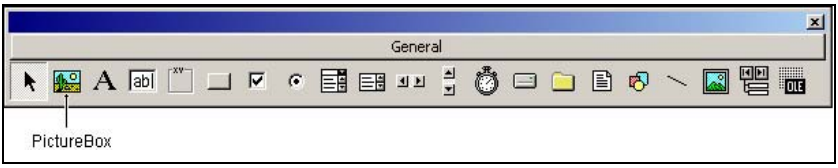


Рис. 3.20. Компонент `PictureBox`

Таблица 3.12. *Свойства компонента* `PictureBox`

Свойство	Описание
`Picture`	Картинка, отображаемая в поле компонента. Задать картинку можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`AutoSize`	Свойство разрешает (значение — `True`) или запрещает (значение — `False`) автоматическое изменение размера компонента (области вывода иллюстрации) в соответствии с размером картинки, загруженной в компонент
`BorderStyle`	Стиль границы компонента. Если значение свойства равно 1, то граница — тонкая линия, если значение равно 0, то граница не отображается
`BackColor`	Цвет фона компонента. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы

Таблица 3.12 *(продолжение)*

Свойство	Описание
`Font`	Шрифт, которым метод `Print` выводит текст
`ForeColor`	Для метода `Print` задает цвет символов, для методов вычерчивания графических примитивов (объектов) — цвет линий
`FillColor`	Задает цвет закраски внутренних областей графических примитивов (объектов), вычерчиваемых в поле (на поверхности) компонента
`FillStyle`	Стиль закраски графических объектов, вычерчиваемых в поле компонента соответствующими методами: • 0 (`Solid`) — сплошная заливка; • 1 (`Transparent`) — закраска "прозрачным" цветом; • 2 (`HorizontalLine`) — горизонтальная штриховка; • 3 (`VerticalLine`) — вертикальная штриховка. Цвет линий штриховки определяет свойство `FillColor`
`DrawStyle`	Вид контура графических объектов, вычерчиваемых в поле компонента соответствующими методами: • 0 (`Solid`) — сплошная линия; • 1 (`Dash`) — пунктирная линия; • 2 (`Dot`) — линия из точек; • 3 (`Dash-Dot`) — линия вида "точка-тире"; • 4 (`Dash-Dot-Dot`) — линия вида "тире-точка-точка"; • 5 (`Transparent`) — "прозрачная" линия
`DrawWidth`	Толщина линий для графических объектов
`ScaleWidth`	Ширина рабочей области компонента, т. е. без учета ширины левой и правой границ. Единицу измерения задает свойство `ScaleMode`
`ScaleHeight`	Высота рабочей области компонента, т. е. без учета ширины нижней и верхней границ компонента. Единицу измерения задает свойство `ScaleMode`

Таблица 3.12 *(окончание)*

Свойство	Описание
`ScaleMode`	Задает единицу измерения размеров компонента и объектов на его поверхности. Значение этого свойства не влияет на единицы измерения свойств `Width` и `Height` (независимо от него их значения измеряются в твипах)
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Следующая программа (График) демонстрирует использование компонента `PictureBox` в качестве фона для вывода графики. Форма программы приведены на рис. 3.21, значения свойств компонента `Picture1` — в табл. 3.13.

Рис. 3.21. Форма программы График (компонент `PictureBox`)

Программа строит в поле компонента `Picture1` график изменения курса доллара (вопросы программирования графики подробно рассмотрены в *5 главе*). Поскольку курс меняется незна-

чительно (в пределах единицы), то программа строит график в отклонениях. Текст программы приведен в листинге 3.10, окно — на рис. 3.22.

Таблица 3.13. *Значения свойств компонента* `Picture1`

Свойство	Значение
BorderStyle	0 — None
AutoSize	True
Picture	back.bmp
ScaleMode	3 — Pixel

Листинг 3.10. График (компонент PictureBox)

```
Const N = 10 ' данные за 10 дней
Dim usd(1 To 31) As Double ' значения курса
Dim min, max As Double

Private Sub Form_Load()

    Dim i As Integer

    ' данные о курсе доллара
   usd(1) = 27.84: usd(2) = 28: usd(3) = 28.12: usd(4) = 28
   usd(5) = 28: usd(6) = 28.95: usd(7) = 28.89: usd(8) =
28.87
   usd(9) = 28.75: usd(10) = 28.75

    ' умножим значения на 100, чтобы работать с целыми
      числами
    For i = 1 To N
        usd(i) = usd(i) * 100
    Next i
```

```
    ' найти минимальное значение
    min = usd(1)
    For i = 2 To N
        If usd(i) < min Then min = usd(i)
    Next i

    ' найти максимальное значение
    max = usd(1)
    For i = 2 To N
        If usd(i) > max Then max = usd(i)
    Next i

End Sub

' график строит процедура обработки события Paint
Private Sub Picture1_Paint()

    Dim k As Double ' количество пикселов на единицу
                    ' значения функции

    ' График строим в приращениях.
    ' Изменение координаты следующей точки относительно
    ' предыдущей:
    Dim dx As Integer ' по X
    Dim dy As Integer ' по Y

    Dim i As Integer

    Dim cx, cy As Integer ' положение указателя графического
                          ' вывода

    ' заголовок
    Picture1.CurrentX = 100
    Picture1.CurrentY = 5
    Picture1.Font.Size = 14
```

```
Picture1.Font.Bold = False
Picture1.ForeColor = vbWhite
Picture1.Print "Изменение курса доллара"

Picture1.CurrentX = 101
Picture1.CurrentY = 6
Picture1.Font.Size = 14
Picture1.Font.Bold = False
Picture1.ForeColor = vbBlack
Picture1.Print "Изменение курса доллара"

Picture1.Font.Size = 10
Picture1.Font.Bold = False
' Picture1.DrawWidth = 2

dx = (Picture1.ScaleWidth - 20) / N
k = (Picture1.ScaleHeight - 50) / (max - min)

' координаты первой точки графика
Picture1.CurrentX = 20
Picture1.CurrentY = (Picture1.ScaleHeight - 15)_
                    - (usd(1) - min) * k



' так как метод Print изменит положение указателя
' вывода, сохраним значения CurrentX и CurrentY
cx = Picture1.CurrentX
cy = Picture1.CurrentY

Picture1.Print usd(1) / 100

' указатель вывода переместим в точку, в которой
' он был до выполнения метода Print
Picture1.CurrentX = cx
Picture1.CurrentY = cy
```

```
    ' остальные точки графика
    For i = 2 To N
        ' вычислить координату Y относительно
        ' предыдущей точки
        dy = (usd(i - 1) - usd(i)) * k

        ' провести линию из предыдущей точки в текущую
        Picture1.Line -Step(dx, dy)

        cx = Picture1.CurrentX
        cy = Picture1.CurrentY

        Picture1.Print usd(i) / 100

        Picture1.CurrentX = cx
        Picture1.CurrentY = cy
    Next i
End Sub
```

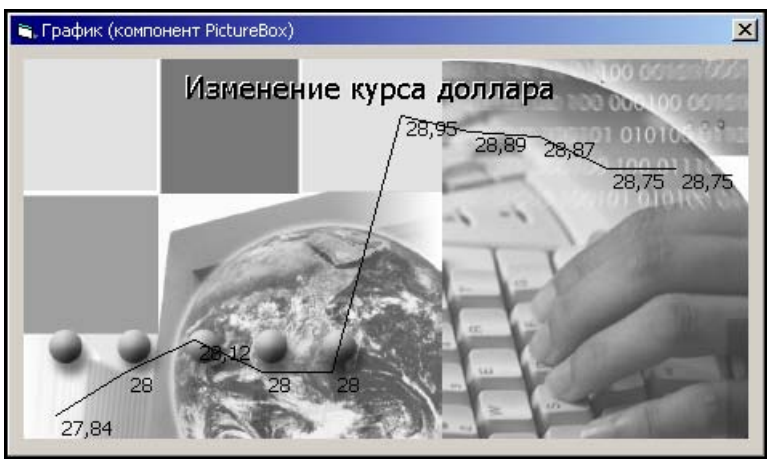


Рис. 3.22. График изменения курса доллара выведен на поверхность компонента `PictureBox`

Компонент *Image*

Компонент `Image` (рис. 3.23) предназначен для отображения иллюстраций. Основное отличие компонента `Image` от компонента `PictureBox` состоит в том, что он позволяет масштабировать иллюстрации. Еще одно отличие: на поверхности компонента `Image` нельзя рисовать. Иллюстрацию, отображаемую в поле компонента, можно задать во время создания формы (присвоив значение свойству `Picture`) или загрузить из файла. Свойства компонента `Image` приведены в табл. 3.14.

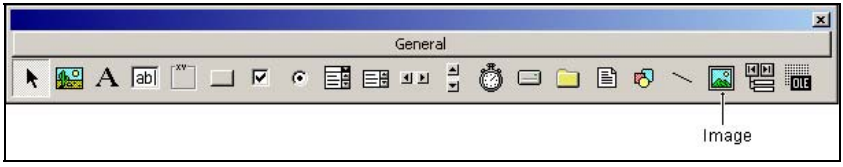


Рис. 3.23. Компонент `Image`

Таблица 3.14. *Свойства компонента* `Image`

Свойство	Описание
`Picture`	Картинка (иллюстрация), отображаемая в поле компонента. Задать иллюстрацию можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`Stretch`	Указывает (значение — `True`) на то, что нужно выполнить масштабирование иллюстрации (сжать или растянуть) в соответствии с размером компонента. Если размер компонента непропорционален размеру иллюстрации, то иллюстрация будет искажена. Если значение свойства равно `False`, то масштабирование не выполняется

Таблица 3.14 *(окончание)*

Свойство	Описание
`Width`, `Height`	Размер компонента. Если значение свойства `Strech` равно `False`, то после загрузки иллюстрации значение свойства `Width` будет равно ширине иллюстрации, значение свойства `Height` будет равно высоте
`Height`	Высота компонента

Следующая программа (ее форма приведена на рис. 3.24) демонстрирует использование компонента `Image`. Программа позволяет просмотреть иллюстрации, которые находятся в текущем каталоге. Иллюстрация отображается в поле компонента `Image1`. Если размер иллюстрации больше размера компонента, то иллюстрация автоматически масштабируется. Компонент `Label1` используется для отображения информации об иллюстрации. Кнопка **Дальше** (`Command1`) обеспечивает переход к следующей иллюстрации. Текст программы приведен в листинге 3.11.

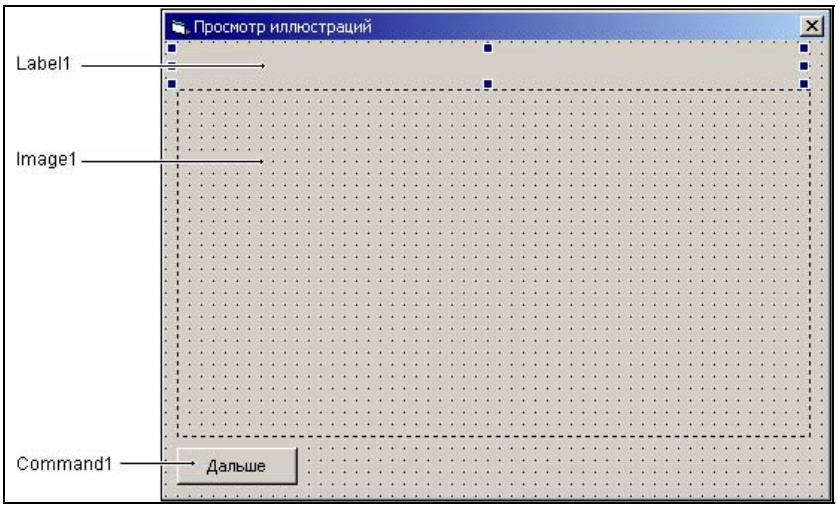


Рис. 3.24. Форма программы Просмотр иллюстраций

Листинг 3.11. Просмотр иллюстраций (компонент Image)

```
Dim w, h As Integer
                    ' первоначальный размер компонента Image
Dim aPicture As String  ' иллюстрация (имя файла)

Private Sub Form_Load()

    ' запомнить размер компонента Image1
    w = Image1.Width
    h = Image1.Height

    aPicture = Dir("*.jpg") ' получить имя jpg-файла
    If aPicture = "" Then
        Label1.Caption = "В каталоге " & CurDir_
                         & " иллюстраций нет"
        Command1.Enabled = False
        Exit Sub
    End If

    DrawPicture (aPicture) ' отобразить иллюстрацию

    aPicture = Dir ' получить имя файла следующей иллюстрации
    If aPicture = "" Then Command1.Enabled = False
End Sub


' отображает иллюстрацию в поле компонента Image1
Sub DrawPicture(aPicture As String)

    Dim kw, kh  As Double ' коэф. масштабирования иллюстрации
                          ' по ширине и высоте
    Dim k As Double       ' общий коэф. масштабирования
                          ' иллюстрации
```

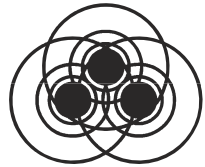
```
    Image1.Visible = False
    Image1.Stretch = False
    Image1.Picture = LoadPicture(aPicture)
    If (Image1.Width > w) Or (Image1.Height > h) Then
        ' размер иллюстрации больше, чем размер
        ' компонента Image
        kw = w / Image1.Width
        kh = h / Image1.Height
        ' чтобы иллюстрация отображалась без искажения,
        ' коэф. масштабирования по ширине и высоте
        ' должен быть одинаковый
        If kw < kh Then
            k = kw
        Else
            k = kh
        End If

    End If

    ' информация об иллюстрации
    Label1.Caption = "Файл: " & aPicture & _
                     " Размер:" & Image1.Width & "x" &_
                       Image1.Height & _
                     " Масштаб:" & Format(k, "0.00")

    Image1.Width = Image1.Width * k
    Image1.Height = Image1.Height * k
    Image1.Stretch = True ' масштабировать
    Image1.Visible = True

End Sub

' щелчок на кнопке Дальше
Private Sub Command1_Click()
    DrawPicture (aPicture)
```

```
    aPicture = Dir ' получить имя файла следующей иллюстрации
    If aPicture = "" Then
        ' в текущем каталоге больше нет иллюстраций
        Command1.Enabled = False
                    ' кнопка Дальше теперь недоступна
    End If
End Sub
```

Непосредственное отображение иллюстрации обеспечивает функция `DrawImage`. В начале ее работы свойству `Strech` присваивается значение `False`. Делается это для того, чтобы узнать истинный размер иллюстрации. После загрузки иллюстрации, если ее размер превышает размер компонента `Image1`, вычисляются коэффициенты масштабирования: сначала по ширине и высоте, а затем — общий. Далее устанавливаются размеры компонента `Image` пропорционально размерам загруженной иллюстрации, свойству `Strech` присваивается значение `True`. В результате иллюстрация масштабируется и отображается без искажения (рис. 3.25).

Рис. 3.25. Окно программы Просмотр иллюстраций

Глава 4



Дополнительные компоненты

Помимо базовых компонентов (универсальных, общего назначения), которые находятся на вкладке **General** палитры компонентов, программист может использовать и другие (дополнительные) компоненты. В этой главе рассказывается о том, как их можно подключить, а также приведено описание некоторых компонентов библиотеки Microsoft Windows Common Control 6.0 (SP4).

Потенциально доступные дополнительные компоненты, точнее библиотеки, в которых они находятся, перечислены на вкладке **Controls** окна **Components** (рис. 4.1), которое становится доступным в результате выбора в меню **Projects** команды **Components**.

Чтобы компонент стал доступным, надо подключить библиотеку, в которой он находится, т. е. установить в выбранное состояние переключатель (перед именем библиотеки) и щелкнуть на кнопке **Применить** (рис. 4.2). В палитре компонентов появятся значки тех компонентов, которые находятся в выбранной библиотеке. В качестве примера на рис. 4.3 приведен вид палитры компонентов после подключения библиотек Microsoft Windows Common Control 6.0 (SP4) и Microsoft Windows Common Control-2 6.0. В палитре появились новые компоненты: `StatusBar`, `ProgressBar`, `UpDown` и др. Далее в этой главе приведено краткое описание этих компонентов, а также примеры их использования.

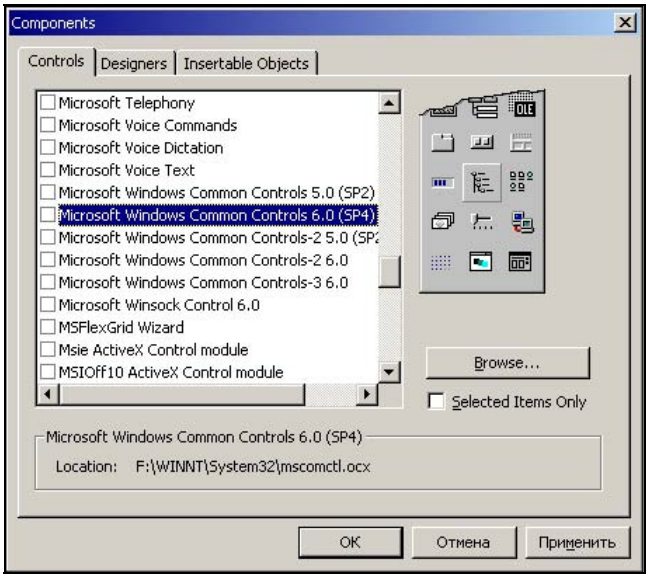


Рис. 4.1. Диалоговое окно **Components** (вкладка **Controls**)

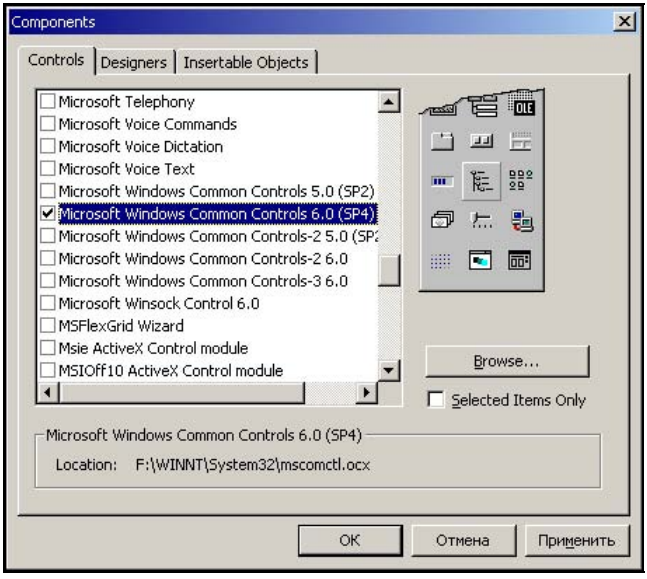


Рис. 4.2. Подключение библиотеки компонентов

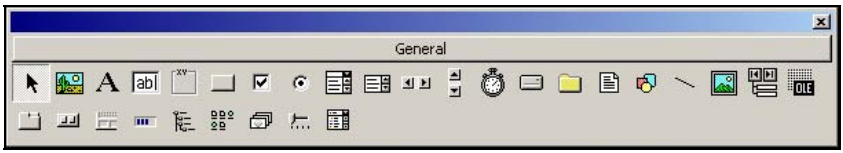


Рис. 4.3. Палитра компонентов после подключения библиотек

Компонент *CommonDialog*

Компонент `ComonDialog` (рис. 4.4) представляет собой стандартное диалоговое окно **Открыть**, **Сохранить**, **Цвет**, **Шрифт**, **Печать** или **Справка**. Чтобы компонент `CommonDialog` стал доступен, надо подключить библиотеку компонентов Microsoft Common Dialog 6.0 (comdlg32.ocx). Свойства компонента приведены в табл. 4.1. Тип окна определяет метод, обеспечивающий отображение диалога (табл. 4.2).

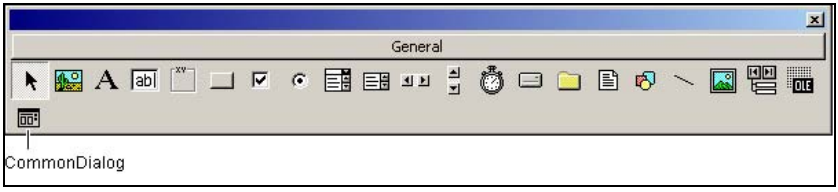


Рис. 4.4. Компонент `CommonDialog`

Таблица 4.1. *Свойства компонента* `CommonDialog`

Свойство	Описание
`DialogTitle`	Заголовок окна
`FileName`	Полное (включая путь) имя файла, выбранного в диалоге **Открыть**. Если диалог был завершен нажатием кнопки **Cancel**, значение свойства — пустая строка

Таблица 4.1 *(окончание)*

Свойство	Описание
`FileTitle`	Имя файла, выбранного в диалоге **Открыть**. Если диалог был завершен нажатием кнопки **Cancel**, значение свойства — пустая строка
`Filter`	Задает фильтр, который используется для формирования списка файлов, отображаемых в диалогах **Открыть** и **Сохранить**. Если фильтр не задан, то отображаются все файлы каталога. Можно задать несколько фильтров (в этом случае пользователь сможет выбрать нужный фильтр в списке **Тип файлов**)
`InitDir`	Каталог, содержимое которого отображается в окне
`Flags`	Флаги спецификации для диалоговых окон
`DefaultExt`	Расширение, которое будет добавлено к имени файла в случае, если пользователь не укажет расширение явно (в диалоге **Сохранить**)

Таблица. 4.2. *Методы отображения объекта* `CommonDialog`

Метод	Диалог
`ShowOpen`	**Открыть**
`ShowSave`	**Сохранить**
`ShowColor`	**Цвет**
`ShowFont`	**Выбор шрифта**
`ShowPrinter`	**Печать**
`ShowHelp`	**Справка**

Форма программы **Просмотр иллюстраций** приведена на рис. 4.5, она демонстрирует использование компонента `CommonDialog` для отображения диалога **Открыть**. Диалог служит для выбора ката-

лога, в котором находятся файлы иллюстраций. Значения свойств компонента `CommonDialog1` приведены в табл. 4.3, текст программы — в листинге 4.1. Следует обратить внимание: размер компонента `Image1` подобран таким образом, что иллюстрации размером 1600×1200 отображаются без искажения. Чтобы отображались без искажения иллюстрации другого размера, надо выполнить их масштабирование (внести соответствующие изменения в процедуру `ShowPicture`).

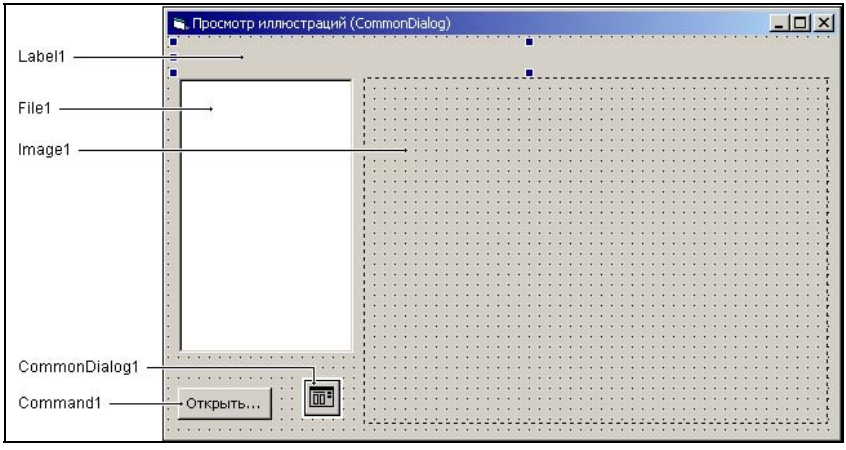


Рис. 4.5. Форма программы Просмотр иллюстраций

Таблица 4.3. *Значения свойств компонента* `CommonDialog1`

Свойство	Значение
`DialogTitle`	Открыть
`InitDir`	d:\
`Filter`	Фото (*.jpg;*.gif) | *.jpg;*.gif |
	Картинки (*.bmp;*.gif;*.jpg) |*.bmp;*.gif;*.jpg |
	Все (*.*) | *.*

Листинг 4.1. Просмотр иллюстраций (компонент CommonDialog)

```
Const cdlOFNFileMustExist = &H1000
Const cdlOFNHideReadOnly = &H4

Dim Path As String

' начало работы программы
Private Sub Form_Load()
    CommonDialog1.Flags = cdlOFNFileMustExist Or_
                          cdlOFNFileMustExist
    If File1.ListCount <> 0 Then
        ' в текущем каталоге есть jpg-файлы
        Path = CurDir + "\"
        Label1.Caption = Path + File1.List(0)
        File1.ListIndex = 0
    End If
End Sub

' щелчок на кнопке Открыть (выбор файла)
Private Sub Command1_Click()

    CommonDialog1.FileName = ""
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName = "" Then
        ' пользователь сделал щелчок на кнопке Отмена
        Exit Sub
    End If

    ' пользователь выбрал файл

    Path = Mid(CommonDialog1.FileName, 1,_
Len(CommonDialog1.FileName) - Len(CommonDialog1.FileTitle))
```

```
'   MsgBox "File: " & CommonDialog1.FileName & vbCr & _
'          "File Title: " & CommonDialog1.FileTitle & vbCr & _
'          "Path: " & path

    ' отобразить список графических файлов в поле компонента
    ' File1 (FileList)
      File1.Pattern = "*." & Right(CommonDialog1.FileName, 3)
      File1.Path = Path

      ' найти выбранный файл в списке File1.List
      Dim i As Integer
      i = 0
      Do While File1.List(i) <> CommonDialog1.FileTitle
        i = i + 1
      Loop
      File1.ListIndex = i

End Sub

' выбор другого элемента списка в результате
' щелчка на имени файла или нажатия клавиши
' перемещения курсора
Private Sub File1_Click()
    Label1.Caption = Path & File1.List(File1.ListIndex)
    ShowPicture (Path & File1.List(File1.ListIndex))
End Sub

' отобразить иллюстрацию в поле компонента Image1
Sub ShowPicture(FileName As String)
    ' размер компонента Image установлен пропорционально
    ' 1600x1200, поэтому фотографии этого размера
    ' отображаются без искажения.
    ' Чтобы иллюстрации другого размера отображались без
```

```
    ' искажения, надо выполнить масштабирование.
    Image1.Picture = LoadPicture(FileName)
End Sub
```

Компонент *StatusBar*

Компонент `StatusBar` представляет собой полосу вывода служебной информации, которая, как правило, находится в нижней части окна и разделена на несколько областей (панелей). В каждой из этих областей отображается определенная информация (например, наименование выбранной в данный момент команды меню, состояние клавиатуры, время). Чтобы компонент `StatusBar` стал доступен, надо подключить библиотеку компонентов Microsoft Windows Common Controls 6.0 SP4 (mscomctl.ocx). Свойства компонента `StatusBar` приведены в табл. 4.4.

Таблица 4.4. *Свойства компонента* `StatusBar`

Свойство	Описание
`Style`	Стиль (вид) области состояния. Панель может быть обычной (0 — `sbrNormal`) или "простой" (1 — `sbrSimple`). Обычная панель разделена на области, простая представляет собой одну-единственную область.
`SimpleText`	Текст, который отображается на панели состояния, если панель простая (значение свойства `Style` равно `sbrSimple`)

Чтобы полоса состояния была разделена на области, в нее надо добавить соответствующее количество *панелей*. Для этого надо раскрыть окно свойств компонента `StatusBar` (из контекстного меню компонента выбрать команду **Properties**) и на вкладке **Panels** щелкнуть на кнопке **InsertPanel** (рис. 4.6) столько раз, на сколько областей должна быть разделена строка состояния. После этого можно настроить каждую отдельную панель. Свойства панели компонента `StatusBar` приведены в табл. 4.5.

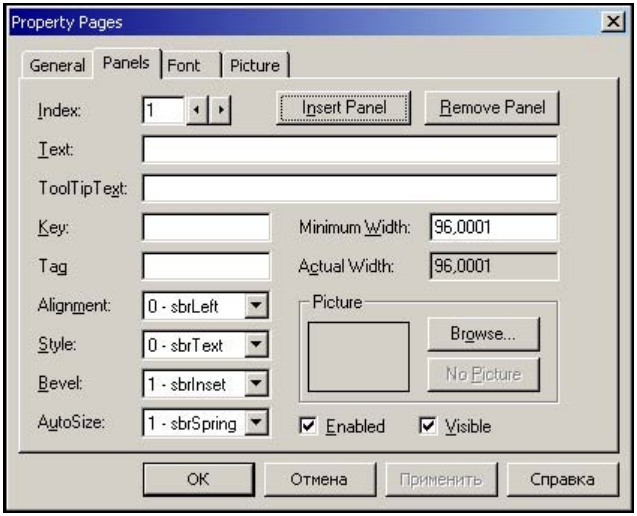


Рис. 4.6. Настройка панелей компонента `StatusBar`

Таблица 4.5. *Свойства панели (объекта* `StatusBarPanel`*)*

Свойство	Описание
`Style`	Тип информации, отображаемый на панели. Например: • текст (0 — `sbrText`); • время (5 — `sbrTime`); • дата (6 — `sbrDate`)ж • состояние клавиатуры (1 — `sbrCaps`, 2 — `sbrNum`, 3 — `sbrIns`)
Text	Текст, отображаемый на панели (если значение свойства `Style` равно `sbrText`)
`Alignment`	Размещение текста на панели: • прижат к левой границе (0 — `sbrLeft`); • прижат к правой границе (2 — `sbrRight`); • расположен по центру (1 — `sbrCenter`)

Таблица 4.5 *(окончание)*

Свойство	Описание
Picture	Картинка, отображаемая на панели
AutoSize	Признак автоматического изменения размера панели.
	Если значение свойства равно 2 (sbrContents), то ширина панели зависит от ее содержания (длины текста).
	Если значение свойства равно 1 (sbrSpring), то ширина панели устанавливается такой, чтобы находящаяся справа другая панель была прижата к правой границе окна.
	Если справа панели нет, то ширина устанавливается такой, чтобы правая граница панели была прижата к правой границе формы

Следующая программа — Компонент StatusBar (ее форма приведена на рис. 4.7, а значения свойств компонента StatusBar — в табл. 4.6). На панелях полосы состояния отражается количество символов, которое пользователь ввел в поле редактирования, и текущее время. Вывод информации о количестве введенных символов выполняет процедура обработки события Change компонента TextBox (листинг 4.2), время отображается автоматически, т. к. значение свойства Style второй панели равно sbrTime.

Таблица 4.6. *Значения свойств компонента* StatusBar

Свойство	Значение
Name	StatusBar1
Style	0 — sbrNormal
StatusBar1.Panels(1).Style	0 — sbrText
StatusBar1.Panels(1).Autosize	1 — sbrSpring
StatusBar1.Panels(2).Style	5 — sbrTime
StatusBar1.Panels(2).Autosize	0 — sbrNone
StatusBar1.Panels(2).Alignment	1 — sbrCenter

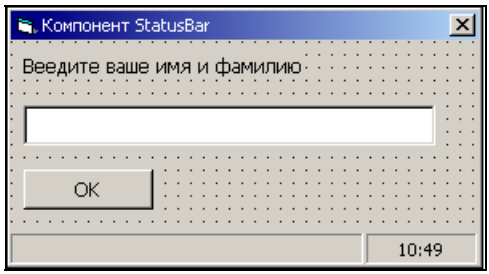


Рис. 4.7. Форма программы Компонент StatusBar

Листинг 4.2. Компонент `StatusBar`

```
' нажатие клавиши в поле редактирования
Private Sub Text1_Change()
    Form1.StatusBar1.Panels(1).Text = Len(Text1.Text)
    If Len(Text1.Text) = 0 Then
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If
End Sub
```

Компонент *ProgressBar*

Компонент `ProgressBar` (рис. 4.8) представляет собой индикатор, который обычно используется для наглядного представления протекания процесса, например, обработки (копирования) файлов, загрузки информации из сети и т. п. Чтобы компонент был доступен, надо подключить библиотеку компонентов Microsoft Windows Common Controls 6.0 SP4 (mscomctl.ocx). Свойства компонента `ProgressBar` приведены в табл. 4.7.

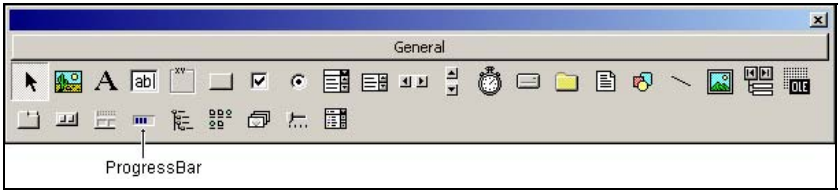


Рис. 4.8. Компонент `ProgressBar`

Таблица 4.7. *Свойства компонента* `ProgressBar`

Свойство	Описание
`Value`	Значение, которое отображается в поле компонента (на индикаторе) в виде полосы или ряда прямоугольников. Длина полосы (количество прямоугольников) пропорционально значению свойства `Value`. Свойство доступно только во время работы программы
`Min`	Минимально допустимое значение свойства `Value`
`Max`	Максимально допустимое значение свойства `Value`
`Scrolling`	Вид индикатора. Индикатор может представлять собой последовательность прямоугольников (0 – `ccScrollingStandard`) или полосу: (1 — `ccScrollingSmoth`)
`Appearance`	Вид компонента: • вровень с поверхностью формы (0 — `ccFlat`); • в стиле 3D (1 – `cc3D`)
`Border`	Граница компонента: • есть (1 — `ccFixedSingle`); • нет (0 — `ccNone`)

Форма следующей программы (Таймер) приведена на рис. 4.9, а текст в листинге 4.3 (демонстрирует использование компонента `ProgressBar`). Компонент `ProgressBar1` используется для индикации времени, прошедшего с момента запуска таймера (щелчка на кнопке **OK**), или времени, оставшегося до истечения заданного промежутка, если установлен переключатель **обратный отсчет**.

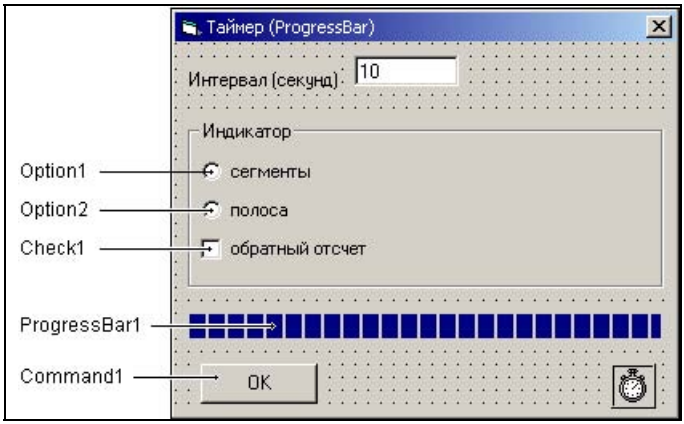


Рис. 4.9. Форма программы Таймер

Листинг 4.3. Компоненты `ProgressBar`

```
Dim delay As Integer
Dim dv As Integer
      ' приращение значения Value компонента ProgressBar:
      ' 1 - при прямом счете, -1 - при обратном
      ' щелчок на кнопке OK
Private Sub Command1_Click()

    ' вид индикатора
    If Option1.Value = True Then
```

```
        ' стандартный - сегменты
        ProgressBar1.Scrolling = ccScrollingStandard
    Else
        ' полоса
        ProgressBar1.Scrolling = ccScrollingSmooth
    End If

    ' таймер генерирует сигнал с периодом 100 мс, а в поле
    ' редактирования пользователь вводит секунды,
    ProgressBar1.Max = Text1.Text * 10

    If Check1.Value = vbChecked Then
        ' обратный отсчет
        ProgressBar1.Value = ProgressBar1.Max
        dv = -1
    Else
        ' прямой счет
        ProgressBar1.Value = 0
        dv = 1
    End If

    Command1.Enabled = False
    Timer1.Enabled = True

End Sub

' сигнал от таймера
Private Sub Timer1_Timer()
    ProgressBar1.Value = ProgressBar1.Value + dv
    If ProgressBar1.Value = ProgressBar1.Max Or _
            ProgressBar1.Value = ProgressBar1.Min Then
        Timer1.Enabled = False
        ProgressBar1.Value = 0
```

```
        Command1.Enabled = True
    End If
End Sub

' нажатие клавиши в поле Text1
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If Not ((KeyAscii >= 48 And KeyAscii <= 57) Or
            KeyAscii = 8) Then
        KeyAscii = 0
    End If
End Sub
```

Компонент *UpDown*

Компонент `UpDown` (рис. 4.10) представляет собой две кнопки, используя которые можно изменить (увеличить или уменьшить) значение внутренней переменной-счетчика. Компонент `UpDown` можно связать с другим компонентом, например, `Label` или `TextBox`, и использовать его в качестве индикатора значения переменной-счетчика. Свойства компонента `UpDown` приведены в табл. 4.8.

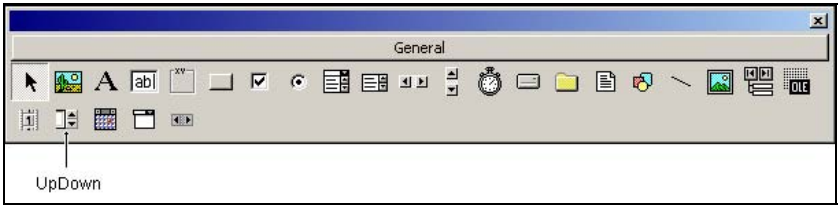


Рис. 4.10. Компонент `UpDown`

Чтобы компонент `UpDown` был доступен, надо подключить библиотеку компонентов Microsoft Windows Common Controls-2 6.0 (MSCOMCT2.OCX).

Таблица 4.8. *Свойства компонента* `UpDown`

Свойство	Описание
`Value`	Счетчик. Значение счетчика изменяется в результате щелчка на кнопке **Up** (Увеличение) или **Down** (Уменьшение)
`Increment`	Величина изменения значения переменной-счетчика
`Max`	Верхняя граница диапазона изменения значений переменной-счетчика `Value`
`Min`	Нижняя граница диапазона изменения значений переменной-счетчика `Value`
`BuddyControl`	Компонент, который используется в качестве индикатора значения переменной-счетчика (например, `Label` или `TextBox`)
`BuddyProperty`	Свойство компонента, указанного в `BuddyControl`, обеспечивающее индикацию значения переменной-счетчика (`Caption`, если в качестве индикатора используется компонент `Label`)
`AutoBuddy`	Автоматическое определение свойства компонента-индикатора, используемого для индикации состояния переменной-счетчика. Если в качестве индикатора используется компонент `Label`, то автоматически в качестве значения свойства `BuddyProperty` устанавливается `Caption`
`SyncBuddy`	Синхронизация (значение — `True`) изменений значения `Value` и значения свойства компонента-индикатора
`Orientation`	Ориентация кнопок (стрелок на кнопках) компонента: • 0 (`OrientationVertical`) — по вертикали (вверх, вниз); • 1 (`OrientationHorizontal`) — по горизонтали (влево, вправо)

Таблица 4.8 *(окончание)*

Свойство	Описание
`Wrap`	Если значение свойства равно `False`, то при достижении максимального значения переменной `Value`, ее значение не изменяется при последующих нажатиях кнопки **Up**. Аналогично для кнопки **Down**. Если значение свойства равно `True`, то при аналогичных действиях максимальное значение переменной `Value` изменяется на минимальное и наоборот
`Enabled`	Доступность (значение свойства — `True`) или недоступность компонента (значение свойства — `False`)
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

В качестве примера использования компонента `UpDown` приведена программа Таймер. Форма программы представлена на рис. 4.11, значения свойств компонентов `UpDown` — в табл. 4.9.

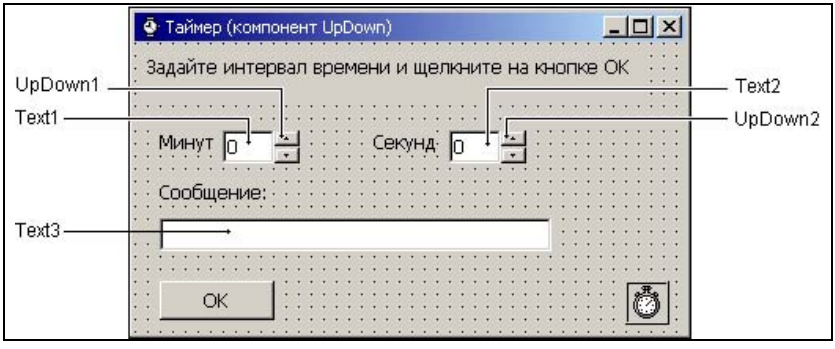


Рис. 4.11. Форма программы Таймер

Таблица 4.9. *Значения свойств компонентов* `UpDown`

Свойство	Значение
UpDown1.Min	0
UpDown1.Max	59
UpDown1.Wrap	True
UpDown1.BuddyControl	Text1
UpDown1.BuddyProperty	Text
UpDown1.SyncBuddy	True
UpDown2.Min	0
UpDown2.Max	59
UpDown2.Wrap	True
UpDown2.BuddyControl	Text2
UpDown2.BuddyProperty	Text
UpDown2.SyncBuddy	True

Назначение программы Таймер — вывести информационное сообщение и звуковой сигнал через заданный пользователем интервал времени. Значение интервала можно ввести непосредственно в поля редактирования или задать при помощи кнопок компонентов `UpDown`. При использовании компонента `UpDown` значение переменной-счетчика изменяется "по кругу". Это значит, что если значение свойства `Value` совпадает со значением нижней границы диапазона изменения переменной-счетчика (свойство `Min`), и пользователь сделает щелчок на кнопке **Уменьшить**, то значение свойства `Value` станет равным верхней границе диапазона изменения переменной-счетчика (свойство `Max`). Аналогичным образом изменяется значение свойства `Value` при достижении верхней границы диапазона. Текст программы Таймер приведен в листинге 4.4.

Листинг 4.4. Таймер (компонент UpDown)

```
Dim Delay As Integer ' интервал времени (секунд)

Private Sub Form_Load()
    Timer1.Enabled = False
    Timer1.Interval = 1000 ' сигнал от таймера каждую секунду
    Text1.Text = 45
    Text3.Text = "Отдохни, попей кофейку!"
End Sub

' нажатие клавиши в поле Tetx1 (количество минут)
Private Sub Text1_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 8  ' <Backspace>
        Case 48 To 57 ' цифры
            If Len(Text1.Text) = 2 Then KeyAscii = 0
        Case 13
            If Val(Text1.Text) > 59 Then Text1.Text = "59"
            Text2.SetFocus
        Case Else
            KeyAscii = 0
    End Select
End Sub

' нажатие клавиши в поле Tetx2 (количество секунд)
Private Sub Text2_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 8  ' <Backspace>
        Case 48 To 57 ' цифры
            If Len(Text2.Text) = 2 Then KeyAscii = 0
        Case 13
            If Val(Text2.Text) > 59 Then Text2.Text = "59"
```

```
            Text3.SetFocus
        Case Else
            KeyAscii = 0
    End Select
End Sub

' нажатие клавиши в поле Сообщение
Private Sub Text3_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then Command1.SetFocus
End Sub

' щелчок на кнопке OK
Private Sub Command1_Click()
    Form1.WindowState = vbMinimized ' свернуть окно программы
    Delay = Val(Text1.Text) * 60 + Val(Text2.Text) ' интервал
    Timer1.Enabled = True ' пуск таймера
End Sub

' сигнал от таймера
Private Sub Timer1_Timer()
    If Delay = 0 Then
        ' время прошло
        Timer1.Enabled = False
              ' остановить таймер и вывести сообщение
        MsgBox Text3.Text, vbOKOnly + vbInformation, "Таймер"
        Form1.WindowState = vbNormal ' развернуть окно
     Else
        Delay = Delay - 1
     End If
End Sub
```

Компонент *Menu*

В Visual Basic нет компонента, который представляет собой меню. Тем не менее добавить меню в форму не сложно.

Создание и настройку меню обеспечивает утилита Menu Editor (рис. 4.12). Чтобы ее запустить, надо в меню **Tools** (или в контекстном меню формы) выбрать соответствующую команду.

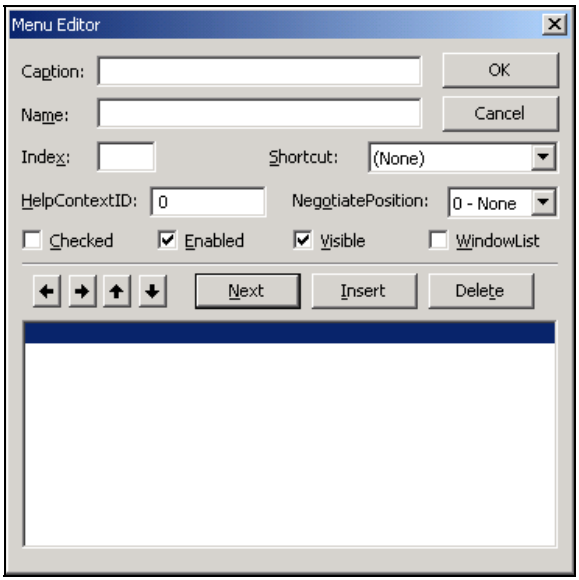


Рис. 4.12. Утилита Menu Editor, обеспечивающая создание меню

Чтобы создать меню, надо в поля **Caption** и **Name** окна **Menu Editor** ввести соответственно название и идентификатор меню (например, `Файл` и `File`) и нажать клавишу <Enter>. В результате в форму будет добавлено меню. После этого надо ввести команды меню. Команды вводятся аналогичным образом (название — в поле **Caption**, идентификатор — в поле **Name**). Но перед тем как ввести название первой команды, надо сделать щелчок на кнопке **Понизить уровень** (на ней нарисована направленная вправо стрелка). После того как будут введены все команды те-

кущего меню, и перед тем, как ввести название следующего меню, надо сделать щелчок на кнопке **Повысить уровень** (стрелка влево). В качестве примера на рис. 4.13 приведено окно **Menu Editor** в конце работы по созданию меню для программы MEdit.

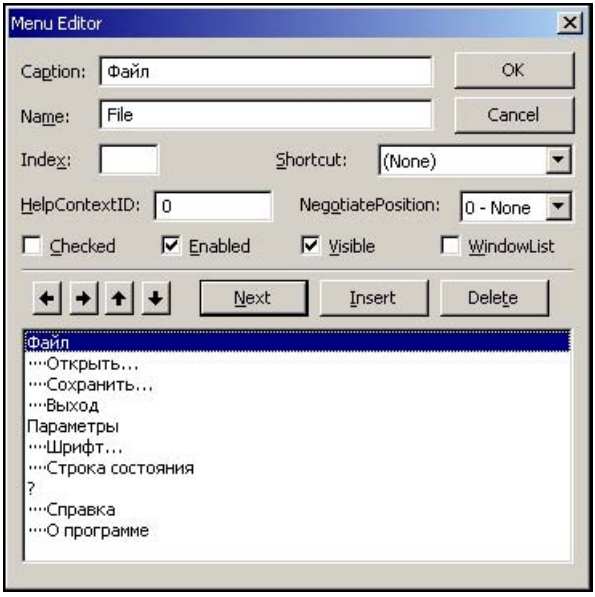


Рис. 4.13. Диалоговое окно **Menu Editor**

Следует обратить внимание на то, что каждое меню и каждая команда меню представляют собой объекты типа `Menu`. Свойства объекта типа `Menu` приведены в табл. 4.10. Изменить характеристики меню или команды меню можно как в окне **Menu Editor**, так и обычным способом — в окне свойств.

Таблица 4.10. *Свойства объекта* `Menu`

Свойство	Описание
`Caption`	Название меню или команды
`Name`	Идентификатор элемента меню

Таблица 4.10 *(окончание)*

Свойство	Описание
`Enabled`	Доступен. Если значение свойства равно `False`, то элемент меню не доступен (в результате щелчка на элементе меню событие `Click` не происходит, название элемента меню отображается инверсным цветом по отношению к доступному пункту меню)
`Checked`	Признак того, что элемент меню выбран. Если значение свойства равно `True`, то элемент помечается галочкой. Свойство `Checked` обычно используется для тех элементов меню, которые используются для отображения значений параметров
`Shortcut`	Свойство определяет функциональную клавишу (или комбинацию клавиш), нажатие которой активизирует выполнение команды

Объект `Menu` (меню или команда меню) способен воспринимать событие `Click`, которое возникает в результате щелчка на элементе меню или в результате нажатия клавиши <Enter> (если элемент выбран). Чтобы создать процедуру обработки события `Click` для элемента меню, надо в окне сделать щелчок на нужном элементе меню.

Следующая программа — Medit (ее форма приведена на рис. 4.14) — демонстрирует использование меню. Характеристики меню — в табл. 4.11, текст программы — в листинге 4.5.

Таблица 4.11. *Характеристики меню программы MEdit*

Caption	**Name**
Файл	File
Открыть	FileOpen
Сохранить	FileSave
Выход	FileExit
Параметры	Parameters

Таблица 4.11 *(окончание)*

Caption	Name
Шрифт	ParametersFont
Строка состояния	PatametersStatusBar
?	Help
Справка	HelpHelp
О программе	HelpAbout

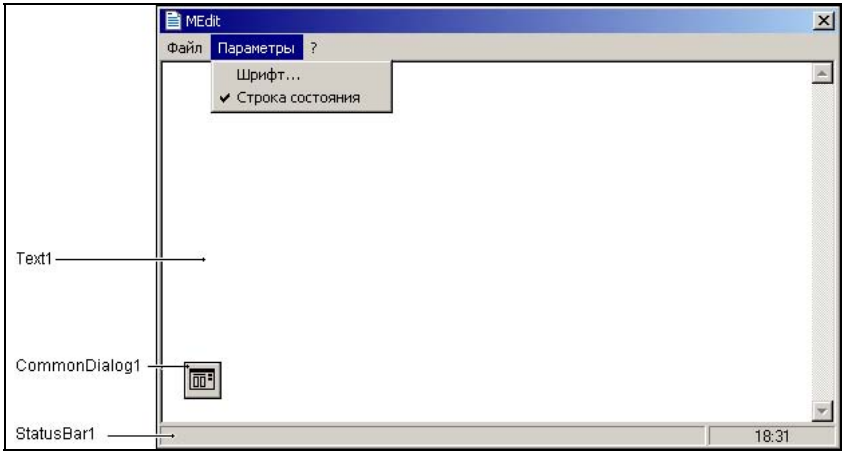


Рис. 4.14. Форма программы MEdit

Листинг 4.5. MEdit (использование меню)

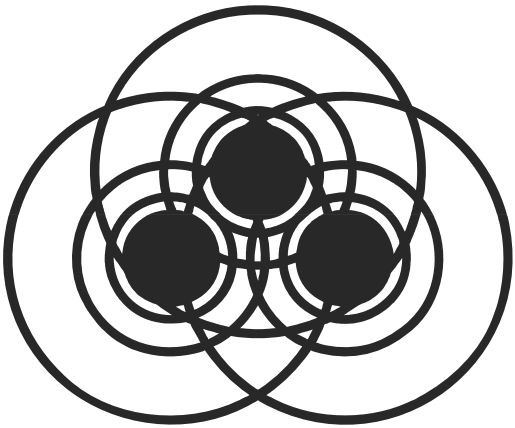
```
Option Explicit
Dim fName As String      ' имя файла
Dim fChanged As Boolean ' True - текст изменен

' выбор в меню "Файл" команды "Открыть"
Private Sub FileOpen_Click()
    Dim fn As Integer
```

```
    Dim buf As String
    CommonDialog1.FileName = "*.txt"
    CommonDialog1.ShowOpen

    If CommonDialog1.FileName <> "*.txt" Then
        ' пользователь выбрал файл
        fName = CommonDialog1.FileName
        fn = FreeFile
        ' прочитать текст из файла
        Open fName For Input As fn
        Input #fn, buf
        Close #fn

        Text1.Text = buf
        Text1.SelStart = Len(Text1.Text)
                        ' курсор в конец текста
        frmMEdit.Caption = "MEdit - " & fName
        fChanged = False ' изменений нет

    End If
End Sub

' команда Файл/Сохранить
Private Sub FileSave_Click()
    SaveFile
End Sub

' команда Файл/Выход
Private Sub FileExit_Click()
    Dim r As Integer

    If fChanged Then
        r = MsgBox("В текст изменен. Сохранить измененный_
```

```
        текст?", vbExclamation + vbYesNo, "MEdit")
        If r = vbYes Then
            SaveFile
        End If
    End If
    Unload frmMEdit ' завершить работу программы
End Sub

' команда Формат/Шрифт
Private Sub ParametersFont_Click()
    CommonDialog1.Flags = cdlCFScreenFonts
    CommonDialog1.ShowFont
    If CommonDialog1.FontName <> "" Then
        ' пользователь выбрал шрифт
        Text1.FontName = CommonDialog1.FontName
        Text1.FontSize = CommonDialog1.FontSize
        Text1.FontBold = CommonDialog1.FontBold
        Text1.FontItalic = CommonDialog1.FontItalic
    End If

End Sub

' команда О программе
Private Sub HelpAbout_Click()
    frmAbout.Show vbModal
End Sub

Private Sub Form_Load()
    Text1.Width = frmMEdit.ScaleWidth
    Text1.Height = frmMEdit.ScaleHeight - StatusBar1.Height
End Sub
```

```
' команда Строка состояния
Private Sub ParametersStatusBar_Click()
    If ParametersStatusBar.Checked Then
        ' команда помечена галочкой и
        ' строка состояния отображается
         StatusBar1.Visible = False ' скрыть строку состояния
        Text1.Height = frmMEdit.ScaleHeight
        ParametersStatusBar.Checked = False ' убрать галочку
    Else
      Text1.Height = frmMEdit.ScaleHeight - StatusBar1.Height
      StatusBar1.Visible = True
      ParametersStatusBar.Checked = True
    End If
End Sub

' событие Change - изменилось содержимое компонента TextBox
Private Sub Text1_Change()
    fChanged = True

End Sub

Sub SaveFile()
    Dim fn As Integer
    Dim i As Integer

    If fName = "" Then
        ' получить у пользователя имя файла
        CommonDialog1.FileName = ""
        CommonDialog1.DefaultExt = "txt"
        CommonDialog1.Filter = "*.txt"
        CommonDialog1.ShowSave
        If CommonDialog1.FileName <> "" Then
```

```
            fName = CommonDialog1.FileName
        Else
            Exit Sub
        End If
    End If

    ' записать текст в файл
    fn = FreeFile
    Open fName For Output As fn
    Write #fn, Text1.Text
    frmMEdit.Caption = "MEdit - " & fName
    Close #fn
    fChanged = False
End Sub
```

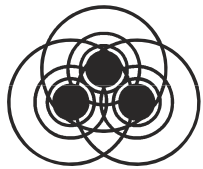


ЧАСТЬ II

ПРАКТИКУМ ПРОГРАММИРОВАНИЯ

Вторая часть книги посвящена практике программирования. В ней рассматриваются вопросы программирования графики, мультимедиа, баз данных. Уделено внимание созданию справочной системы и программы установки.

Глава 5



Графика

Visual Basic позволяет создавать программы, которые работают с графикой. В этой главе рассказывается о том, что надо сделать, чтобы на поверхности формы появилась фотография, картинка, сформированная из графических примитивов (линий, прямоугольников, окружностей, точек), или иллюстрация, созданная в графическом редакторе. Так же вы познакомитесь с принципами анимации, т. е. узнаете, как можно "оживить" картинку.

Программа может вывести графику на поверхность формы или компонента `PictureBox`. Чтобы во время работы программы на поверхности объекта появилась, например, иллюстрация или линия, необходимо вызвать соответствующий метод. Так, в результате выполнения инструкции

```
Form1.Line (10,10) - (50,10)
```

на поверхности формы появится горизонтальная линия.

Графику на поверхности объекта должна формировать процедура обработки события `Paint`. Это объясняется необходимостью обновления графики при каждом появлении объекта на экране, а событие `Paint` как раз и возникает всякий раз, когда объект появляется на экране (в т. ч. и после того, как пользователь сдвинет другое окно, которое частично или полностью перекрывает окно программы).

Графическая поверхность

Методы вычерчивания (вывода) графических примитивов рассматривают поверхность формы (компонента `PictureBox`) как

холст, на котором они могут *рисовать* путем изменения цвета отдельных точек — пикселов. Положение пиксела на графической поверхности характеризует горизонтальная (*x*) и вертикальная (*y*) координаты. Координаты отсчитываются от левого верхнего угла. Возрастают координаты слева направо и сверху вниз. Точка, находящаяся в левом верхнем углу поверхности, имеет координаты (0, 0), а в правом нижнем — (`ScaleWidth-1`, `ScaleHeight-1`), где `ScaleWidth` и `ScaleHeight` — соответственно ширина и высота рабочей области формы (рис. 5.1).

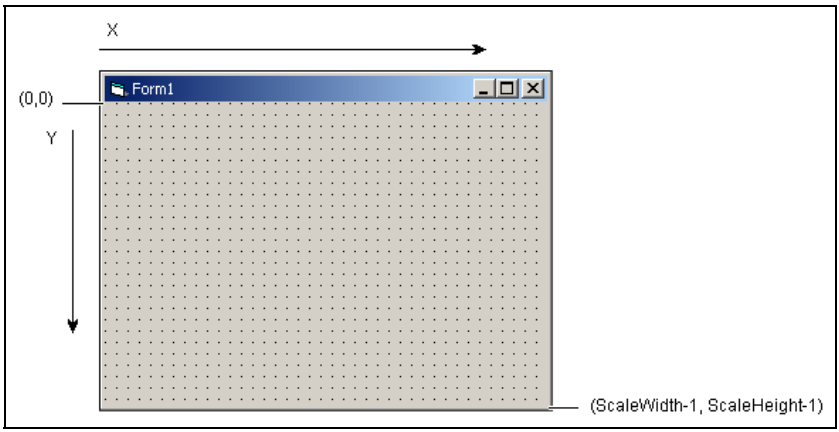


Рис. 5.1. Координаты точек графической поверхности

Следует обратить внимание на то, что размер графической поверхности и, следовательно, координаты могут измеряться в твипах (эта единица используется по умолчанию), пикселах, сантиметрах и др. При программировании графики наиболее удобной единицей измерения является пиксел (это позволяет получить прямой доступ к каждой точке графической поверхности). Поэтому свойству `ScaleMode` формы (компонента `PictureBox`) надо присвоить значение `Pixel` (во время создания формы) или `vbPixels` (в процедуре обработки события `Load`).

Координаты точек графической поверхности можно отсчитывать от левого верхнего угла или от *указателя графического вывода*.

Указатель графического вывода — это невидимый маркер, который находится в той точке графической поверхности, где была завершена последняя операция вывода графики. Например, после выполнения инструкции

```
Form1.Line (10,10)-(50,10),vbGreen
```

эта инструкция рисует на поверхности формы зеленую горизонтальную линию, указатель графического вывода располагается в точке (50, 10). В начале работы программы указатель находился в точке (0, 0). Информация о положении указателя графического вывода отражается в свойствах `CurrentX` и `CurrentY`. Программа может переместить указатель графического вывода в нужную точку, для этого надо присвоить соответствующие значения свойствам `CurrentX` и `CuttentY`. Например, в результате выполнения следующих инструкций:

```
Form1.CurrentX = ScaleWidth / 2 - TextWidth("Microsoft Visu-
al_ Basic") / 2
Form1.CurrentY = ScaleHeight / 2
Print "Microsoft Visual Basic"
```

на поверхности формы (в ее центре) появится текст Microsoft Visual Basic.

Графические примитивы

Любая картинка, чертеж, схема представляет собой совокупность графических *примитивов*: точек, линий, окружностей, дуг, текста и др.

Вычерчивание (вывод) графических примитивов на графической поверхности выполняют соответствующие методы (табл. 5.1).

Таблица 5.1. *Методы вычерчивания (вывода) графических примитивов*

Метод	Действие
`Line`	В зависимости от значения параметров рисует линию, прямоугольник или контур прямоугольника

Таблица 5.1 *(окончание)*

Метод	Действие
`Circle`	В зависимости от значения параметров рисует окружность, эллипс, дугу, круг или сектор
`PSet`	Рисует точку
`Print`	Выводит текст

Точка

Точку на графической поверхности рисует метод `PSet`.

Инструкция вызова метода `PSet` в общем виде выглядит так:

```
Объект.Pset Step(x,y),Color
```

Параметр `Объект` задает объект, на поверхности которого надо нарисовать точку. Если точку надо нарисовать на поверхности формы, то этот параметр можно не указывать.

Параметры `x` и `y` задают координаты точки графической поверхности, цвет которой надо изменить.

Параметр `Color` задает цвет точки. В качестве параметра `Color` можно использовать одну из именованных констант (табл. 5.2). Также в качестве параметра `Color` можно использовать значение функции `RGB`, которая возвращает код цвета, заданного красной, зеленой и синей составляющими (как известно, любой цвет можно получить путем смешивания в разных пропорциях красной, зеленой и синей красок). У функции `RGB` три параметра: первый задает долю красной (`red`), второй — зеленой (`green`), третий — синей (`blue`) составляющей. Значение каждого из параметров должно находиться в диапазоне от 0 до 255. Например, значением функции `RGB` (205, 127, 50) является код "золотого" цвета (значения параметров функции `Rgb` для других цветов можно найти в *главе 11*). Параметр `Color` не является обязательным. Если он не указан, то цвет закраски точки определяет значение свойства `ForeColor` графической поверхности, на которой рисует метод.

Таблица. 5.2. *Константы, используемые при задании цвета*

Константа	Цвет
vbBlack	Черный
vbRed	Красный
vbGreen	Зеленый
vbYellow	Желтый
vbBlue	Синий
vbMagenta	Пурпурный
vbCyan	Бирюзовый
vbWhite	Белый

Ключевое слово **Step** можно не указывать. В этом случае параметры x1 и y1 задают абсолютные координаты точки. Если слово **Step** указано, то координаты точки отсчитываются от текущего положения указателя графического вывода.

Размер (диаметр) точки, которую рисует метод PSet, определяет текущее значение свойства DrawWidth поверхности, на которой рисует метод.

Линия

Вычерчивание прямой линии выполняет метод Line.

Инструкция вызова метода Line, обеспечивающая вычерчивание линии, в общем виде выглядит так:

```
Объект.Line Step(x1,y,)-Step(x2,y2),Color
```

Параметр Объект задает объект, на поверхности которого надо нарисовать линию. Если линию надо нарисовать на поверхности формы, то этот параметр можно не указывать.

Параметры x1 и y1 задают координаты точки начала линии, а параметры x2 и y2 — координаты точки конца.

Параметр Color задает цвет линии. В качестве параметра Color можно использовать одну из именованных констант (см. табл. 5.2)

или значение функции `Rgb`. Параметр `Color` является необязательным. Если он не указан, то цвет линии определяет значение свойства `ForeColor` графической поверхности, на которой рисует метод.

Ключевые слова **`Step`** можно не указывать. В этом случае параметры `x1`, `y1` и `x2`, `y2` задают абсолютные координаты концов линии. Если слово **`Step`** указано перед параметрами `x1`, `y1`, то координаты точки начала линии отсчитываются от указателя графического вывода. Если слово **`Step`** указано перед параметрами `x2`, `y2`, то координаты точки конца линии отсчитываются от точки ее начала.

Толщину и вид (стиль) линии определяют соответственно свойства `DrawWith` и `DrawStyle` графической поверхности, на которой рисует метод `Line`. В табл. 5.3 приведены константы, используя которые можно задать вид линии. Следует обратить внимание: линия, толщина которой больше чем 1, может быть только сплошной (нарисовать пунктирную линию толщиной больше, чем 1 пиксел, нельзя).

Таблица 5.3. *Константы, используемые для задания вида линии*

Константа	Вид (стиль) линии
`VbSolid`	Сплошная
`VbDash`	Штриховая (длинные штрихи)
`VbDot`	Пунктирная (короткие штрихи)
`VbDashDot`	Штрих-пунктирная
`VbDashDotDot`	Штрих, два пунктира

Окно программы Line Demo, которая демонстрирует использование функции `Line`, приведено на рис. 5.2, а ее текст — в листинге 5.1.

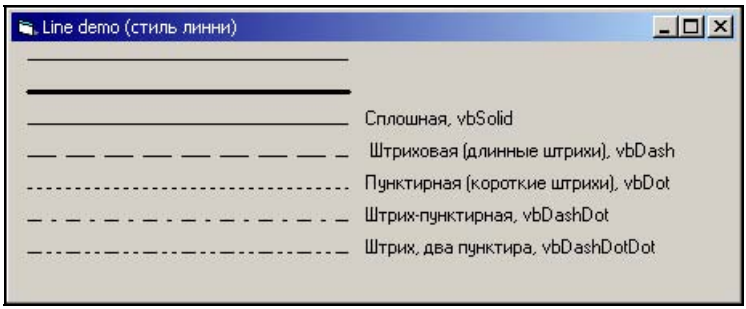


Рис. 5.2. Окно программы Line Demo (линии различного стиля)

Листинг 5.1. Line demo (линии)

```
Option Explicit

Dim st(0 To 5) As String ' названия стилей

Private Sub Form_Load()

    ScaleMode = vbPixels

    ' Индекс элемента массива соответствует значению
    ' константы. Например, значение vbDot равно двум.
    st(0) = "Сплошная, vbSolid"
    st(1) = "Штриховая (длинные штрихи), vbDash"
    st(2) = "Пунктирная (короткие штрихи), vbDot"
    st(3) = "Штрих-пунктирная, vbDashDot"
    st(4) = "Штрих, два пунктира, vbDashDotDot"

End Sub

Private Sub Form_Paint()
    Dim y As Integer ' вертикальная координата линии
    Dim i As Integer
```

```
    ' горизонтальная линия
    DrawStyle = vbSolid
    Line (10, 10)-(210, 10)

    DrawWidth = 3
    ' ForeColor = vbBlue
    ' гор. линия из точки (10,30)
    Line (10, 30)-Step(200, 0)

    ' линии различных стилей
    DrawWidth = 1
    ForeColor = vbBlack
    y = 50
    For i = 0 To 4
        DrawStyle = i
        Line (10, y)-Step(200, 0)

        ' сместить указатель вывода вправо и вверх
        CurrentX = CurrentX + 10
        CurrentY = CurrentY - 10

        Print st(i)
        y = y + 20
    Next i
End Sub
```

Прямоугольник

Метод `Line` позволяет нарисовать не только линию, но и прямоугольник (рис. 5.3).

Инструкция вызова метода `Line`, обеспечивающая рисование прямоугольника, в общем виде выглядит так:

```
Объект.Line Step(x1,y1) - Step(x2,y2),Color,B
```

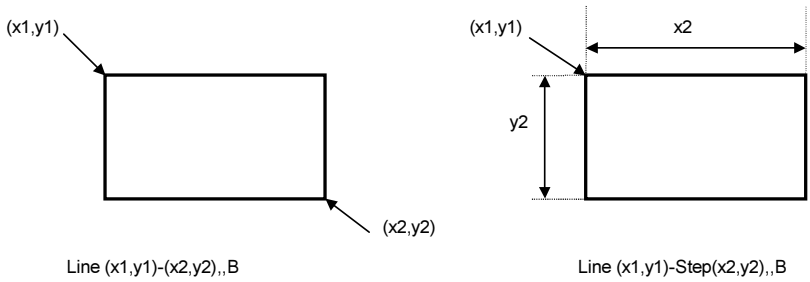


Рис. 5.3. Метод `Line`, позволяющий рисовать прямоугольник

Параметр `Объект` задает объект, на поверхности которого надо нарисовать прямоугольник. Если прямоугольник надо нарисовать на поверхности формы, то этот параметр можно не указывать.

Параметры `x1` и `y1` задают координаты левого верхнего (нижнего) угла прямоугольника, а параметры `x2`, `y2` — координаты правого нижнего (верхнего) угла.

Параметр `B` указывает на то, что метод `Line` должен нарисовать прямоугольник. Цвет границы задает параметр `Color`. В качестве этого параметра можно использовать одну из именованных констант (см. табл. 5.2) или значение функции `RGB`. Параметр `Color` является необязательным. Если он не указан, то цвет границы определяет значение свойства `ForeColor` графической поверхности, на которой рисует метод.

Толщину и вид (стиль) линии границы прямоугольника определяют соответственно свойства `DrawWith` и `DrawStyle` графической поверхности, на которой рисует метод `Line`. В качестве значения свойства `DrawStyle` можно использовать одну из приведенных в табл. 5.4 констант. Если ширина границы больше чем 1 пиксел, то размер прямоугольника (по внешней границе) будет больше размера области, заданной параметрами `x1`, `y1` и `x2`, `y2`. Чтобы размер прямоугольника (по внешней границе) был равен размеру области (в том случае, когда ширина границы больше чем 1 пиксел), свойству `DrawStyle` надо присвоить значение `vbInsideSolid`. Тогда линия границы будет нарисована так, что ее внешний край будет расположен строго по границе области, заданной точками (`x1`, `y1`) и (`x2`, `y2`).

Цвет и стиль закраски внутренней области прямоугольника определяют соответственно свойства `FillColor` и `FilStyle` той графической поверхности, на которой рисует метод. По умолчанию значение свойства `FillStyle` равно `vbSFTransparent`, поэтому метод рисует только границу прямоугольника. Чтобы внутренняя область прямоугольника была закрашена цветом, заданным свойством `FillColor`, значение свойства `FillStyle` должно быть равно `vbFSSolid` (сплошная заливка). Внутренняя область прямоугольника может быть заштрихована. Константы, при помощи которых можно задать стиль закраски приведены в табл. 5.4.

Таблица 5.4. *Стили закраски области (прямоугольника)*

Константа	Способ (стиль) закраски
`vbFSTransparent`	Прозрачная. Внутренняя область прямоугольника не закрашивается
`vbFSSolid`	Обычная (сплошная) закраска
`vbHorizontalLine`	Горизонтальная штриховка
`vbVerticalLine`	Вертикальная штриховка
`vbUpwardDiagonal`	Диагональная штриховка (наклон линий влево)
`vbDownwardDiagonal`	Диагональная штриховка (наклон линий вправо)
`vbCross`	Клетка
`vbDiagonalCross`	Диагональная клетка

Ключевые слова **`Step`** можно не указывать. В этом случае параметры `x1`, `y1` и `x2`, `y2` задают абсолютные координаты углов прямоугольника.

Если слово **`Step`** указано перед параметрами `x1`, `y1`, то координаты левого верхнего (нижнего) угла прямоугольника отсчитываются от текущего положения указателя графического вывода.

Если слово **`Step`** указано перед параметрами `x2`, `y2`, то координаты правого нижнего (верхнего) угла прямоугольника отсчитываются от другого диагонального угла, т. е. фактически парамет-

ры x2, y2 задают размер прямоугольника (x2 — ширина, y2 — высота).

Если вместо параметра B указать параметр BF, то метод нарисует закрашенный прямоугольник, цвет границы и цвет закраски которого будут совпадать. Цвет прямоугольника в этом случае определяет параметр Color или (если он не указан) свойство ForeColor. Следует обратить внимание: после того, как метод Line нарисует прямоугольник, указатель графического вывода будет находиться в точке (x2, y2).

Следующая программа (Line demo (прямоугольник)) демонстрирует, как нарисовать прямоугольник при помощи метода Line. В рассматриваемой программе этот метод вызывается шесть раз с разными параметрами, в результате чего на поверхности формы появляются шесть разных прямоугольников (рис. 5.4), текст приведен в листинге 5.2.

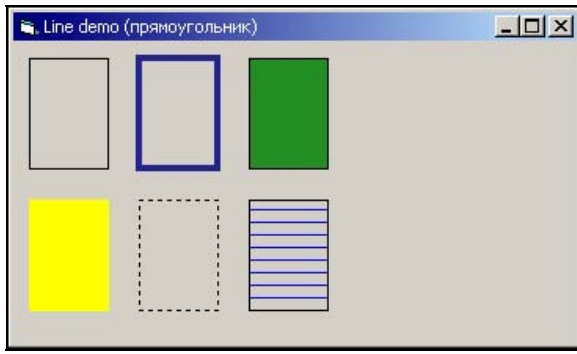


Рис. 5.4. Метод Line с параметром B рисует прямоугольник

Листинг 5.2. Line demo (прямоугольник)

```
Option Explicit

Private Sub Form_Load()
    ScaleMode = vbPixels
End Sub
```

```
Private Sub Form_Paint()

    ' прямоугольник (контур) 50x70
    ' цвет границы определяет свойство формы ForeColor
     DrawStyle = vbFSTransparent
     Line (10, 10)-Step(50, 70), , B

    ' прямоугольник (контур) 50x70
    ' цвет границы - темно-синий
    ' ширина границы - 3 пиксела
    DrawWidth = 4
    Line (80, 10)-(130, 80), RGB(35, 35, 142), B

    ' закрашенный прямоугольник 50x70
    ' цвет границы и цвет закраски не совпадают
    ' цвет закраски задает свойство формы FillColor
    FillStyle = vbFSSolid
    FillColor = RGB(35, 142, 35) ' лесной зеленый
    DrawWidth = 1
    ' после вывода предыдущего прямоугольника указатель
    ' вывода в точке (130,80)
    Line Step(20, -70)-Step(50, 70), vbBlack, B

    ' закрашенный прямоугольник 50x70
    ' без границы
    Line (10, 100)-Step(50, 70), vbYellow, BF

    ' прямоугольник (контур)
    ' граница - штрих-пунктир
    DrawStyle = vbDot
    DrawWidth = 1
    FillStyle = vbFSTransparent ' прямоугольник не
                                  закрашивать
    Line (80, 100)-Step(50, 70), , B
```

```
    ' заштрихованный прямоугольник
    DrawStyle = vbSolid
    FillStyle = vbHorizontalLine
    FillColor = vbBlue ' цвет штриховки
    Line (150, 100)-Step(50, 70), vbBlack, B

End Sub
```

Окружность и круг

Метод `Circle` в зависимости от значения параметров чертит окружность, эллипс, дугу или сектор. Инструкция вызова метода `Circle`, обеспечивающая рисование окружности или круга в общем виде выглядит так:

```
Объект.Circle Step(x,y), r, Color
```

Параметр `Объект` задает объект, на поверхности которого надо нарисовать окружность. Если окружность надо нарисовать на поверхности формы, то этот параметр можно не указывать.

Параметры `x1` и `y1` задают координаты центра окружности (круга). Если указано слово **`Step`**, то положение центра отсчитывается от текущего положения указателя графического вывода.

Параметр `r` задает радиус окружности (круга).

Параметр `Color` задает цвет окружности или границы круга. В качестве параметра `Color` можно использовать одну из именованных констант (см. табл. 5.2) или значение функции `RGB`. Параметр `Color` является необязательным. Если параметр не указан, то цвет окружности (границы круга) определяет значение свойства `ForeColor` графической поверхности, на которой рисует метод.

Толщину и вид (стиль) линии окружности (границы круга) определяют соответственно свойства `DrawWith` и `DrawStyle` графической поверхности, на которой метод рисует.

Цвет и стиль закраски внутренней области окружности определяют соответственно свойства `FillColor` и `FilStyle` той графи-

ческой поверхности, на которой рисует метод. Чтобы внутренняя область окружности была закрашена, значение свойства `FillStyle` должно быть отлично от `vbFSTransparent`.

Следующая программа — Афины 2004 (рис. 5.5). Процедура обработки события `Paint` рисует на поверхности формы символ Олимпийских игр. Текст программы, приведенный в листинге 5.3, демонстрирует использование метода `Circle`. Следует обратить внимание на то, что явно заданы координаты только первой окружности (кольца), координаты каждого следующего кольца задаются относительно предыдущего.

Рис. 5.5. Окно программы Афины 2004

Листинг 5.3. Программа Афины 2004

```
Private Sub Form_Load()
    ScaleMode = vbPixels
End Sub

Private Sub Form_Paint()
    Dim x, y As Integer
    Dim r As Integer

    x = 40
    y = 40
    r = 20
    DrawWidth = 2
```

```
    ' первый ряд колец рисуем слева направо
    Circle (x, y), r, RGB(50, 50, 205)
    ' второе кольцо сдвинуто вправо на 2.25r
    Circle Step(2.25 * r, 0), r, vbBlack
    Circle Step(2.25 * r, 0), r, vbRed

    ' второй ряд колец рисуем справа налево
    ' правое кольцо второго ряда сдвинуто вниз и влево
    ' относительно правого кольца первого ряда
    Circle Step(-1.125 * r, r), r, vbYellow
    Circle Step(-2.25 * r, 0), r, RGB(35, 142, 35)

    CurrentX = 20
    CurrentY = 100
    Print "Быстрее, выше, сильнее!"

End Sub
```

Дуга и сектор

Инструкция вызова метода `Circle`, обеспечивающая рисование дуги или сектора в общем виде, выглядит так:

```
Объект.Circle Step(x,y), r, Color, start, fin
```

Параметры `x`, `y`, `r` и `Color`, как и при рисовании окружности, определяют соответственно координату центра окружности (круга), из которой вырезана дуга (сектор), радиус окружности (круга) и цвет линии дуги (границы сектора). Толщину и стиль линии дуги или границы сектора, а также цвет и стиль заливки внутренней области сектора определяют соответственно параметры `DrawWidh`, `DrawStyle`, `FillColor` и `FillStyle` графической поверхности, на которой рисует метод.

Параметр `start` задает начальную точку дуги — точку пересечения линии окружности и прямой, проведенной из центра окружности под углом `start` относительно оси `X`.

Параметр `fin` задает конечную точку дуги (рис. 5.6). Угловые координаты измеряются в радианах и возрастают *против* часовой стрелки. Дуга вычерчивается от начальной точки к конечной также *против* часовой стрелки. Следует обратить внимание на то, что метод `Circle` рисует дугу, если значения параметров `start` и `fin` положительные, а если перед параметрами поставлен знак минус, то метод рисует сектор (рис. 5.7). Чтобы нарисовать сектор из точки, соответствующей углу 0 градусов, вместо нуля надо указать 2*3.14.

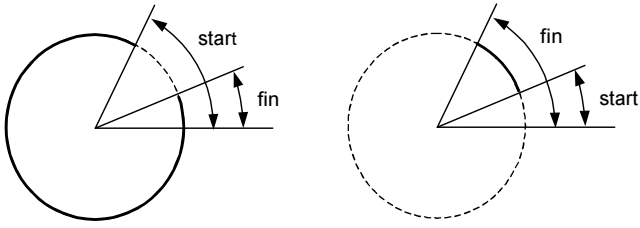


Рис. 5.6. Значения параметров `start` и `fin` определяют дугу

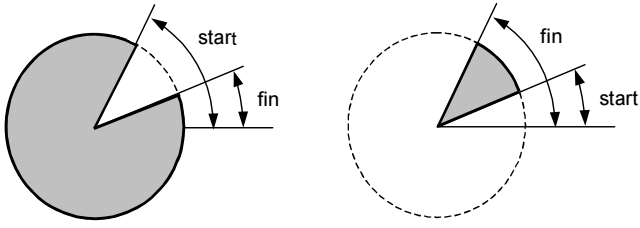


Рис. 5.7. Метод `Circle` позволяет нарисовать сектор

В качестве примера использования метода `Circle` в листинге 5.4 приведена программа, которая рисует на поверхности формы простейшую круговую диаграмму (рис. 5.8).

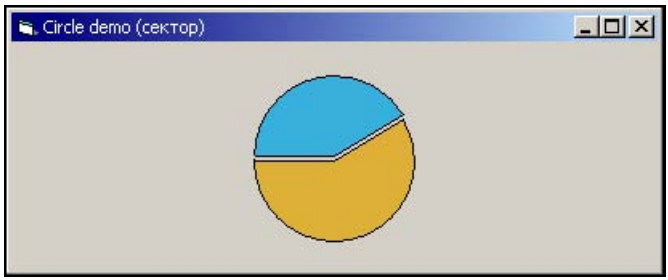


Рис. 5.8. Метод `Circle` позволяет нарисовать простейшую круговую диаграмму

Листинг 5.4. Метод `Circle`

```
Option Explicit

Private Sub Form_Load()
    ScaleMode = vbPixels
End Sub

Private Sub Form_Paint()
    Dim a1, a2 As Double
       ' углы прямых, делящих круг на два сектора
    Dim x, y As Integer
    Dim r As Integer

    Const k = 3.1415926 / 180 ' коэффициент пересчета
                                величины
                              ' угла из градусов в радианы

    a1 = 30 * k
    a2 = 180 * k

    x = Form1.ScaleWidth / 2
    y = Form1.ScaleHeight / 2
    r = 40
```

```
    FillStyle = vbFSSolid ' сектор закрашен

    FillColor = RGB(56, 176, 222)
    Circle (x, y), r, , -a1, -a2
                ' малый сектор (0-30 градусов)

    ' Центр большого сектора смещен на три точки вниз
    ' относительно центра малого сектора
    FillColor = RGB(222, 176, 56)
    Circle Step(0, 3), r, , -a2, -a1
                ' большой сектор (30-360 градусов)

End Sub

Private Sub Form_Resize()
    Form1.Refresh
End Sub
```

Эллипс

Метод `Circle` позволяет нарисовать также эллипс, эллиптическую дугу или эллиптический сектор. Инструкция вызова метода `Circle`, обеспечивающая рисование эллипса, в общем виде выглядит так:

```
Объект.Circle Step(x,y), r, Color, start, fin, aspect
```

Параметры `x` и `y` определяют координату центра эллипса, `Color` — цвет границы. Толщину и стиль линии эллипса, дуги или сектора, а также цвет и стиль заливки внутренней области эллипса (сектора) определяют соответственно параметры `DrawWidh`, `DrawStyle`, `FillColor` и `FillStyle` графической поверхности, на которой рисует метод.

Параметры `start` и `fin` задают точки начала и конца эллиптической дуги. Параметр `r` задает больший радиус эллипса, а па-

раметр `aspect` — коэффициент сжатия (трансформации). Если значение параметра `aspect` меньше единицы, то эллипс получается путем сжатия окружности по вертикали, если больше, то по горизонтали (рис. 5.9). В случае если значение параметра `aspect` равно 1, метод рисует окружность.

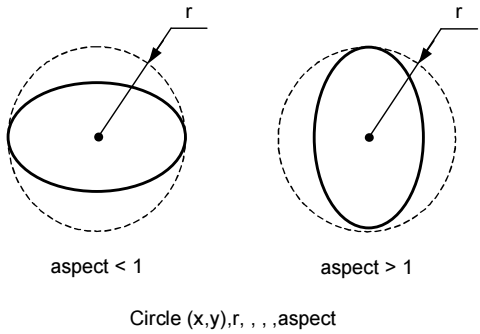


Рис. 5.9. Метод `Circle` позволяет нарисовать эллипс

В качестве примера использования метода `Circle` в листинге 5.5 приведена программа, которая рисует на поверхности формы глобус (рис. 5.10). Следует обратить внимание: программа спроектирована таким образом, что глобус всегда находится в центре окна. Достигается это тем, что процедура обработки события `Resize` вызывает метод `Refresh`, что в свою очередь приводит к возникновению события `Paint`.

Листинг 5.5. Глобус

```
Option Explicit

Private Sub Form_Initialize()
    Form1.ScaleMode = vbPixels
End Sub

Private Sub Form_Paint()
```

```
    Dim x, y As Integer ' центр окна

    x = Form1.ScaleWidth / 2
    y = Form1.ScaleHeight / 2

    FillColor = RGB(56, 176, 222) ' цвет - "Осеннее небо"
    FillStyle = vbSolid
    Circle (x, y), 40, , , , 1  ' круг

    FillStyle = vbFSTransparent
                    ' область внутри эллипса не закрашивать
    Circle Step(0, 0), 40, , , , 0.5 ' "горизонтальный"
                                       эллипс
    Circle Step(0, 0), 40, , , , 0   ' линия
    Circle Step(0, 0), 40, , , , 1.5 ' "вертикальный" эллипс
    Circle Step(0, 0), 40, , , , 3   ' "вертикальный" эллипс

End Sub

' пользователь изменил размер окна
Private Sub Form_Resize()
    Form1.Refresh ' перерисовать окно программы (инициировать
                  ' событие Paint)
End Sub
```

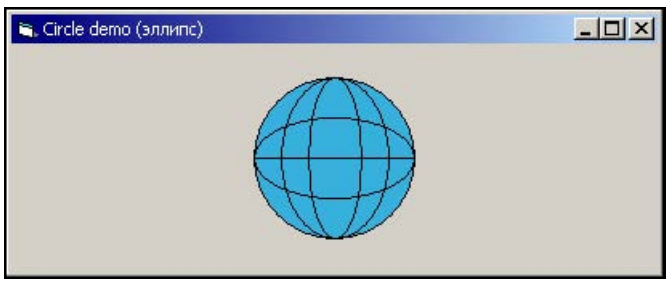


Рис. 5.10. Программа, которая рисует на поверхности формы глобус

Текст

Вывод текста на графическую поверхность выполняет метод `Print`. Инструкция вызова метода в общем виде выглядит так:

```
Объект.Print Строка
```

Параметр `Строка` задает строку, которую надо вывести. Позицию, в которой появится строка, определяет текущее положение указателя графического вывода.

Прежде чем вызвать метод `Print`, надо установить указатель графического вывода (присвоить значение свойствам `CurrentX` и `CurrentY`) в ту точку формы, где должен быть левый верхний угол области вывода текста. Например:

```
CurrentX = 20
CurrentY = 20
Print  "Microsoft Visual Basic"
```

Результат выполнения этих инструкций приведен на рис. 5.11.

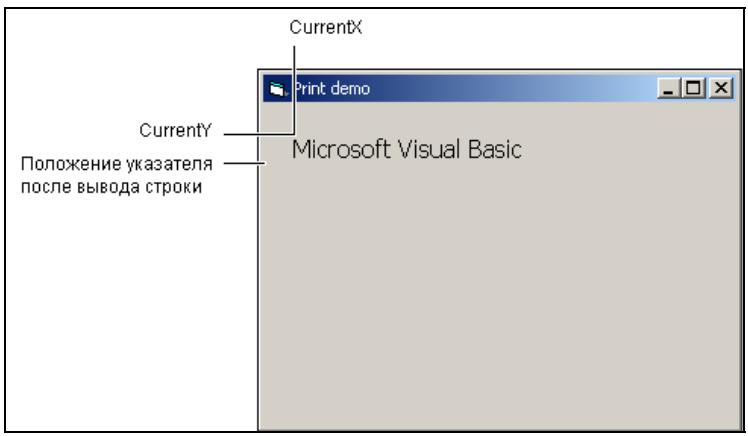


Рис. 5.11. Метод `Print` выводит текст от текущего положения (`CurrentX`, `CurrentY`) указателя графического вывода

После вывода строки указатель автоматически перемещается в точку (`0,y`). Значение координаты `y` зависит от высоты области

вывода текста, которая, в свою очередь, определяется характеристиками (размером) шрифта, используемого для вывода текста.

Шрифт, который метод `Print` применяет для отображения текста, определяет свойство `Font` графической поверхности, цвет символов — свойство `ForeColor`.

Если в качестве параметра метода `Print` указать переменную или выражение, тип которого отличается от `String`, то метод автоматически выполнит преобразование значения в строку. Например, если в программе объявлена переменная `Today` типа `Date`, то в результате выполнения следующих инструкций:

```
Today = Now ' значение функции Now - текущая дата и время
CurrentX = 10
CurrentY = 10
Print Today
```

в форму будут выведены текущие дата и время.

Для преобразования значения выражения в строку нужного формата весьма полезна функция `Format`. У этой функции два параметра: выражение, значение которого надо преобразовать в строку, и строка форматирования. Например, значение `Format(c, "### ### ##0.00 руб.")` — это строковое представление значения переменной `c`, в котором цифры, объединенные в группы по три, разделены пробелами, после десятичного разделителя всегда отображаются две цифры (даже если дробная часть числа равна нулю), а в конец строки добавляется указанное обозначение денежной единицы.

Следует обратить внимание на то, что приведенная строка форматирования обеспечивает автоматическое округление числа по известным правилам. Например, если значение переменной `c` равно `28580,446`, то значением функции `Format` будет строка `28 580,45`.

Наиболее часто используемые форматы приведены в табл. 5.5 (`Now` — функция, которая возвращает текущие дату и время).

Таблица 5.5. *Наиболее часто используемые форматы отображения данных*

Формат	Описание	Пример использования
#.##	Дробное число с двумя знаками после запятой (десятичного разделителя). Если дробная часть числа равна нулю, то цифры дробной части (нули) не отображаются	`Format (x, "#.##")`
#0.00	Дробное число с двумя знаками после запятой (десятичного разделителя). Выполняется округление	`Format (x, "#0.00")`
### ### ##0.00	Дробное число с двумя знаками после десятичного разделителя, цифры целой части объединены в группы по три и разделены пробелами. Выполняется округление. Используется для отображения денежных величин (в этом случае в конец строки форматирования обычно добавляют обозначение денежной единицы)	`Format (summ, "### ### ##0.00 руб.")`
#0.00%	Процент. Значение автоматически умножается на 10 и в конец строки добавляется символ процента	`Format(discount, "#0.00%")`
dd/mm/yy	Дата в формате день, месяц, год. Символ-разделитель определяет операционная система	`Format(Now, "Сегодня dd/mm/yy")`
dddd	День недели	`Format(Now, "Сегодня dd/mm, dddd")`
mmmm	Месяц в полном формате	`Format(Now, "Сегодня dd mmmm, dddd")`

Следующая программа (рис. 5.12) демонстрирует использование метода `Print` для вывода текста на поверхность формы. В диалоговом окне программы отображаются приветствие и инфор-

мация о текущей дате. Следует обратить внимание на то, что текст расположен в центре окна. Положение текста вычисляется на основе информации о размере шрифта, который используется для отображения текста. Текст программы приведен в листинге 5.6. Метод `TextWidth` возвращает ширину области, которую займет текст при его отображении на поверхности формы, метод `TextHeight` — высоту.

Рис. 5.12. Программа, демонстрирующая использование метода `Print` (текст расположен в центре окна)

Листинг 5.6. Приветствие

```
Private Sub Form_Paint()

    Dim h As Integer      ' время (часы)

    Dim st As String

    h = Hour(Time)

    Select Case h
        Case 0 To 4
            st = "Доброй ночи!"
        Case 5 To 12
            st = "Доброе утро!"
```

```
        Case 13 To 16
            st = "Добрый день!"
        Case 17 To 23
            st = "Добрый вечер!"
    End Select

    Font.Name = "Times New Roman"
    FontSize = 28
    ForeColor = RGB(35, 35, 142)

    CurrentX = ScaleWidth / 2 - TextWidth(st) / 2
    CurrentY = ScaleHeight / 2 - TextHeight(st)
    Print st

    Font.Name = "Tahoma"
    FontSize = 12
    ForeColor = vbBlack

    st = Format(Now, "Сегодня d mmmm, dddd")
    CurrentX = ScaleWidth / 2 - TextWidth(st) / 2
    Print st

End Sub
```

Иллюстрации

Компоненты `PictureBox` и `Image` (рис. 5.13) обеспечивают отображение иллюстраций форматов: BMP, GIF, JPG, JPEG, PNG, а также метафайлов (WMF, EMF) и значков (ICO). Компонент `PictureBox` обладает более широкими, по сравнению с компонентом `Image`, возможностями (в частности, на поверхности компонента можно рисовать). Вместе с тем компонент `PictureBox` не позволяет отображать иллюстрацию, размер которой больше размера самого компонента (для отображения больших

иллюстраций следует использовать компонент Image, который обладает возможностью масштабирования). Основные свойства компонента PictureBox приведены в табл. 5.6.

Рис. 5.13. Компоненты PictureBox и Image, обеспечивающие отображение иллюстраций

Таблица 5.6. *Свойства компонента* PictureBox

Свойство	Описание
Picture	Иллюстрация, отображаемая в поле компонента
AutoSize	Признак автоматического изменения размера компонента в соответствии с размером иллюстрации. Если значение свойства равно True, то размер компонента соответствует размеру иллюстрации
Visible	Признак отображения компонента на поверхности формы. Позволяет скрыть компонент (False) или сделать его видимым (True)
Appearance	Стиль компонента. Поле компонента может быть как бы приподнято над поверхностью формы (в этом случае значение свойства равно 3D) или находиться на одном уровне с поверхностью формы (в этом случае значение свойства равно Flat)
BorderStyle	Тип границы компонента. Граница может быть тонкой (FixedSingle) или отсутствовать (None)
ScaleMode	Единица измерения размеров компонента и объектов на нем

Таблица 5.6 *(окончание)*

Свойство	Описание
`ScaleWidth`	Ширина рабочей области компонента (без учета ширины левой и правой границ)
`ScaleHeight`	Высота рабочей области компонента (без учета ширины нижней и верхней границ компонента)
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Width`	Ширина компонента с учетом ширины границы
`Height`	Высота компонента с учетом ширины границы

Иллюстрацию, которая отображается в поле компонента `PictureBox`, можно задать как во время разработки формы приложения, так и во время работы программы.

Чтобы задать иллюстрацию во время разработки формы, надо в строке свойства `Picture` щелкнуть на кнопке с тремя точками (рис. 5.14) и в диалоговом окне **Load Picture** выбрать файл иллюстрации. Следует обратить внимание: файл, в котором находится иллюстрация, выбранная таким образом, во время работы программы не нужен (Visual Basic помещает копию иллюстрации в выполняемый файл программы).

Если размер иллюстрации больше размера компонента, но не превышает размера области формы, которая может быть использована для отображения иллюстрации, то свойству `AutoSize` можно присвоить значение `True`. В результате размер компонента будет соответствовать размеру выбранной иллюстрации. Если есть ограничения на размер поля, которое может использоваться для отображения иллюстрации, то вместо компонента `PictureBox` придется использовать компонент `Image`.

Основные свойства компонента `Image` приведены в табл. 5.7.

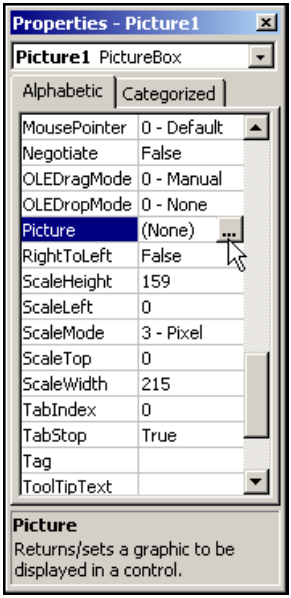


Рис. 5.14. Выбор иллюстрации во время разработки формы

Таблица 5.7. *Свойства компонента* `Image`

Свойство	Описание
`Picture`	Иллюстрация, отображаемая в поле компонента
`Strech`	Признак необходимости выполнения масштабирования иллюстрации таким образом, чтобы она занимала всю область отображения иллюстрации. Масштабирование выполняется, если значение свойства равно `True`
`Width`	Ширина компонента (области отображения иллюстрации) с учетом ширины границы
`Height`	Высота компонента с учетом ширины границы
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы

Таблица 5.7 *(окончание)*

Свойство	Описание
`Visible`	Признак отображения компонента. Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)
`Appearance`	Стиль компонента. Поле компонента может быть как бы приподнято над поверхностью формы (в этом случае значение свойства равно 3D) или находиться на одном уровне с поверхностью формы (в этом случае значение свойства равно `Flat`)
`BorderStyle`	Тип границы компонента. Граница может быть тонкой (`FixedSingle`) или отсутствовать (`None`)

Следует обратить внимание, что после загрузки в компонент `Image` иллюстрации (если значение свойства `Strech` равно `False`) размер компонента будет соответствовать размеру иллюстрации. Если значение свойства `Strech` равно `False`, то изменение значения свойства `Picture` не приведет к изменению значений свойств `Width` и `Height`.

Загрузку иллюстрации в поле компонента `PictureBox` (`Image`) во время работы программы обеспечивает функция `LoadPicture`. В качестве параметра функции надо указать файл иллюстрации, а значение функции присвоить свойству `Picture`. Например, в результате выполнения инструкции

```
Picture1.Picture = LoadPicture("d:\images\ufo.bmp")
```

в поле компонента `Picture1` появится иллюстрация, находящаяся в указанном файле.

Форма следующей программы — Просмотр иллюстраций − приведена на рис. 5.15, значения свойств формы и компонентов — в табл. 5.8, а текст — в листинге 5.7. Эта программа демонстрирует использование компонента `Image`, она позволяет выбрать каталог и просмотреть иллюстрации (например, фотографии, которые в этом каталоге находятся). Так как размер иллюстрации (фотографии) может быть больше размера формы, то для отображения иллюстраций в программе используется компонент `Image`, а не `PictureBox`.

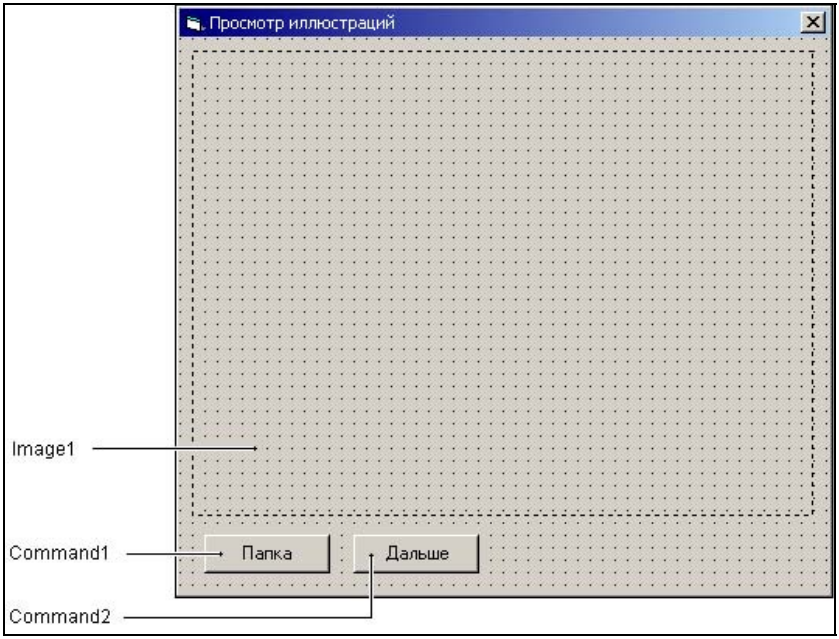


Рис. 5.15. Форма программы Просмотр иллюстраций

Таблица 5.8. *Значения свойств формы и компонентов*

Свойство	Значение
Form1.BorderStyle	FixedSingle
Form1.ScaleMode	Pixel
Form1.ScaleWidth	418
Form1.ScaleHeight	358
Image1.Width	400
Image1.Height	300

Выбор каталога осуществляется в стандартном окне **Обзор папок** (рис. 5.16), которое становится доступным в результате щелчка

на кнопке **Папка** (см. рис. 5.15). Переход к следующей фотографии выполняется щелчком на кнопке **Дальше**.

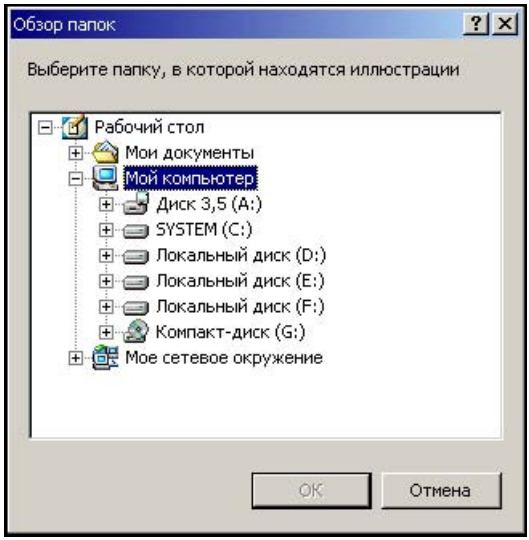


Рис. 5.16. Выбор папки, в которой находятся иллюстрации

Листинг 5.7. Просмотр иллюстраций

```
Option Explicit

Private Declare Function SHBrowseForFolder Lib "shell32"_
        Alias "SHBrowseForFolderA" (ByRef b As Any) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32"_
       (ByVal ResPIDL As Any, ByVal patch As String) As Long

' константы SHELL API
Const CSIDL_DRIVES As Long = 17
Const BIF_RETURNONLYFSDIRS As Long = 1
Const MAX_PATH = 260
```

```
' эта структура используется для передачи информации
' в функцию SHBrowseFolder, которая выводит диалоговое
' окно "Выбор папки"
Private Type bi
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String ' выбранная папка (без пути)
    lpzTitle As String       ' подсказка
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Dim Folder As String  ' папка, в которой находятся
                      иллюстрации
Dim fn As String      ' файл иллюстрации
Dim ImageW, ImageH As Integer
                        ' исходный размер компонента Image
Dim kx, ky, k As Single
                ' коэффициенты масштабирования: по X,Y
                  и общий

Private Sub Form_Load()
    ' запомнить размер компонента Image1
    ImageW = Image1.Width
    ImageH = Image1.Height
End Sub

' выводит в поле компонента Image1 иллюстрацию
Private Sub DisplayPic(fn As String)
    ' fn - имя файла иллюстрации
    Image1.Visible = False
    Image1.Stretch = False
    Image1.Picture = LoadPicture(fn
```

```
    If (Image1.Picture.Width <= ImageW) And _
      (Image1.Picture.Height <= ImageH) Then
       ' размер иллюстрации меньше размера компонента
       Image1.Visible = True
    Else
       ' иллюстрация больше, чем компонент Image
       ' надо масштабировать
       Image1.Stretch = True
       kx = ImageW / Image1.Picture.Width
       ky = ImageH / Image1.Picture.Height
       ' чтобы иллюстрация отображалась без искажения,
       ' коэффициенты масштабирования по обеим осям
       ' должны быть равными
       If kx < ky Then k = kx Else k = ky
       Image1.Width = Image1.Picture.Width * k
       Image1.Height = Image1.Picture.Height * k
       Image1.Visible = True
    End If
End Sub

Private Sub Command1_Click()
   ' Чтобы не потерять имя просматриваемого каталога
   ' (в случае, если пользователь активизирует процесс выбора
   ' папки, а затем откажется), используем буферную
   ' переменную. Если пользователь действительно
   ' выберет новый каталог, то запишем имя этого каталога
   ' в переменную Folder.

   '  Folder = InputBox( _
   ' "Введите имя папки, в которой находятся иллюстрации", _
   ' "Просмотр иллюстраций", "d:\temp\")

    Dim buf As String
    buf = GetFolder(Form1.hwnd, _
         "Выберите папку, в которой находятся иллюстрации")
```

```
    If buf = "" Then Exit Sub

    ' пользователь ввел имя папки
    Folder = buf

    ' получить имя jpg-файла
    fn = Dir(Folder + "*.jpg")

    If fn = "" Then
        MsgBox "В указанном каталоге иллюстраций (jpg) нет", _
                vbExclamation, "Просмотр иллюстраций"
        Exit Sub
    End If

   DisplayPic(Folder + fn)
       ' вывести иллюстрацию в поле компонента Image1

       ' получить имя файла следующей иллюстрации
    fn = Dir
    If fn = "" Then
       ' в каталоге Folder больше нет файлов с расширением
         jpg
        Command2.Enabled = False
    Else
        Command2.Enabled = True
    End If

End Sub

' щелчок на кнопке Дальше
Private Sub Command2_Click()
    DisplayPic(Folder + fn)
       ' вывести иллюстрацию в поле компонента Image1
```

```
        ' получить имя файла следующей иллюстрации
    fn = Dir
    If fn = "" Then
      ' в каталоге Folder больше нет файлов с расширением jpg
        Command2.Enabled = False
    End If
End Sub

Private Function GetFolder(hwnd As Long, msg As String)_
                  As String

    Dim ResPIDL As Long
    Dim t As bi
    Dim Path As String

    Dim r As Long
    Dim p As String

    t.hwndOwner = hwnd
    t.lpzTitle = msg
    t.ulFlags = BIF_RETURNONLYFSDIRS
          ' кнопка OK доступна, если пользователь выбрал
            папку
    t.pszDisplayName = String(MAX_PATH, 0)

    ResPIDL = SHBrowseForFolder(t) ' вывести окно Выбор папки
    If ResPIDL <> 0 Then
        ' пользователь выбрал папку
        ' получить ее полное имя
        Path = String(MAX_PATH, Chr(0))
        r = SHGetPathFromIDList(ResPIDL, Path)
        p = InStr(1, Path, Chr(0))
```

```
        Path = Mid(Path, 1, p - 1)
        If Mid(Path, p - 1, 1) <> "\" Then Path = Path + "\"
    End If
    GetFolder = Path
End Function
```

В начале работы программы процедура обработки события `Load` записывает в переменные `ImageW` и `ImageH` размер компонента `Image1`. Эта информация используется в процессе работы программы для вычисления коэффициента масштабирования.

Процедура обработки события `Click` на кнопке **Папка** вызывает функцию `GetFolder`, которая выводит стандартное окно **Выбор папки**. Непосредственный вывод окна выполняет API-функция `SHBrowseForFolder`. После выбора пользователем нужной папки функция `GetFolder` возвращает вызвышей ее функции обработки события `Click` имя папки. Затем функция `Dir` выполняет поиск файла с расширением jpg. Если в выбранной пользователем папке есть файлы с указанным расширением, то функция `Dir` возвращает имя первого по порядку файла (чтобы получить имя следующего файла, надо вызвать функцию `Dir` еще раз, но уже без параметров).

Вывод иллюстрации в поле компонента `Image` выполняет функция `DisplayPic`. Сначала она присваивает значение `True` свойству `Visible` (тем самым делает компонент `Image` невидимым), отменяет режим автоматического изменения размера иллюстрации и загружает иллюстрацию. Загрузку иллюстрации выполняет функция `LoadPicture`.

Если размер загруженной иллюстрации меньше или равен размеру поля отображения иллюстрации, то свойству `Visible` присваивается значение `True`, компонент `Image` становится видимым, иллюстрация появляется в окне программы.

Если размер иллюстрации больше, то необходимо выполнить масштабирование. Здесь надо обратить внимание на то, что для выполнения масштабирования не достаточно присвоить значе-

ние `True` свойству `Strech` перед загрузкой иллюстрации (если размер иллюстрации не пропорционален размеру компонента, то иллюстрация будет искажена). Чтобы иллюстрация отображалась без искажения, размер компонента `Image` должен быть пропорционален размеру иллюстрации. Поэтому после загрузки иллюстрации надо выбрать коэффициент масштабирования. В рассматриваемой программе в качестве коэффициента масштабирования выбирается минимальный из всех коэффициентов по ширине и высоте. Для вычисления коэффициентов масштабирования используется информация о первоначальном размере компонента `Image` (значение переменных `ImageW` и `ImageH`) и о размере загруженной в данный момент иллюстрации (значение свойств `Picture.Width` и `Picture.Height` компонента `Image`). После вычисления коэффициента масштабирования процедура `DisplayPic` устанавливает размер компонента пропорциональным размеру загруженной иллюстрации, присваивает значение `True` свойствам `Strech` и `Visible` и возвращает управление процедуре обработки события `Click` на кнопке **Папка**, которая вызывает функцию `Dir`, чтобы получить имя файла следующей иллюстрации. Если иллюстрация в выбранном каталоге имеется, то становится доступной кнопка **Дальше**.

Процедура обработки события `Click` на кнопке **Дальше** активизирует процесс отображения иллюстрации, имя которой находится в переменной `fn` и вызывает процедуру `Dir`, чтобы получить имя следующего файла. Если файлов иллюстраций в выбранном пользователем каталоге больше нет (в этом случае функция `Dir` возвращает пустую строку), то свойству `Enabled` кнопки **Дальше** присваивается значение `False`.

Битовые образы

Для формирования сложных изображений используют *битовые образы*. Битовый образ — это картинка (как правило, небольшая), которая находится в памяти компьютера. Типичный пример битового образа — картинка для командной кнопки или

пункта меню. Битовые образы широко используются при программировании игр.

Создать битовый образ (Bitmap) можно путем загрузки заранее подготовленной картинки из файла или *ресурса*. Также битовый образ можно сформировать во время работы программы, скопировав фрагмент другого битового образа или нарисовав нужную картинку непосредственно на поверхности битового образа.

В Visual Basic битовый образ — это объект типа `StdPicture`. Свойства объекта `StdPicture` (табл. 5.9) содержат информацию о битовом образе.

Таблица 5.9. *Свойства объекта* `StdPicture`

Свойство	Описание
`Type`	Тип битового образа. Битовый образ может представлять собой картинку, иконку или метафайл
`Width`	Ширина картинки
`Height`	Высота картинки
`Handle`	Идентификатор объекта. Используется функциями API для доступа к битовому образу

Создание битового образа путем загрузки из файла обеспечивает функция `LoadPicture`. Например, следующий фрагмент кода обеспечивает создание битового образа путем загрузки картинки из файла:

```
Dim ufo As StdPicture
Set ufo = LoadPicture("d:\temp\ufo.bmp")
```

После того как битовый образ сформирован (загружен из файла или из ресурса), его можно вывести на поверхность формы или компонента `PictureBox`. Сделать это можно при помощи метода `PaintPicture`.

Инструкция вызова метода `PaintPicture` в общем виде выглядит так:

```
Объект.PaintPicture aBitmap, x1,y1,w1,h1,x2,y2,w2,h2,opcode
```

Параметры `x2`, `y2`, `w2` и `h2` задают фрагмент битового образа, который надо вывести на поверхность объекта `Объект`.

Параметры `x1`, `y1`, `w1` и `h1` задают положение и размер области на поверхности объекта, в которую надо вывести фрагмент битового образа, заданный параметрами `x2`, `y2`, `w2` и `h2`.

Таким образом, метод `PaintPicture` позволяет вывести на графическую поверхность любой фрагмент битового образа. Параметр `opcode` определяет, каким образом формируется цвет точек результирующего изображения. Например, если значение параметра равно `vbSrcCopy`, то фрагмент битового образа замещает фрагмент графической поверхности.

В простейшем случае в инструкции вызова метода `PaintPicture` достаточно указать битовый образ и координаты точки поверхности, от которой надо вывести картинку. Например, инструкция

```
Form1.PaintPicture ufo,10,20
```

выводит на поверхность формы битовый образ `ufo` — изображение UFO ("неопознанного летающего объекта").

К сожалению, метод `PaintPicture` не позволяет задать "прозрачный" цвет, поэтому на графической поверхности отображается не только изображение объекта, но и фон вокруг него (рис. 5.17).

Однако область вокруг объекта все-таки можно сделать прозрачной. Для этого надо в дополнение битовому образу, где находится картинка, создать другой битовый образ, содержащий *маску* картинки. В этой маске те точки, которые соответствуют точкам картинки и которые должны быть видны, надо закрасить в черный цвет, а точки фона — в белый. В самой же картинке точки фона должны быть окрашены в черный цвет.

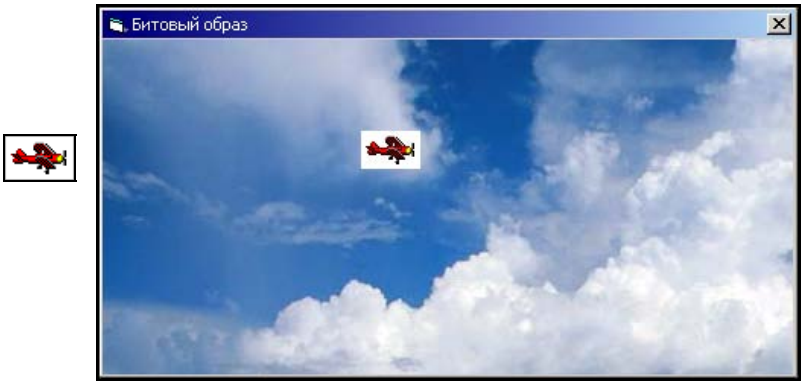


Рис. 5.17. Битовый образ и его изображение на графической поверхности

В качестве примера на рис. 5.18 приведена картинка и соответствующая ей маска.

Рис. 5.18. Пример картинки и маски

Чтобы получить эффект прозрачного фона, надо на графическую поверхность при помощи метода `PaintPicture` вывести сначала маску, затем (в эту же точку) — картинку. При выводе маски значение параметра `opcode` метода `PainPicture` должно быть равно `vbSrcAnd`, при выводе картинки — `vbSrcPaint`.

Программа (листинг 5.8) демонстрирует, как при помощи маски можно добиться эффекта прозрачного фона (после запуска программы в ее окне появляется самолет на фоне облачного неба). Здесь используются два битовых образа: самолет и маска. Загрузку битовых образов выполняет процедура обработки события `Load`. Самолет рисует процедура обработки события `Paint` (сначала она выводит на поверхность формы маску, затем — самолет). Окно программы (после щелчка на кнопке **Маска**) приве-

дено на рис. 5.19. Командные кнопки **Маска** и **Bitmap** позволяют увидеть процесс отображения битового образа.

Листинг 5.8. Прозрачный фон

```
Option Explicit

' битовые образы
Dim plane As StdPicture     ' картинка
Dim plane_m As StdPicture   ' маска
Private Sub Form_Load()
    ' загрузить битовые образы
    Set plane = LoadPicture("d:\temp\plane.bmp")
    Set plane_m = LoadPicture("d:\temp\plane_m.bmp")
End Sub

Private Sub Form_Paint()
    ' вывести маску
    Form1.PaintPicture plane_m, 50, 80, , , , , , , vbSrcAnd
    ' вывести картинку
    Form1.PaintPicture plane, 50, 80, , , , , , , vbSrcPaint
End Sub

' щелчок на кнопке Маска
Private Sub Command1_Click()
    Form1.PaintPicture plane_m, 200, 170, , , , , , ,
vbSrcAnd

End Sub

' щелчок на кнопке Bitmap
Private Sub Command2_Click()
    Form1.PaintPicture plane, 200, 170, , , , , , ,
vbSrcPaint
End Sub
```

Рис. 5.19. Использование маски
для создания эффекта прозрачного фона

Мультипликация

Мультипликацией (анимацией) называют изображение, элементы которого движутся или изменяются.

Существуют два подхода к реализации компьютерной анимации. Первый предполагает наличие заранее подготовленной серии картинок (кадров), последовательное отображение которых и создает эффект анимации (этот подход используют создатели мультфильмов).

Второй подход, который используют разработчики компьютерных игр, предполагает создание кадров анимации "налету", во время работы программы. При реализации этого подхода очередной кадр создается путем вывода изображения *объекта* в нужную точку "экрана" (рис. 5.20). Изображение объекта может быть сформировано из графических примитивов или представлять собой заранее подготовленную картинку.

Программа Пинг-понг (рис. 5.21) демонстрирует принципы создания анимации "на лету", показывает, как можно заставить

объект (в данном случае окружность) двигаться в окне программы. Значения свойств формы этой программы приведены в табл. 5.10.

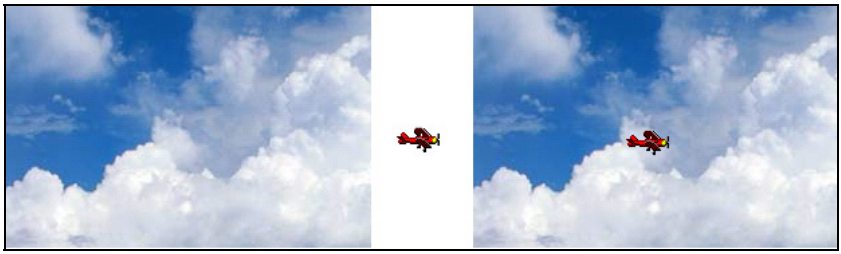


Рис. 5.20. Экран, объект и кадр

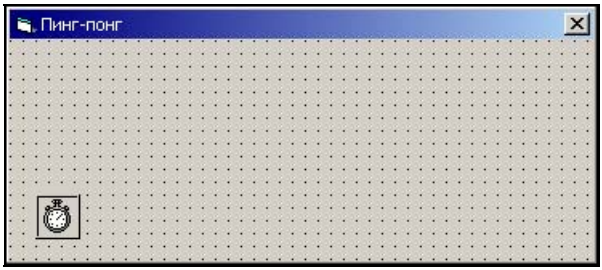


Рис. 5.21. Форма и окно программы Пинг-понг

Таблица 5.10. *Значения свойств формы программы Пинг-понг*

Свойство	Значение
`BorderStyle`	0 — `Fixed Single`
`ScaleMode`	3 — `Pixel`

Чтобы у наблюдателя сложилось впечатление, будто объект (в рассматриваемой программе это мячик) движется в окне программы, надо вывести изображение объекта на графическую поверхность, затем через некоторое время удалить его и снова вывести это изображение, но уже на некотором расстоянии от его

первоначального положения. Подбором времени между выводом и удалением изображения, а также расстояния между новым и старым положением (шага перемещения) можно добиться впечатления равномерного движения.

В рассматриваемой программе (текст приведен в листинге 5.9) основную работу выполняет процедура обработки события `Timer`, которая удаляет изображение мячика (окружность) и рисует его на новом месте при каждом новом возникновении этого события. Направление движения мячика задают переменные `dx` и `dy`. Если значение `dx` (`dy`) положительное, то мячик движется слева направо (сверху вниз), если отрицательное — то справа налево (снизу вверх). При достижении мячиком вертикальной или горизонтальной границы окна знак `dx` (или `dy`) меняется на противоположный, в результате чего мячик как бы отскакивает от границы и движется в противоположном направлении. Скорость движения объекта определяют абсолютные значения `dx` и `dy`, а также период возникновения события `Timer` (задает значение свойства `Interval` компонента `Timer`).

Листинг 5.9. Пинг-понг

```
Option Explicit

Dim x, y As Integer     ' положение мячика
Dim dx, dy As Integer   ' приращение координат
Dim r As Integer        ' радиус мячика
Dim cBall As Long       ' цвет мячика
Dim cBack As Long       ' цвет поля
Dim wp, hp As Integer   ' размер поля (формы)

Private Sub Form_Load()
    r = 2
    x = r
    y = 50
    dx = 1
    dy = 1
```

```
    cBall = RGB(217, 217, 25) ' ярко-золотой
    cBack = RGB(33, 94, 33)   ' зеленый "охотничий"

    Form1.BackColor = cBack

    wp = Form1.ScaleWidth
    hp = Form1.ScaleHeight

    ' настройка и запуск таймера
    Timer1.Interval = 10
    Timer1.Enabled = True

End Sub

' сигнал от таймера
Private Sub Timer1_Timer()

    ' стереть изображение мяча
    Form1.Circle (x, y), r, cBack

    ' ** вычислить новое положение мяча **
    If dx > 0 Then
        ' мяч движется вправо
        If x + dx + r > wp Then dx = -dx
    Else
        ' мяч движется влево
        If x + dx - r < 0 Then dx = -dx
    End If

    If dy > 0 Then
        ' мяч движется вниз
        If y + dy + r > Form1.ScaleHeight Then dy = -dy
    Else
        ' мяч движется вверх
```

```
        If y + dy - r < 0 Then dy = -dy
    End If
    x = x + dx
    y = y + dy

    ' нарисовать мяч в новой точке
    Form1.Circle (x, y), r, cBall

End Sub
```

Наблюдать за движением мячика быстро надоедает, возникает желание отбить его. Следующая программа (листинг 5.10) демонстрирует, как можно сделать графику интерактивной. В окне программы (рис. 5.22) два объекта: мячик и ракетка, которой при помощи клавиш перемещения курсора управляет игрок. И мячик, и ракетку рисует процедура обработки события `Timer`.

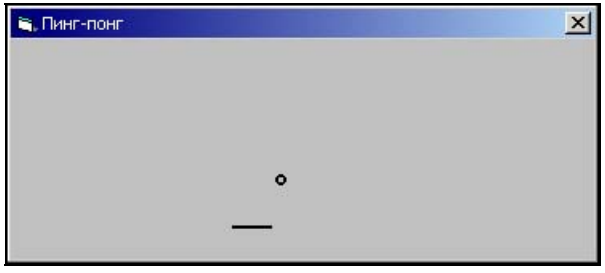


Рис. 5.22. Движением ракетки можно управлять при помощи клавиш перемещения курсора

Факт нажатия клавиши перемещения курсора фиксирует процедура обработки события `KeyDown`, которое возникает при нажатии клавиши клавиатуры. Эта процедура записывает в переменную `rd` код направления движения ракетки. Значение переменной `rd` контролирует процедура обработки события `Timer` и, в зависимости от ее значения, сдвигает ракетку влево или вправо. Следует обратить внимание на то, как процедура обработки события `Timer` сдвигает ракетку. Она каждый раз, когда надо

сдвинуть ракетку, не рисует ее заново (предварительно стерев), а только стирает небольшой кусочек с одной стороны и дорисовывает с другой. Процедура обработки события KeyUp фиксирует факт отпускания клавиши — записывает в переменную rd ноль и тем самым информирует процедуру обработки события Timer о том, что ракетку перерисовывать не надо.

Листинг 5.10. Пинг-понг - 2

```
Option Explicit
Dim x, y As Integer     ' положение мячика
Dim dx, dy As Integer   ' приращение координат
Dim r As Integer        ' радиус мячика
Dim cBall As Long       ' цвет мячика
Dim cBack As Long       ' цвет поля
Dim wp, hp As Integer   ' размер поля (формы)

' это переменные для управления движением ракетки
Dim rd As Integer ' 0 - ракетка не движется;
                  ' 1 - движется влево; 2 - движется вправо
Dim rx1, rx2 As Integer ' координаты X концов ракетки
Dim ry As Integer       ' координата Y концов ракетки
Dim rdx As Integer      ' шаг перемещения ракетки

Private Sub Form_Load()
    r = 3
    x = r
    y = 50
    dx = 1
    dy = 1
    cBall = RGB(217, 217, 25)
    cBack = RGB(33, 94, 33)

    Form1.BackColor = cBack
```

```
    wp = Form1.ScaleWidth
    hp = Form1.ScaleHeight

    ' ** управление ракеткой **
    rd = 0        ' ракетка на месте
    rx1 = 100
    rx2 = 125
    ry = Form1.ScaleHeight - 20
    rdx = 2 ' шаг движения ракетки

End Sub

Private Sub Form_Paint()
        Form1.Line (rx1, ry)-(rx2, ry), vbRed
                              ' нарисовать ракетку
End Sub

Private Sub Timer1_Timer()

    ' стереть изображение мяча
    Form1.Circle (x, y), r, cBack



    ' ** вычислить новое положение мяча **
    If dx > 0 Then
        ' мяч движется вправо
        If x + dx + r > wp Then dx = -dx
    Else
        ' мяч движется влево
        If x + dx - r < 0 Then dx = -dx
    End If

    If dy > 0 Then
        ' мяч движется вниз
```

```
    If (x >= rx1) And (x <= rx2) And (y = ry - r - 1) Then
        ' мячик попал в ракетку
        dy = -dy
    Else
        If y + dy + r > Form1.ScaleHeight Then dy = -dy
    End If
Else
    ' мяч движется вверх
    If (x >= rx1) And (x <= rx2) And (y >= ry - r)_
                  And (y <= ry + r) Then
' Мячик отскочил от нижней стенки и попал в ракетку снизу.
        ' Чтобы не было дырок в ракетке, перерисуем ее.
        Line (rx1, ry)-(rx2, ry), vbRed
    End If
    If y + dy - r < 0 Then dy = -dy
End If
x = x + dx
y = y + dy

' нарисовать мяч в новой точке
Form1.Circle (x, y), r, cBall

'*** ракетка ***
If rd <> 0 Then
    ' игрок нажал и удерживает одну из клавиш
    ' "стрелка вправо" или "стрелка влево"
    ' (см. Form_KeyDown)
    If rd = 1 Then
       ' вправо
       If rx2 < wp Then
          Line (rx1, ry)-(rx1 + rdx, ry), cBack
                          ' стереть часть слева
```

```
            Line (rx2, ry)-(rx2 + rdx, ry), vbRed
                                ' дорисовать справа
            rx1 = rx1 + rdx
            rx2 = rx2 + rdx
         End If
      Else
         ' влево
         If rx1 > 1 Then
           Line (rx2, ry)-(rx2 - rdx, ry), cBack
                                ' стереть часть справа
           Line (rx1 - rdx, ry)-(rx1 + rdx, ry), vbRed
                                ' дорисовать
           rx1 = rx1 - rdx
           rx2 = rx2 - rdx
         End If
      End If
   End If


End Sub

' нажата клавиша
Private Sub Form_KeyDown(KeyCode As Integer, Shift As
Integer)
   If rd <> 0 Then
      ' пользователь удерживает клавишу, ракетка движется
      Exit Sub
   End If

   Select Case KeyCode
      Case 37 ' Шаг (стрелка) право
         rd = 2
      Case 39 ' Шаг (стрелка) влево
         rd = 1
```

```
    End Select
End Sub

' отпущена клавиша
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
    rd = 0
End Sub
```

Использование битовых образов

В предыдущих примерах изображение объектов формировалось из графических примитивов. Теперь на примере программы Полет в облаках (листинг 5.11) рассмотрим, как можно существенно улучшить графику программы за счет использования битовых образов.

Как и в предыдущих программах, эффект перемещения объекта достигается за счет перерисовки изображения объекта с некоторым смещением относительно его прежнего положения. Процесс вывода изображения объекта (битового образа) с использованием маски был подробно описан. Теперь рассмотрим, что нужно сделать, чтобы стереть нарисованный объект.

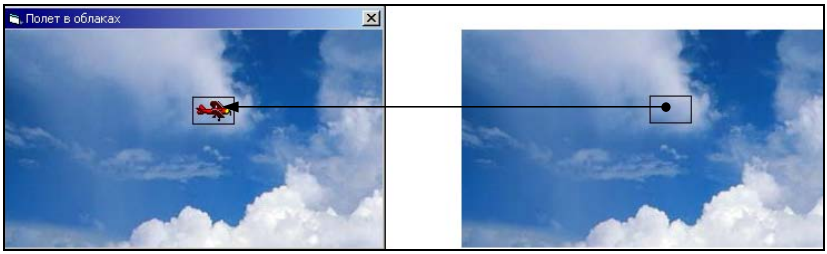


Рис. 5.23. Чтобы стереть изображение объекта, надо вывести фрагмент фона в ту область, где находится объект

Удалить объект (восстановить фон) можно путем перерисовки всей фоновой картинки или только той ее части, которая была

перекрыта объектом. В рассматриваемой программе используется второй подход. Удаляет изображение объекта метод `PaintPicture`, который копирует фрагмент фона в ту область графической поверхности, в которой находится изображение объекта (рис. 5.23).

Листинг 5.11. Полет в облаках

```
Option Explicit

' битовые образы
Dim back As StdPicture      ' фон
Dim plane As StdPicture     ' самолет
Dim plane_m As StdPicture   ' маска

Dim x, y As Integer    ' координаты объекта (самолета)
Dim dx, dy As Integer  ' приращение координат
Dim w, h As Integer ' ширина и высота битового образа

Private Sub Form_Load()
    Set back = LoadPicture("d:\temp\sky.bmp")
    Set plane = LoadPicture("d:\temp\plane.bmp")
    Set plane_m = LoadPicture("d:\temp\plane_m.bmp")

' установить размер формы равным размеру фонового рисунка
' ширина и высота объекта StdPicture измеряется в единицах,
' которые называются HiMETRIC
' для преобразования величины из HiMETRIC в твипы
' используются методы ScaleX и ScaleY

    Form1.Width = ScaleX(back.Width, vbHimetric, vbTwips)
    Form1.Height = (Form1.Height - Form1.ScaleHeight) + _
                   ScaleY(back.Height, vbHimetric, vbTwips)
```

```
    Form1.Picture = back

    w = ScaleX(plane.Width, vbHimetric, vbPixels)
    h = ScaleY(plane.Height, vbHimetric, vbPixels)

    Form1.ScaleMode = vbPixels

    ' начальное положение объекта
    x = 0
    y = 100

' скорость движения объекта определяют приращение координаты
' и период следования сигналов от таймера
    dx = 1
    Timer1.Interval = 10
    Timer1.Enabled = True

End Sub

Private Sub Timer1_Timer()
     ' стереть объект
     Form1.PaintPicture back, x, y, w, h, x, y, w, h

     ' *** нарисовать объект ***
     ' вывести маску
     Form1.PaintPicture plane_m, x, y, , , , , , , vbSrcAnd
     ' вывести картинку
     Form1.PaintPicture plane, x, y, , , , , , , vbSrcPaint

     If x < Form1.ScaleWidth Then
        x = x + dx
     Else
```

```
        x = 0
        dx = 1 + 1 * Rnd

    End If

End Sub
```

Процедура обработки события `Load` загружает необходимые для работы программы битовые образы (изображение объекта, соответствующую изображению объекта маску объекта и фоновый рисунок), устанавливает размер окна в соответствии с размером фонового рисунка, задает начальное положение объекта, выполняет настройку и запуск таймера.

Основную работу по формированию кадров мультипликации выполняет процедура обработки события `Timer`. Она сначала путем копирования фрагмента фонового рисунка на поверхность формы стирает изображение объекта (восстанавливает "испорченный" фон), затем рисует объект на новом месте.

Запустив приведенную программу, можно заметить, что изображение самолета мерцает. Это объясняется тем, что глаз успевает заметить, как самолет исчез и затем появился снова. Чтобы устранить мерцание, необходимо сделать так, чтобы самолет не исчезал, а просто смещался. Добиться этого можно, если создавать изображение не на поверхности формы, а на невидимой графической поверхности (в памяти), а уже затем выводить готовое изображение на поверхность формы. Приведенная в листинге 5.12 программа демонстрирует реализацию описанного метода формирования изображения.

Листинг 5.12. Полет в облаках - 2

```
Option Explicit

' функции API
Private Declare Function GetDC Lib "user32"_
    (ByVal hwnd As Long) As Long
```

```
Private Declare Function DeleteDC Lib "gdi32"_
    (ByVal hdc As Long) As Long
Private Declare Function CreateCompatibleDC Lib "gdi32" _
    (ByVal hdc As Long) As Long
Private Declare Function SelectObject Lib "gdi32" _
    (ByVal hdc As Long, ByVal hObject As Long) As Long
Private Declare Function BitBlt Lib "gdi32" _
    ByVal hDestDC As Long, ByVal x As Long, ByVal y As Long, _
    ByVal nWidth As Long, ByVal nHeight As Long, _
    ByVal hSrcDC As Long, ByVal xSrc As Long, _
    ByVal ySrc As Long, ByVal dwRop As Long) As Long

Private Const SRCAND = &H8800C6
Private Const SRCPAINT = &HEE0086
Private Const SRCCOPY = &HCC0020

Dim hdcs As Long ' контекст источника
Dim hdcd As Long ' контекст приемника

Dim plane As StdPicture      ' объект (самолет)
Dim plane_m As StdPicture    ' маска объекта
Dim back As StdPicture       ' фоновый рисунок
Dim aFrame As StdPicture     ' кадр


Dim x, y As Integer   ' координаты кадра
Dim dx, dy As Integer ' шаг движения объекта
Dim sx, sy As Integer ' смещение объекта в кадре

Dim w, h As Integer ' размер кадра

Dim r As Long ' результат выполнения API-функции
```

```
Private Sub Form_Load()

    ' *** загрузить картинки ***
    Set back = LoadPicture("d:\temp\sky.bmp")
                                        ' фоновый рисунок
    Set plane = LoadPicture("d:\temp\plane.bmp")
                                        ' объект
    Set plane_m = LoadPicture("d:\temp\plane_m.bmp")
                                        ' маска

    ' Создадим объект aFrame таким "хитрым" способом
    ' Внимание! Глубина цветовой палитры битового образа,
    ' используемого для инициализации aFrame, должна
    ' совпадать с глубиной цветовой палитры битового образа
    ' back.
     Set aFrame = LoadPicture("d:\temp\plane.bmp")

' установить размер формы равным размеру фонового рисунка
' ширина и высота объекта StdPicture измеряется в единицах,
' которые называются HiMETRIC
' для преобразования величины из HiMETRIC в твипы
' используются методы ScaleX и ScaleY
    Form1.ScaleMode = vbTwips
    Form1.Width = ScaleX(back.Width, vbHimetric, vbTwips)
    Form1.Height = (Form1.Height - Form1.ScaleHeight) + _
                    ScaleY(back.Height, vbHimetric, vbTwips)
    Form1.Picture = back

    ' определить размер битового образа объекта
    w = Int(ScaleX(plane.Width, vbHimetric, vbPixels))
    h = Int(ScaleY(plane.Height, vbHimetric, vbPixels))

    Form1.ScaleMode = vbPixels
```

```
    x = -w: y = 70    ' положение объекта
    dx = 1: dy = 0    ' шаг движения по X и Y
    sx = dx: sy = dy  ' смещение объекта в кадре

End Sub

Private Sub Form_Paint()
     Call frame
End Sub

' выводит изображение объекта со смещением
' относительно его текущего положения
Sub frame()

   hdcd = CreateCompatibleDC(Form1.hdc)
                          ' создать контекст для приемника
   r = SelectObject(hdcd, aFrame)
                          ' связать контекст и битовый образ

   hdcs = CreateCompatibleDC(Form1.hdc)
                          ' создать контекст для источника

   ' *** формируем кадр ***
   r = SelectObject(hdcs, back) ' фон
   r = BitBlt(hdcd, 0, 0, w + dx, h + dy, hdcs, x, y, SRCCOPY)

   r = SelectObject(hdcs, plane_m) ' маска
   r = BitBlt(hdcd, sx, sy, w, h, hdcs, 0, 0, SRCAND)

   r = SelectObject(hdcs, plane) ' объект
   r = BitBlt(hdcd, sx, sy, w, h, hdcs, 0, 0, SRCPAINT)

   ' отобразить кадр
   r = SelectObject(hdcs, aFrame)
```

```
    r = BitBlt(Form1.hdc, x, y, w + dx, h + dy, hdcd,_
               0, 0, SRCCOPY)

    ' уничтожить созданный контекст
     r = DeleteDC(hdcs)
     r = DeleteDC(hdcd)
End Sub

Private Sub Timer1_Timer()
    Call frame ' отобразить кадр (объект)

     If x < Form1.ScaleWidth Then
             x = x + dx
         Else
             x = -w
             y = 70 + Rnd * 70
             dx = 1 + Rnd * 1
     End If

End Sub
```

Как и в предыдущей программе, изображения фона, объекта (самолета) и маски объекта загружаются из файлов и представляют собой объекты типа `StdPicture`. Очередной кадр, точнее фрагмент кадра, формируется на находящейся в памяти невидимой графической поверхности `aFrame`.

Примечание

Вывод графики на уровне операционной системы выполняют API-функции (API — Application Program Interface). Обычно программа не использует функции API напрямую. Однако в некоторых случаях, когда решить поставленную задачу стандартными средствами нельзя, или когда использование API-функций делает решение поставленной задачи более эффективным, их применение вполне оправдано. Чтобы можно было использовать API-функцию, в текст программы надо поместить объявление этой функции. Сформи-

ровать объявление необходимой API-функции, константы или типа данных можно при помощи утилиты API Text Viewer, которая входит в состав Visual Basic.

Вывод очередного кадра выполняет процедура `Frame`. Сначала она копирует на поверхность `aFrame` фрагмент фонового рисунка (стирает объект), затем со смещением `dx` от левого края выводит изображение объекта. Таким образом получается кадр, в котором объект сдвинут относительно его текущего положения. После этого сформированный кадр выводится на поверхность формы, в результате чего количество операций вывода на видимую графическую поверхность сокращается с трех до одной.

Формирование кадра и его вывод обеспечивает API-функция `BitBlt`, которая, как и метод `PaintPicture`, копирует изображения с одной графической поверхности на другую (использование `BitBlt` вместо `PaintPicture` объясняется тем, что метод `PaitPicture` не может "рисовать" на невидимой поверхности).

Следует обратить внимание на то, что начальным значением переменной `x`, которая определяет положение левой верхней точки битового образа движущейся картинки, является отрицательное число, равное ширине битового образа. Поэтому в начале работы нашей программы самолет не виден (картинка отрисовывается за границей видимой области). С каждым событием `Timer` значение координаты `x` увеличивается, на экране появляется та часть битового образа, координаты которой больше нуля. Таким образом, у наблюдателя создается впечатление, что самолет вылетает из-за левой границы окна.

Загрузка битового образа из ресурса программы

Загрузить необходимую программе картинку можно не только из файла, но также из *ресурса*. Ресурс — это данные, которые находятся в выполняемом файле. Различают следующие виды ресурсов: битовый образ, строка символов и курсор. Несомненным плюсом использования ресурсов является сокращение количества файлов, образующих приложение (все необходимые програм-

ме картинки находятся в выполняемом файле), повышение защищенности программы (пользователь, например, не сможет случайно удалить нужные программе файлы или внести изменения в них).

Чтобы программа могла загрузить из ресурса данные (например, битовый образ), необходимо создать файл ресурсов, добавить созданный файл ресурсов в проект, изменить программу так, чтобы она загружала данные не из файла, а из ресурса, а после этого выполнить компиляцию.

Создание файла ресурсов

Создать файл ресурсов можно при помощи утилиты VB Resource Editor, которая входит в Visual Basic. Чтобы запустить эту утилиту, надо в меню **Tools** выбрать команду **Resource Editor**. Если этой команды в меню нет, то можно в меню **Add-Ins** выбрать команду **Add-In Manager**. Затем в появившемся окне **Add-In Manager** (рис. 5.24) надо выбрать строку VB 6 Resource Editor, установить переключатели **Loaded/Unloaded** и **Load on Startup** и щелкнуть на кнопке **OK**.

Чтобы добавить в файл ресурсов битовый образ, надо щелкнуть на кнопке **Add Bitmap** (рис. 5.25) и в появившемся окне выбрать bmp-файл, содержащий ту картинку, которую надо поместить в ресурс. В результате в файл ресурсов будет добавлен новый ресурс — битовый образ, а в окне редактора ресурсов появится новый элемент списка **Bitmap**. Число, которое находится справа от значка битового образа, — это автоматически сформированный идентификатор ресурса, именно его надо будет использовать в программе (в инструкции загрузки битового образа из ресурса). Идентификатор ресурса можно изменить. Для этого надо сделать щелчок правой кнопкой мыши на значке ресурса, из контекстного меню выбрать команду **Properties** и в поле **ID** окна редактора свойств ввести новый идентификатор ресурса. Следует обратить внимание на то, что при записи идентификатора ресурса можно использовать только строчные (большие) буквы латинского алфавита и цифры.

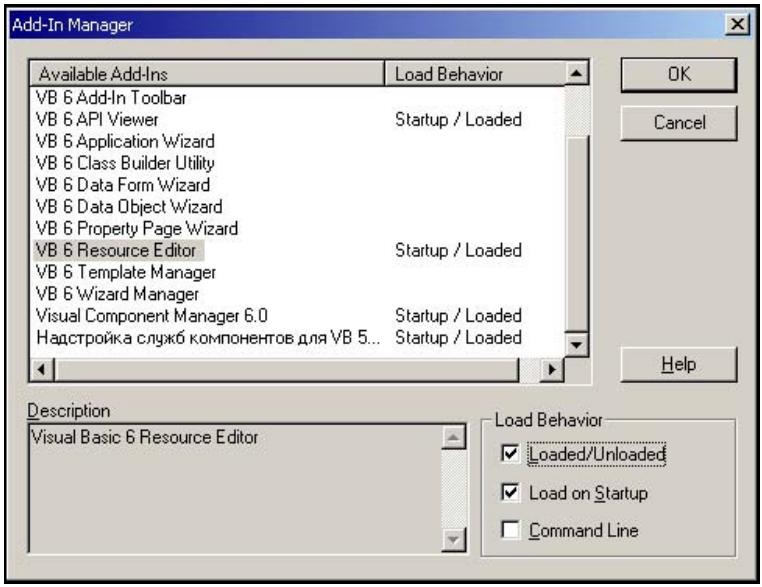


Рис. 5.24. Установка редактора ресурсов

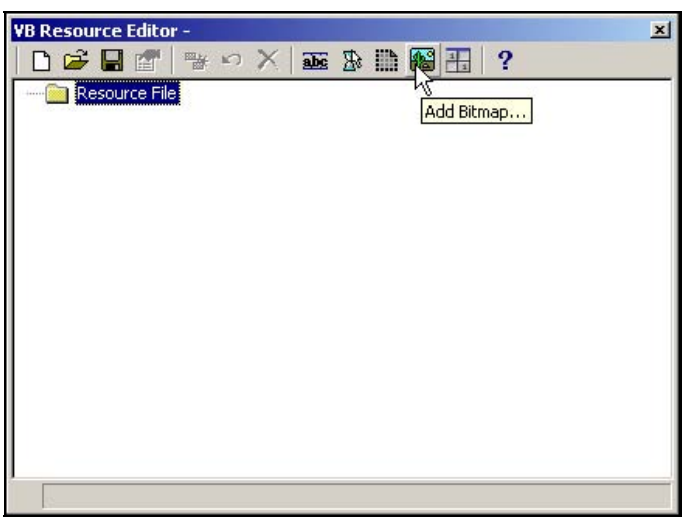


Рис. 5.25. Чтобы добавить в ресурс картинку, надо сделать щелчок на кнопке **AddBitmap**

После того как все необходимые программе иллюстрации будут добавлены, файл ресурсов надо сохранить в каталоге проекта.

На рис. 5.26 приведено окно редактора ресурсов в конце работы над файлом ресурсов для программы Мультипликация. Файл называется bitmaps.res и содержит три битовых образа: самолет (101), маску (102) и фоновый рисунок (103).

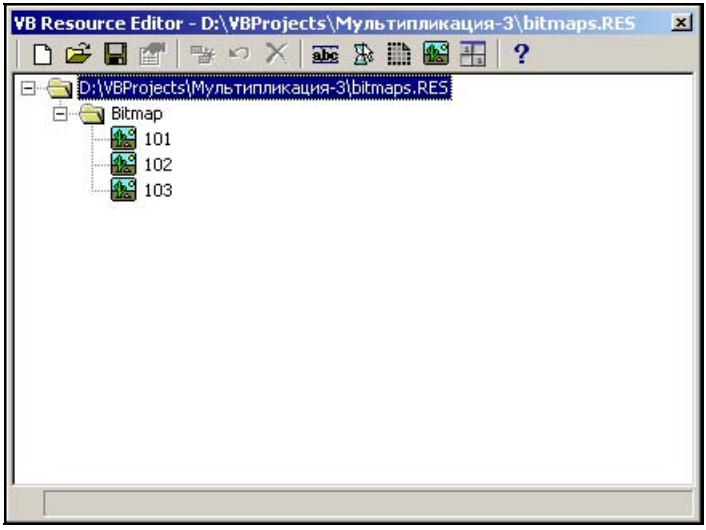


Рис. 5.26. Пример файла ресурсов

Доступ к файлу ресурсов

Чтобы в процессе компиляции ресурсы, находящиеся в файле ресурсов, были помещены в выполняемый файл, в проект надо добавить ссылку на файл ресурсов: в контекстном меню окна **Project Explorer** выбрать команду **Add ▸ Resource File** и затем — файл ресурсов. В результате в проект будет добавлена ссылка на файл ресурсов (рис. 5.27).

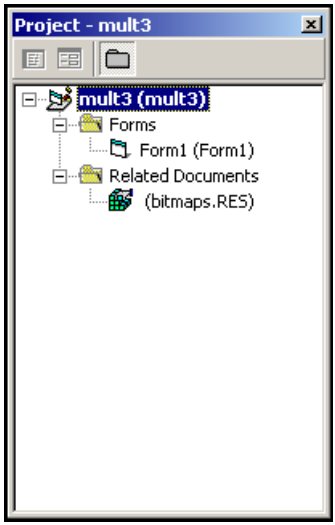


Рис. 5.27. Результат добавления ссылки на файл ресурсов

Загрузка ресурса

Загрузку битового образа из ресурса можно выполнить при помощи функции `LoadResPicture` или `LoadBitmap`.

Функцию `LoadResPicture` можно использовать для загрузки картинки в компонент `PictureBox`, `Image`, или в объект `StdPicture`. В качестве параметров функции `LoadResPicture` надо указать идентификатор битового образа и константу `vbResBitmap`.

Например, инструкция

```
Image1 = LoadResPicture(101, vbResBitmap)
```

обеспечивает загрузку из ресурса и отображение в поле компонента `Image1` иллюстрации (битового образа), идентификатором которой является `101`. Аналогичным образом можно выполнить загрузку иллюстрации в поле компонента `PictureBox`.

API-функция `LoadBitmap` также обеспечивает загрузку битового образа из ресурса. В качестве параметров функции надо указать

идентификатор программы, которая вызывает функцию LoadBitmap и идентификатор ресурса. Например, инструкция

```
hBitmap3 = LoadBitmap(App.hInstance, "#101")
```

загружает из ресурса картинку и записывает в переменную hBitmap3 указатель на область памяти, в которой эта картинка находится.

В качестве примера в листинге 5.13 приведен текст программы Полет в облаках - 3, в которой битовые образы загружаются из ресурса.

Листинг 5.13. Полет в облаках - 3

```
Option Explicit

' функции API
Private Declare Function GetDC Lib "user32"_
                (ByVal hwnd As Long) As Long
Private Declare Function DeleteDC Lib "gdi32"_
                (ByVal hdc As Long) As Long

Private Declare Function CreateCompatibleDC Lib "gdi32"_
               (ByVal hdc As Long) As Long

Private Declare Function CreateCompatibleBitmap Lib "gdi32"_
                  (ByVal hdc As Long, ByVal nWidth As Long,_
                   ByVal nHeight As Long) As Long

Private Declare Function SelectObject Lib "gdi32"_
          (ByVal hdc As Long, ByVal hObject As Long) As Long

Private Declare Function BitBlt Lib "gdi32"_
   (ByVal hDestDC As Long, ByVal x As Long, ByVal y As Long,_
```

```
    ByVal nWidth As Long, ByVal nHeight As Long,_
    ByVal hSrcDC As Long, ByVal xSrc As Long,_
    ByVal ySrc As Long, ByVal dwRop As Long) As Long

Private Const SRCAND = &H8800C6
Private Const SRCPAINT = &HEE0086
Private Const SRCCOPY = &HCC0020

Dim hBitmap4 As Long   ' кадр: фон(область)+ маска + картинка

Dim hdcs As Long ' контекст источника
Dim hdcd As Long ' контекст приемника

Dim plane As StdPicture     ' объект (самолет)
Dim plane_m As StdPicture   ' маска объекта
Dim back As StdPicture      ' фоновый рисунок


Dim x, y As Integer    ' координаты кадра
Dim dx, dy As Integer ' шаг движения объекта
Dim sx, sy As Integer ' смещение объекта в кадре.
          ' При движении вправо (вниз) смещение равно dx (dy),
          ' при движении влево (верх) - нулю.

Dim w, h As Integer ' размер кадра

Dim r As Long ' результат выполнения API функции

Private Sub Form_Load()
    ' *** загрузить картинки из ресурса***
    Set plane = LoadResPicture(101, vbResBitmap)     ' объект
```

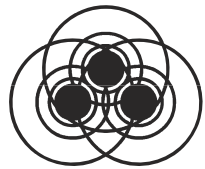
```
    Set plane_m = LoadResPicture(102, vbResBitmap) ' маска
    Set back = LoadResPicture(103, vbResBitmap)
                                          ' фоновый рисунок

' установить размер формы равным размеру фонового рисунка
    Form1.ScaleMode = vbTwips
    Form1.Width = ScaleX(back.Width, vbHimetric, vbTwips)
    Form1.Height = (Form1.Height - Form1.ScaleHeight) + _
                   ScaleY(back.Height, vbHimetric, vbTwips)
    Form1.Picture = back

    ' определить размер битового образа объекта
    w = Int(ScaleX(plane.Width, vbHimetric, vbPixels))
    h = Int(ScaleY(plane.Height, vbHimetric, vbPixels))

    Form1.ScaleMode = vbPixels

    ' создать битовый образ - кадр
  hBitmap4 = CreateCompatibleBitmap(Form1.hdc, w + dx, h + dy)

    x = -w: y = 70    ' положение объекта
    dx = 1: dy = 0    ' шаг движения по X и Y
    sx = dx: sy = dy ' смещение объекта в кадре
End Sub

Private Sub Form_Paint()
    Call frame
End Sub

' выводит изображение объекта
Sub frame()
   hdcd = CreateCompatibleDC(Form1.hdc)
                          ' создать контекст для приемника
```

```
   r = SelectObject(hdcd, hBitmap4)
                              ' связать контекст и битовый образ
   hdcs = CreateCompatibleDC(Form1.hdc)
                              ' создать контекст для источника

   ' формируем кадр
   r = SelectObject(hdcs, back) ' фон
   r = BitBlt(hdcd, 0, 0, w + dx, h + dy, hdcs, x, y, SRCCOPY)

   r = SelectObject(hdcs, plane_m) ' маска
   r = BitBlt(hdcd, sx, sy, w, h, hdcs, 0, 0, SRCAND)

   r = SelectObject(hdcs, plane) ' объект
   r = BitBlt(hdcd, sx, sy, w, h, hdcs, 0, 0, SRCPAINT)

   ' отобразить кадр
   r = SelectObject(hdcs, hBitmap4) ' объект
   r = BitBlt(Form1.hdc, x, y, w + dx, h + dy, hdcd,_
              0, 0, SRCCOPY)

   ' уничтожить созданный контекст
    r = DeleteDC(hdcs)
    r = DeleteDC(hdcd)
End Sub

Private Sub Timer1_Timer()
   Call frame ' отобразить кадр (объект)

    If x < Form1.ScaleWidth Then
            x = x + dx
       Else
```

```
            x = -w
            y = 70 + Rnd * 70
            dx = 1 + Rnd * 1
    End If

End Sub
```

Еще раз следует обратить внимание на то, что при загрузке картинок из ресурса файл ресурсов и файлы иллюстраций, которые использовались в процессе создания файла ресурсов, во время работы программы не нужны. Преимущества загрузки картинок из ресурса очевидны: при распространении программы не надо заботиться о том, чтобы во время работы программы были доступны файлы иллюстраций — все необходимые программе картинки находятся в файле программы.

Глава 6



Мультимедиа

Большинство современных программ, работающих в среде Windows, являются мультимедийными. Такие программы обеспечивают просмотр видеороликов и мультипликации, воспроизведение музыки, речи, звуковых эффектов. Типичными примерами мультимедийных программ являются игры и обучающие программы.

Visual Basic предоставляет в распоряжение программиста компонент `MMControl` (Microsoft Multimedia Control), который обеспечивает воспроизведение звука и видео. Для воспроизведения звука можно также использовать API-функцию `PlaySound`.

Функция *PlaySound*

API-функция `PlaySound` весьма удобна для реализации звуковых эффектов (например, фонового музыкального сопровождения в игровой программе или звукового сигнала, сопровождающего появление сообщения). Функция позволяет воспроизвести звук, который находится в WAV-файле. Также функцию `PlaySound` можно использовать для воспроизведения стандартных звуковых сигналов Windows.

Чтобы функцию `PlaySound` можно было использовать, в текст программы надо поместить ее объявление:

```
Private Declare Function PlaySound Lib "winmm.dll" Alias
"PlaySoundA"  (ByVal lpszName As String, ByVal hModule As
Long, ByVal dwFlags As Long)
 As Long
```

Инструкция вызова функции PlaySound в общем виде выглядит так:

```
r = PlaySound(Файл, 0, Режим)
```

Параметр Файл задает имя файла, который надо воспроизвести, параметр Режим — режим воспроизведения. Различают два режима воспроизведения звука: синхронный и асинхронный.

Если звук воспроизводится в синхронном режиме, то функция PlaySound возвращает управление вызвавшей ее программе только после того, как процесс воспроизведения будет завершен. При воспроизведении звука в асинхронном режиме управление вызвавшей программе передается сразу, как только процесс воспроизведения будет активизирован.

Чтобы задать режим воспроизведения, надо указать в качестве параметра Режим константу &H1 (асинхронный режим) или &H0 (синхронный режим). Например, инструкция

```
PlaySound ("tada.wav", 0, &H1)
```

обеспечивает воспроизведение звукового файла в асинхронном режиме.

Следует обратить внимание на то, что если имя звукового файла указано не полностью (не задан путь к файлу), то функция PlaySound выполнит поиск файла: сначала в текущем каталоге, затем в каталогах Windows. Если все-таки указанный звуковой файл не будет найден, то будет воспроизведен "стандартный звук" (задается в настройках Windows). Программист может заблокировать воспроизведение "стандартного звука" (для этого значение параметра **Режим** надо увеличить на &H2).

Рассмотрим программу, демонстрирующую использование функции PlaySound. Появление окна программы (рис. 6.1) сопровождается звуком "УТОПИЯ — открыть". Затем пользователь может выбрать один из находящихся в папке Media звуковых файлов и прослушать его в асинхронном или в синхронном режиме. Завершение работы программы сопровождается звуком "УТОПИЯ — свернуть". Для выбора звукового файла используется компонент FileListBox (File1). Настройку компонента File1, а также инициализацию воспроизведения звукового сигнала, сопровождающего появление окна программы, выполняет

процедура обработки события `Initialize`. Следует обратить внимание на то, что если по какой-либо причине файла УТОПИЯ — открыть.wav в каталоге Media не окажется, то появление окна не будет сопровождаться "стандартным звуком", т. к. в инструкции вызова функции `PlaySound` указан параметр `SND_NODEFAULT`. Текст программы приведен в листинге 6.1.

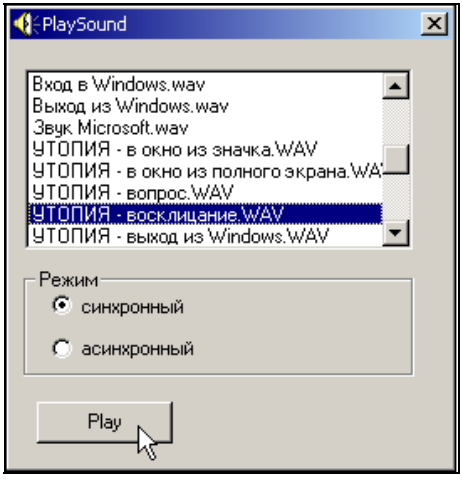


Рис. 6.1. Окно программы, демонстрирующей использование функции `PlaySound`

Листинг 6.1. Использование API-функции `PlaySound`

```
Option Explicit

' чтобы получить доступ к API-функции, функцию надо объявить
Private Declare Function PlaySound Lib "winmm.dll" Alias
"PlaySoundA" _
            (ByVal lpszName As String, ByVal hModule As Long, _
             ByVal dwFlags As Long) As Long

Const SND_ASYNC = &H1       ' асинхронный режим воспроизведения
                            ' звука
```

```
Const SND_SYNC = &H0       ' синхронный режим воспроизведения
                           ' звука
Const SND_NODEFAULT = &H2 ' не воспроизводить "Стандартный
                           ' звук"

Private Sub Form_Initialize()
    ' функция Environ с параметром "windir" возвращает
    ' название каталога, в который установлен Windows
     File1.Path = Environ("windir") + "\Media"
     PlaySound "УТОПИЯ - открыть", 0, SND_ASYNC +
SND_NODEFAULT
End Sub

' щелчок на кнопке Play
Private Sub Command1_Click()
    Dim aWAVFile As String ' файл, который надо воспроизвести

    aWAVFile = File1.Path + "\" + File1.FileName
    Command1.Enabled = False
    ' кнопка Play будет доступна, когда функция
    ' PlaySound завершит работу
    If Option1.Value Then
        ' режим синхронного воспроизведения
        '(ждать окончания воспроизведения)
        PlaySound aWAVFile, 0, SND_SYNC
    Else
        ' режим асинхронного воспроизведения
        '(ждать окончания воспроизведения не надо)
        PlaySound aWAVFile, 0, SND_ASYNC
    End If
    Command1.Enabled = True
End Sub
```

```
' завершение работы программы сопровождается звуком
Private Sub Form_Unload(Cancel As Integer)
PlaySound "УТОПИЯ - свернуть.wav", 0, SND_SYNC +
SND_NODEFAULT
End Sub
```

Компонент *MMControl*

Компонент `MMControl` обеспечивает воспроизведение звуковых (WAV, MID, RMI, MP3) и видео- (AVI) файлов а также CD.

Рис. 6.2. Значок компонента `MMControl`

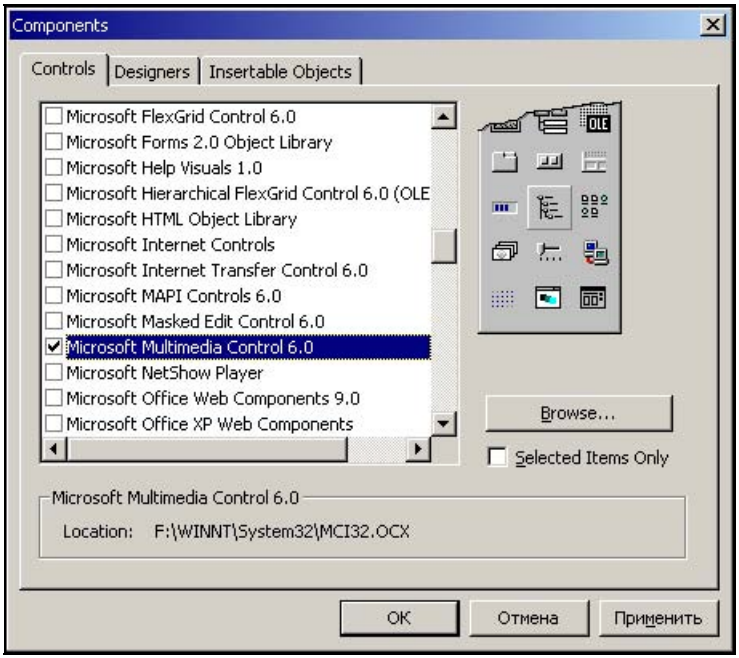


Рис. 6.3. Подключение компонента `MMControl`

Чтобы компонент (рис. 6.2) был доступен, его надо подключить: в меню **Project** выбрать команду **Components**, на вкладке **Controls** окна **Components** установить переключатель **Microsoft Multimedia Control** (рис. 6.3) и щелкнуть на кнопке **OK**.

Компонент `MMControl` представляет собой группу кнопок (рис. 6.4), подобных тем, которые можно видеть на обычном аудио- или видеоплеере. Назначение кнопок поясняет табл. 6.1. Свойства компонента `MMControl`, доступные во время разработки формы, приведены в табл. 6.2. Помимо свойств, перечисленных в таблице, для каждой из кнопок определены свойства `Enable` и `Visible`. Свойство `Enable` позволяет управлять доступом к кнопке, а свойство `Visible` — скрыть кнопку или сделать ее видимой.

Рис. 6.4. Компонент `MMControl`

Таблица 6.1. *Кнопки компонента* `MMControl`

Кнопка	Обозначение	Действие
Воспроизведение	**Play**	Воспроизведение звука или видео
Пауза	**Pause**	Приостановка воспроизведения
Стоп	**Stop**	Остановка воспроизведения
Следующий	**Next**	Переход к следующему кадру
Предыдущий	**Prev**	Переход к предыдущему кадру
Шаг	**Step**	Переход к следующему звуковому фрагменту, например, к следующей песне на CD

Таблица 6.1 *(окончание)*

Кнопка	Обозначение	Действие
Назад	**Back**	Переход к предыдущему звуковому фрагменту, например, к предыдущей песне на CD
Запись	**Record**	Активизирует процесс записи
Открыть	**Eject**	Открывает CD-дисковод компьютера

Таблица 6.2. *Свойства компонента* `MMControl`

Свойство	Описание
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент `MMControl`. Тип устройства задается строковой константой: • `WaveAudio` — проигрыватель звука; • `AVIVideo` — видеопроигрыватель; • `CDAudio` — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
`UpdateInterval`	Интервал генерации события `StatusUpdate`, которое обычно используется для организации обновления индикатора состояния устройства воспроизведения. Например, при воспроизведении CD номера воспроизводимого трека (композиции) и времени воспроизведения

Следует обратить внимание на то, что некоторые свойства (табл. 6.3) компонента `MMControl` доступны только во время работы программы и поэтому в окне **Properties** не отображаются.

Таблица 6.3. *Свойства компонента* `MMControl`*, доступные во время работы программы*

Свойство	Описание
`Length`	Длина (время, необходимое для воспроизведения) открытого файла (например, WAV или AVI) или всех треков AudioCD
`Tracks`	Количество треков на открытом устройстве (количество композиций на AudioCD)
`Position`	Позиция — время, прошедшее от начала воспроизведения файла или CD
`TrackPosition`	Позиция (время) трека (номер трека задает свойство `Track`), отсчитанное от начала CD
`TrackLength`	Длина (время, необходимое для воспроизведения) трека AudioCD
`TimeFormat`	Формат представления значений свойств `Length`, `Position` `TrackLength` и `TrackPosition`. Наиболее универсальным является формат `mciFormatMilliseconds`
`Mode`	Состояние устройства воспроизведения. Устройство может быть в состоянии воспроизведения (`msiModePlay`). Процесс воспроизведения может быть остановлен (`msiModeStop`) или приостановлен (`msiModePause`)
`HWndDisplay`	Идентификатор объекта (компонента), используемого в качестве экрана, на поверхности которого осуществляется отображение видеоклипа. Если значение свойства не задано, то отображение осуществляется в отдельном, создаваемом во время работы программы, окне

Управлять процессом работы устройства воспроизведения (компонентом `MMControl`) может пользователь (при помощи командных кнопок) или программа (путем записи соответствующих команд (табл. 6.4) в свойство `Command`). Следует обратить внимание на то, что в конце работы программы, которая ис-

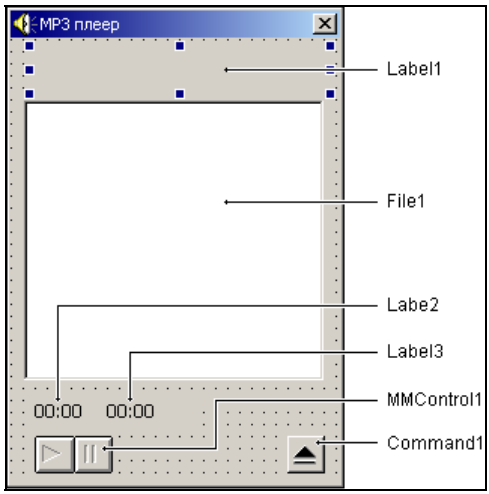
пользует компонент MMControl, устройство воспроизведения обязательно надо отключить (закрыть) при помощи команды **Close**.

Таблица 6.4. *Команды управления компонентом* `MMControl`

Команда	Действие
Open	Инициализирует мультимедиаустройство
Play	Активизирует процесс воспроизведения. Действие метода аналогично щелчку на кнопке **Play**
Stop	Останавливает процесс воспроизведения
Pause	Приостанавливает процесс воспроизведения
Next	Переход к следующему треку, например, к следующей композиции на AudioCD
Prev	Переход к предыдущему треку, например, к следующей композиции на AudioCD
Step	Переход к следующему кадру
Back	Переход к предыдущему кадру

Воспроизведение звука

В качестве примера использования компонента MMControl для воспроизведения звука рассмотрим программу, с помощью которой можно прослушать звуковые (MP3 и WAV) файлы. Форма и окно программы MP3 плеер представлены на рис. 6.5. Компонент File1 (FileList) используется для выбора файла. Во время работы программы в поле компонента Label1 отображается имя воспроизводимого файла, в поле Label2 — время звучания композиции, а в поле Label3 — время от начала воспроизведения композиции. Кнопка Command1 активизирует отображение стандартного диалога **Выбор папки**, в котором пользователь может выбрать каталог, содержащий звуковые файлы. Текст программы приведен в листинге 6.2.



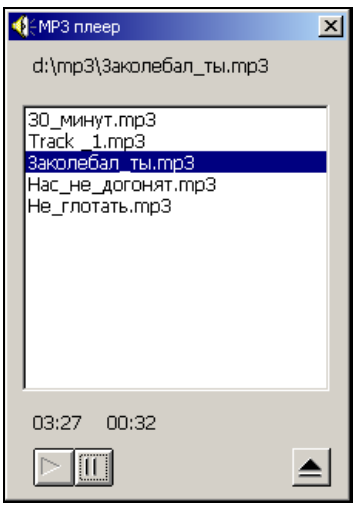


Рис. 6.5. Форма и окно программы MP3 плеер

Листинг 6.2. Программа MP3 плеер на основе компонента `MMControl`

```
Option Explicit
' для воспроизведения звуковых файлов в программе используется
```

```
' компонент Microsoft Multimedia Control (MMControl)
' *****************
       ' эти API-функции используются для доступа
       ' к окну "Выбор папки"
Private Declare Function SHBrowseForFolder Lib "shell32"
Alias "SHBrowseForFolderA" (ByRef b As Any) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32"
(ByVal ResPIDL As Any, ByVal patch As String) As Long

      ' константы SHELL API
Const CSIDL_DRIVES As Long = 17
Const BIF_RETURNONLYFSDIRS As Long = 1
Const MAX_PATH = 260

Const mciTormatMilleseconds = 0

     ' эта структура используется для передачи информации
     ' в функцию SHBrowseFolder, которая выводит диалоговое
     ' окно "Выбор папки"
Private Type bi ' browseinfo
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String ' выбранная папка (без пути)
    lpzTitle As String       ' подсказка
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type
'*******************

         ' инициализация формы
Private Sub Form_Initialize()
```

```
    On Error Resume Next
          ' папка, в которой находятся MP3-файлы
    File1.Path = "d:\mp3"

          ' в поле компонента File1 отображаются файлы
          ' с расширением mp3
    File1.Pattern = "*.mp3"

End Sub

          ' щелчок на имени файла в поле компонента File1
Private Sub File1_Click()

    If MMControl1.Command <> "CLOSE" _
    Then MMControl1.Command = "CLOSE"

    Label1.Caption = File1.Path + "\" + File1.FileName
    If File1.FileName <> "" Then
       MMControl1.FileName = File1.Path + "\" + File1.FileName

       MMControl1.Notify = True

       MMControl1.Command = "OPEN"
          ' MsgBox Str(MMControl1.Error) + ":"
          ' + MMControl1.ErrorMessage
       MMControl1.TimeFormat = mciTormatMilleseconds
       Label2.Caption = toHMS(MMControl1.Length)
       Label3.Caption = "00:00"
    End If
End Sub


          ' обработка события StatusUpdate,
          ' генерируемого MMControl
```

```
Private Sub MMControl1_StatusUpdate()
    Label3.Caption = toHMS(MMControl1.Position)
End Sub

        ' функция преобразует целое (время звучания трека,
        ' диска или текущую позицию воспроизводимого трека)
        ' в формат "часы:минуты:секунды"
Private Function toHMS(time As Long) As String
    Dim H As Integer      ' часы
    Dim M As Integer      ' минуты
    Dim S As Integer      ' секунды

    H = Int((time / 1000) / 3600)
    M = Int((time / 1000 Mod 3600) / 60)
    S = (time / 1000 Mod 3600) Mod 60

    If H > 0 Then toHMS = Str(H) + ":"
    toHMS = toHMS + Format(M, "0#") + ":" + Format(S, "0#")
End Function
                        ' щелчок на кнопке Выбор каталога
Private Sub Command1_Click()

    Dim t As bi           ' информация, которая
                          ' передается функции
                          ' SHBrowseFoFolder, обеспечивающей
                          ' отображение окна Выбор папки
    Dim ResPIDL As Long   ' значение функции SHBrowseFoFolder
    Dim Path As String    ' полное (включая путь) имя
                          ' выбранной папки

    Dim r As Long         ' значение функции
                          SHGetPathFromIDList
    Dim p As String       ' указатель на NUL-символ
```

```
    t.hwndOwner = Form1.hWnd
    t.lpzTitle = "Выберите папку, в которой находятся
                  MP3- или WAV-файлы"
    t.ulFlags = BIF_RETURNONLYFSDIRS      ' кнопка OK
                                ' доступна, если пользователь
                                ' выбрал папку
    t.pszDisplayName = String(MAX_PATH, 0)

    ResPIDL = SHBrowseForFolder(t) ' вывести окно Выбор папки

     If ResPIDL <> 0 Then
        ' пользователь выбрал папку
        ' получить ее полное имя
        Path = String(MAX_PATH, Chr(0))
        r = SHGetPathFromIDList(ResPIDL, Path)
        ' строка Path - это Си-строка, содержащая
        ' NULL-символ (код 0). До этого символа находится
        ' полезная информация, после - мусор.
        p = InStr(1, Path, Chr(0)) ' положение NULL-символа
        Path = Mid(Path, 1, p - 1)
        If Mid(Path, p - 1, 1) <> "\" Then Path = Path + "\"

        File1.Path = Path ' отобразить содержимое
                          ' выбранного каталога
                          ' в поле компонента FileList1
        Label1.Caption = ""
    End If

End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
```

```
    MMControl1.Command = "STOP"    ' остановить
                                   ' воспроизведение
    MMControl1.Command = "CLOSE"   ' "выключить" плеер
End Sub
```

Рассмотрим, как работает программа. Сразу после запуска функция обработки события `Initialaze` присваивает значения свойствам `Path` и `Pattern` компонента `File1` (`FileList`), в результате чего в поле компонента отображается список звуковых файлов, находящихся в каталоге D:\Mp3. Для обработки ошибки, которая возникает, если указанного каталога нет, в программу вставлена инструкция **`On Error`** `Resume` **`Next`** (при возникновении ошибки выполнить следующую инструкцию). Если каталога Mp3 на диске D: нет, то в поле компонента `File1` отображается список звуковых файлов, которые находятся в *текущем* каталоге (т. е. в каталоге, из которого запущена программа).

Также процедура инициализации формы скрывает ненужные кнопки компонента `MMControl`. В начале работы программы кнопка **Play** не доступна. Чтобы активизировать процесс воспроизведения, пользователь должен сначала сделать щелчок на имени файла в поле компонента `File1` (в результате кнопка **Play** становится доступной), затем — на кнопке **Play**. Процедура обработки события `Click` в поле компонента `File1` останавливает воспроизведение текущего файла, открывает выбранный файл и отображает время, необходимое для воспроизведения выбранного файла, в поле `Label2`. Для преобразования времени воспроизведения (свойство `Length` компонента `MMControl`), которое измеряется в миллисекундах, в формат МИН:СЕК используется специально созданная функция `toHMS`.

Щелчок на кнопке **Play** активизирует процесс воспроизведения. Во время воспроизведения компонент `MMControl` генерирует событие `StatusUpdate` (период возникновения события задает значение свойства `UpdateInterval`). Процедура обработки этого события выводит в поле `Label3` значение свойства `Position` — время, прошедшее от начала воспроизведения файла.

MIDI

В компьютерных играх, в обучающих и других программах, а также в качестве звуковых сигналов мобильных телефонов широко используется электронная (или MIDI) музыка. Название MIDI электронная музыка получила от названия интерфейса (Musical Instrument Digital Interface) — способа подключения электронных музыкальных инструментов к компьютеру. Находится электронная музыка в MID- и RMI-файлах.

Следующая программа (ее форма и окно приведены на рис. 6.6 и рис. 6.7) демонстрирует процесс воспроизведения MIDI-музыки при помощи компонента `MMControl`. Музыка (мелодия, находящаяся в MID-файле) начинает звучать сразу после запуска программы.

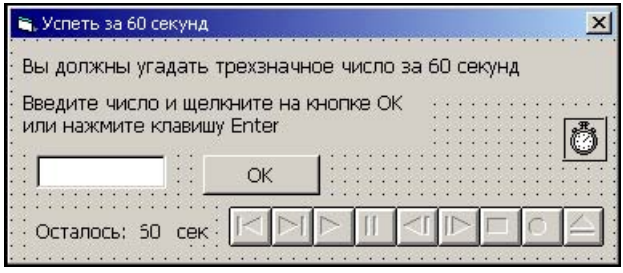


Рис. 6.6. Форма программы, демонстрирующей процесс воспроизведения MIDI-музыки

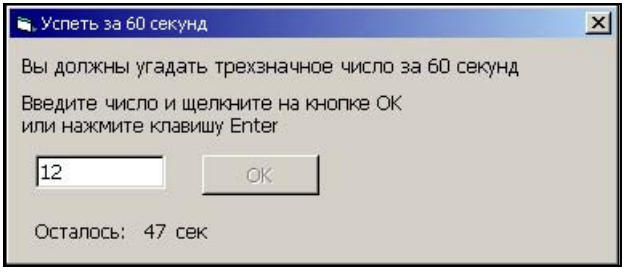


Рис. 6.7. Во время работы программы звучит MIDI-музыка

Как можно заметить, кнопки управления компонента MMControl в окне программы не отображаются (значение свойства Visible компонента MMControl равно **False**). Управление воспроизведением звука (работой медиаплеера) осуществляется программно — путем записи соответствующих команд в свойство Command.

Текст программы приведен в листинге 6.3.

Листинг 6.3. Программа Успеть за 60 секунд

```
Const LC = 3    ' количество цифр в числе, которое надо
                ' угадать
Dim sek As Integer    ' счетчик времени (обратный отсчет)

' настройка формы и компонентов
Private Sub Form_Load()
    MMControl1.Visible = False
    MMControl1.FileName = Dir("*.mid")
    If MMControl1.FileName = "" Then
        ' В текущем каталоге нет ни одного MIDI-файла.
        ' Попробуем загрузить из Windows\Media.
       MMControl1.FileName = Environ("windir") + "\Media\" + _
       Dir(Environ("windir") + "\Media\*.mid")
    End If

    ' музыкальное сопровождение - MIDI-мелодия
    MMControl1.Command = "Open"
    MMControl1.Command = "Play"

    sek = 60 ' время на решение задачи - 60 сек

    Command1.Enabled = False
    Text1.MaxLength = LC
End Sub
```

```
Private Sub Form_Activate()
    Text1.SetFocus ' установить курсор в поле редактирования
End Sub

' сигнал от MMControl
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position = MMControl1.Length Then
        ' воспроизведение мелодии завершено - повторить
        MMControl1.Command = "Prev" ' переход к началу
                                    ' мелодии
        MMControl1.Command = "Play" ' воспроизведение
    End If
End Sub

' нажатие клавиши в поле ввода числа
Private Sub Text1_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 48 To 57    ' цифры
        Case 8           ' <Backspace> - "забой"
        Case 13          ' <Enter>
                         ' проверить, правильное ли число
            Call IsItOk
        Case Else
                         ' остальные символы запрещены
              KeyAscii = 0
     End Select
End Sub

' содержимое поля ввода изменилось
Private Sub Text1_Change()
    ' кнопка OK доступна только в том случае,
    ' если в поле редактирования LC символов
    If Len(Text1.Text) = LC Then
        Command1.Enabled = True
```

```
    Else
        Command1.Enabled = False
    End If
End Sub

' щелчок на кнопке OK
Private Sub Command1_Click()
    Call IsItOk
End Sub

' проверяет, угадал ли игрок число
Sub IsItOk()
    If Text1.Text = "123" Then
        Timer1.Enabled = False ' остановить таймер
        Command1.Enabled = False
        Text1.Enabled = False
        MsgBox "Поздравляю! Вы справились с поставленной_
        задачей за " + Str(60 - sek) + "сек.", vbOKOnly,_
        "Успеть за 60 секунд"
    Else
        MsgBox "НЕ ПРАВИЛЬНО. Осталось " + Str(sek) + _
               "сек.", vbOKOnly, "Успеть за 60 секунд"
        Text1.SetFocus
    End If

End Sub

' сигнал от таймера
Private Sub Timer1_Timer()
    sek = sek - 1
    Label1.Caption = Str(sek)
    If sek = 0 Then
        ' время, отведенное на решение задачи, истекло
        Timer1.Enabled = False
```

```
        Command1.Enabled = False
        Text1.Enabled = False
        MsgBox "К сожалению Вы не справились с поставленной_
        задачей." + vbCr, vbOKOnly, "Успеть за 60 секунд"
    End If
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"  ' остановит воспроизведение
    MMControl1.Command = "Close" ' "выключить" плеер
End Sub
```

Работает программа следующим образом. Процедура обработки события `Load` записывает в свойство `FileName` компонента `MMControl` имя одного из MIDI-файлов, который находится в текущем каталоге. Имя файла возвращает функция `Dir`. Если в текущем каталоге нет ни одного файла с расширением mid, то делается попытка найти MIDI-файл в стандартном каталоге Media, который находится в главном каталоге Windows (здесь Windows — название операционной системы). Так как главный каталог Windows на разных компьютерах может называться по-разному (например, Windows или Winnt), то вместо конкретного имени используется значение функции `Environ` с параметром `windir`. Значением функции `Environ` (от Environment — окружение) с указанным параметром является значение переменной окружения `windir`, которая содержит имя главного каталога Windows, (например, C:\Winnt).

После этого медиаплееру направляются команды `Open` и `Play`. В результате начинает звучать музыка, загруженная из MIDI-файла. В процессе воспроизведения медиаплеер генерирует событие `StatusUpdate`. Процедура обработки этого события сравнивает положение указателя текущей позиции воспроизводимого файла со значением свойства `Length` (длиной — временем, необходимым для воспроизведения) и, если значения равны, отправляет медиаплееру команды `Prev` (перевести указатель

воспроизведения в начало) и `Play`. В результате мелодия начинает звучать снова. Таким образом, мелодия звучит до тех пор, пока окно программы не будет закрыто. Остальные процедуры реализуют алгоритм игры.

Процедура обработки события `KeyPress` компонента `Text` блокирует ввод в поле редактирования запрещенных символов.

Процедура обработки события `Change` этого же компонента делает недоступной кнопку **OK**, если в поле редактирования находится менее трех символов.

Процедура обработки события от таймера обеспечивает обратный отсчет и индикацию времени. Следует обратить внимание на то, что процедура обработки события `Unload` формы останавливает процесс воспроизведения и закрывает плеер.

CD-плеер

Следующий пример показывает, как на основе компонента `MMControl` можно создать вполне приличный проигрыватель компакт-дисков. Вид формы программы приведен на рис. 6.8, значения свойств компонета `MMContol1` — в табл. 6.5. Компонент `Timer1` используется для управления работой "индикатора", обеспечивает мигание сообщения о необходимости вставить в дисковод музыкальный CD.

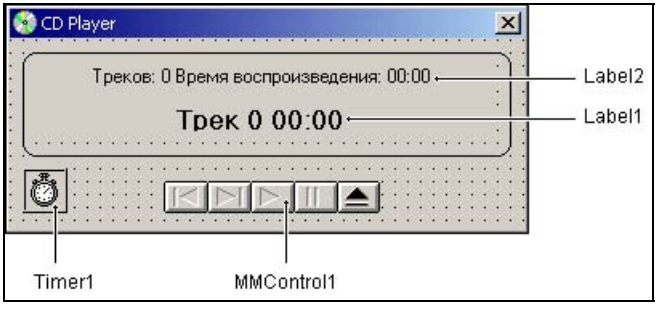


Рис. 6.8. Форма программы CD Player

Таблица 6.5. *Значения свойств компонента* `MMControl1`

Свойство	Значение
`DeviceType`	`CDAudio`
`BackVisible`	`False`
`StepVisible`	`False`
`StopVisible`	`False`
`RecordVisible`	`False`
`BorderStyle`	0 — `msiNone`
`UpdateInterval`	1000

Вид окна программы сразу после ее запуска приведен на рис. 6.9.

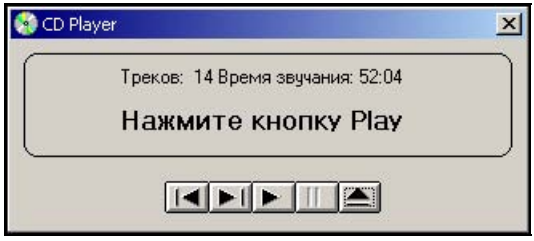


Рис. 6.9. В начале работы программы на индикаторе выводится информация о количестве треков (композиций) и времени воспроизведения CD

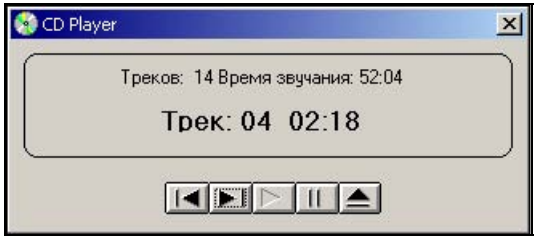


Рис. 6.10. Во время воспроизведения на индикаторе отображается информация о воспроизводимом треке

В случае если диска в дисководе нет или диск не музыкальный, на индикаторе выводится соответствующее сообщение. Щелчок на кнопке **Play** активизирует процесс воспроизведения. Во время воспроизведения на индикаторе отражается номер воспроизводимого трека и время, прошедшее от начала его воспроизведения (рис. 6.10).

Текст программы приведен в листинге 6.4. Основную работу в программе выполняет процедура обработки события `StatusUpdate`, которое генерирует компонент `MMControl` (период возникновения события определяет значение свойства `UpdateInterval`). Процедура (в зависимости от состояния плеера) выводит информацию о диске, воспроизводимом треке или сообщение о необходимости вставить в дисковод музыкальный CD. Информация о воспроизводимом треке выводится, если плеер находится в режиме воспроизведения (значение свойства `Mode` равно 526).

Здесь следует обратить внимание на то, что свойство `Position` содержит позицию (время) от начала диска, а не от начала трека. Поэтому время, прошедшее от начала воспроизведения трека, вычисляется как разность значений свойств `Position` и `TrackPosition` (положение трека на диске тоже задается от его начала).

Также следует обратить внимание на то, что значение свойства `Track` при переходе к следующему треку автоматически не увеличивается. Поэтому если значение `Position` выходит за границу текущего трека (`TrackPosition + TackLength`), то процедура увеличивает значение свойства `Track` или (если значение `Position` больше чем `Length`) останавливает процесс воспроизведения.

Если диска в дисководе нет (в этом случае плеер находится в состоянии `Ready`) или если диск не музыкальный (плеер находится в состоянии `Stop` и количество треков на диске равно 1), то процедура обработки события `StatusUpdate` запускает таймер. Процедура обработки сигнала от таймера инвертирует (изменяет на привоположное) значение свойства `Visible` компонента `Label1`, в результате чего текст, находящийся в поле `Label1`, мигает с периодом 2 · `Timer1.Interval`.

И последнее, на что надо обратить внимание: процедура обработки события Unloud, которое возникает в момент завершения работы программы, останавливает процесс воспризведения и закрывает плеер.

Листинг 6.4. Проигрыватель компакт-дисков

```
Option Explicit

' начало работы программы
Private Sub Form_Load()
    MMControl1.DeviceType = "CDAudio"
    MMControl1.Command = "Open"
    MMControl1.TimeFormat = 0 ' формат счета времени -
                              ' миллисекунды
End Sub

' изменился статус CD, например, в результате щелчка
' на кнопке управления
Private Sub MMControl1_StatusUpdate()
    Select Case MMControl1.Mode
        Case 525: ' msiModeStop
           ' Label1.Caption = "Stop" & Str(MMControl1.Tracks)
            If MMControl1.Tracks > 1 Then
                Label1.Caption = "Нажмите кнопку Play"
                Label2.Caption = _
                    "Треков: " + Str(MMControl1.Tracks) + _
                    " Время звучания: " +_
                    toHMS(MMControl1.Length)
            Else
                Label1.Caption = "Диск не музыкальный!"
                Timer1.Enabled = True
            End If
```

```
        Case 526: ' msiModePlay
            Timer1.Enabled = False
            Label1.Visible = True
            Label2.ForeColor = &H80000012  ' системный цвет
                                           ' Button Text

            If MMControl1.Position < _
                   MMControl1.TrackPosition +_
                   MMControl1.TrackLength Then
                ' вывести информацию о воспроизводимом треке
                Label1.Caption = _
                "Трек: " + Format(MMControl1.Track, "0#")_
                 + "  " + toHMS(MMControl1.Position - _
                 MMControl1.TrackPosition)
            Else
                ' воспроизведение трека закончено
               If MMControl1.Position < MMControl1.Length
Then
                    ' переход к следующему треку
                    MMControl1.Track = MMControl1.Track + 1
                    Label1.Caption = _
                    "Трек: " + Format(MMControl1.Track, "#")_
                    + "  " + toHMS(MMControl1.Position - _
                    MMControl1.TrackPosition)
               Else
                    MMControl1.Command = "Stop"
               End If
            End If

        Case 529: ' msiModePause:\
            Label1.Caption = "Pause"
            Timer1.Enabled = True

        Case 530: ' msiModeReady
            Label1.Caption = "В дисководе нет диска!"
```

```
            Label2.Caption = "Треков: 0 Время звучания: 0:00"
            Label2.ForeColor = &H80000011   ' системный цвет_
                                            ' Disabled Text
            Timer1.Enabled = True
        Case Else
            Label1.Caption = "Mode:" & Str(MMControl1.Mode)
    End Select

End Sub

' таймер обеспечивает мигание индикатора
Private Sub Timer1_Timer()
    DoEvents
    Label1.Visible = Not Label1.Visible
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop" 'остановить воспроизведение
CD
    MMControl1.Command = "Close"
End Sub

' функция преобразует целое (время звучания трека, диска_
' или текущую позицию воспроизводимого трека)_
' в формат "часы:минуты:секунды"
Private Function toHMS(time As Long) As String
    Dim H As Integer     ' часы
    Dim M As Integer     ' минуты
    Dim S As Integer     ' секунды

    H = Int((time / 1000) / 3600)
    M = Int((time / 1000 Mod 3600) / 60)
    S = (time / 1000 Mod 3600) Mod 60
```

```
    If H > 0 Then toHMS = Str(H) + ":"
    toHMS = toHMS + Format(M, "0#") + ":" + Format(S, "0#")
End Function
```

Регулятор громкости

Наиболее просто можно установить требуемую громкость звука при помощи стандартного регулятора громкости (рис. 6.11). Но можно регулировать громкость звука и прямо из программы. Рассмотрим, как можно регулировать громкость воспроизведения MP3-файла (громкость воспроизведения WAV-файла регулируется аналогично).

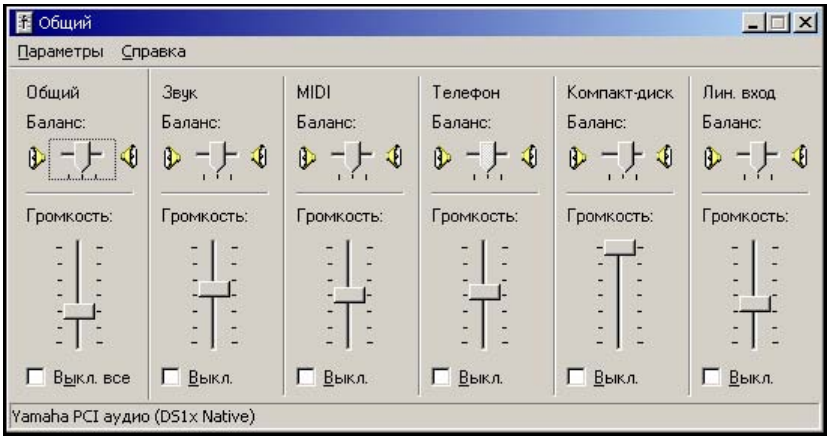


Рис. 6.11. Стандартный регулятор громкости

Задать необходимую громкость воспроизведения WAV-файла можно при помощи API-функции `waveOutSetVolume`. Инструкция вызова функции в общем виде выглядит так:

```
r = waveOutSetVolume(ИдентификаторУстройства, Громкость)
```

Параметр `ИдентификаторУстройства` задает устройство воспроизведения, громкость которого надо установить. При регулировке

громкости воспроизведения WAV-файла значение этого параметра должно быть равно нулю.

Параметр `Громкость` (двойное слово — 4 байта) задает громкость воспроизведения: младшее слово определяет громкость левого канала, старшее — правого. Максимальной громкости звучания канала соответствует шестнадцатеричное значение `FFFF`, минимальной — `0000`. Таким образом, чтобы установить максимальную громкость воспроизведения в обоих каналах, значение параметра `Громкость` должно быть `&HFFFFFFFF` (при записи шестнадцатеричных констант в Visual Basic используется префикс `&H`). Уровню громкости 50% соответствует константа `&h7FFF7FFF`.

Необходимо обратить внимание на то, что функция `waveOutSetVolume` регулирует громкость воспроизведения *звукового канала*, а не общий уровень звука (рис. 6.12).

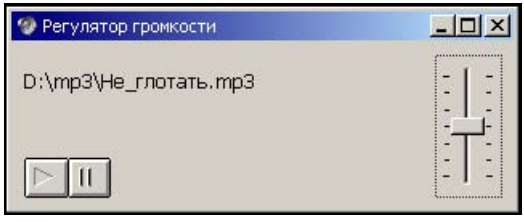


Рис. 6.12. Регулятор громкости

Следующая программа (ее окно приведено на рис. 6.13) демонстрирует, как можно регулировать громкость воспроизведения MP3-файла.

Изменение громкости осуществляется при помощи компонента `Slider` (чтобы этот компонент был доступен, в проект надо поместить ссылку на модуль компонентов Microsoft Windows Common Controls-2 6.0). Значения свойств компонента `Slider` приведены в табл. 6.6.

Следует обратить внимание: при вертикальном расположении компонента минимальному значению свойства `Value` соответствует верхнее положение движка, максимальному — нижнее. Поэтому диапазон изменения значения свойства `Value` задан от

−65 535 до 0. И таким образом при нулевом значении свойства `Value` движок регулятора находится в нижнем, а не в верхнем положении.

Текст программы приведен в листинге. 6.5.

Таблица 6.6. *Значения свойств компонента* `Slider`

Свойство	Значение
`Orientation`	1 — `ccOrientationVertical`
`Height`	89
`Width`	42
`Min`	–65 535
`Max`	0
`TickFrequency`	10 992
`SmallChange`	256
`LageChange`	8192

Листинг 6.5. Регулировка громкости звука

```
Option Explicit

' Громкость звучания устанавливается при помощи
' API-функции waveOutSetVolume
Private Declare Function waveOutSetVolume Lib "winmm.dll" _
    (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

' начало работы программы
Private Sub Form_Load()
    Dim v As Long ' громкость канала
    Dim st As String * 8

    Form1.ScaleMode = vbPixels
    ' настройка компонента Slider
```

```
    With Slider1
      .Min = -65535
      .Max = -0
      .SmallChange = 256
      .LargeChange = 8192
          ' чтобы регулятор выглядел как в окне_
          ' Регулятор громкости
      .Height = 89
      .Width = 42
      .TickFrequency = 10992
    End With

       ' установить громкость - приблизительно 30%_
       ' от максимальной
    v = &H2FFF
    st = Hex(v) & Hex(v)
    ' переменная st содержит строковое представление_
    ' двойного слова
    waveOutSetVolume 0, "&H" & st

    Slider1.Value = -v ' переместить движок в положение,_
       ' соответствующее установленному уровню громкости

    MMControl1.FileName = "D:\mp3\" & Dir("d:\mp3\*.mp3")
    Label1.Caption = MMControl1.FileName
    MMControl1.Command = "OPEN"
End Sub

' событие Scroll генерируется в процессе перемещения движка_
' мышью и, как результат перемещения движка, клавишами_
' перемещения курсора и клавишами PgUp и PgDown
Private Sub Slider1_Scroll()
    Dim v As Long     ' громкость одного канала
    Dim st As String ' строковое представление двойного слова
```

```
    v = Abs(Slider1.Value)
    st = "000" + Hex(v)
    st = Right(st, 4)
    st = st + st

    waveOutSetVolume 0, "&H" + st
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

Непосредственное изменение громкости осуществляет процедура обработки события `Scroll` регулятора громкости (компонента `Slider`), которое периодически генерируется в процессе перемещения движка регулятора мышью или клавишами <PgUp>, <PgDown> и клавишами перемещения курсора вверх или вниз. Процедура обработки этого события сначала преобразует значение свойства `Value` в шестнадцатеричную четырехразрядную строковую константу, затем формирует шестнадцатеричную восьмиразрядную строковую константу, первые четыре разряда которой определяют громкость правого канала, вторые — левого. Сформированная строковая константа передается функции `waveOutSetVolume`. Использование такого "хитрого" формирования параметра функции `waveOutSetVolume` объясняется следующим: параметр `Громкость` должен быть беззнаковым двойным словом. Если бы в Visual Basic такой тип был, то в программе можно было бы записать так:

```
Dim vl as Word
Dim vl as Word
Dim vl as DoubleWord
vl = &H7777
vr =&H7777
v = vl or vr
waveOutSetVolume 0, v
```

Но в Visual Basic такого типа нет. Наиболее близким (на первый взгляд) является тип `Long` (4 байта), но он позволяет корректно изменять в пределах всего диапазона громкость только левого канала (два младших байта). Громкость же правого канала (два старших байта) при использовании этого типа можно менять только в диапазоне от 00000 до 7FFF, т. к. старший двоичный разряд числа типа `Long` является знаковым. Попытка увеличить громкость правого канала на единицу (прибавить `&H10000` к числу типа `Long`, старшие четыре разряда которого содержат `&H7FFF`), приводит к изменению знака и к потере значения переменной, содержащей значение громкости.

MIDI

Регулировку громкости воспроизведения MIDI обеспечивает функция `midiOutSetVolume`. Параметры у этой функции такие же, как и у функции `waveOutSetVolume`, но есть одна особенность. В системе может быть несколько устройств MIDI. Какое из этих устройств используется для воспроизведения MIDI в данный момент и, следовательно, громкость какого устройства должна регулировать программа, определяют настройки операционной системы. Поэтому использовать константу в качестве идентификатора устройства (первого параметра функции `midiOutSetVolume`) не рекомендуется, лучше определить идентификатор активного устройства во время запуска программы. Программа (ее окно приведено на рис. 6.13, а текст в листинге 6.5) показывает, как это сделать.

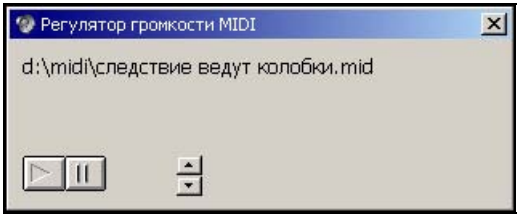


Рис. 6.13. Регулятор громкости MIDI

Листинг 6.6. Регулятор громкости MIDI

```
Option Explicit

' Громкость звучания регулируется при помощи API-функции
Private Declare Function midiOutSetVolume Lib "winmm.dll" _
    (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

' uDeviceID - идентификатор устройства, громкость которого_
' регулируется
' dwVolume - громкость (старшее слово задает громкость
' правого канала, младшее - левого)

Private Declare Function midiOutGetNumDevs Lib "winmm" ()
As Integer

Dim id As Long ' идентификатор MIDI-устройства, которое
               ' регулируем

Private Sub Form_Load()
    Dim n As Integer ' кол-во MIDI-устройств в системе
    Dim r As Long    ' рез-т выполнения API-функции
    Dim v As Long
    Dim i As Long

    ' В системе может быть несколько MIDI-устройств,
    ' выясним, какое из них активно. Для этого попробуем
    ' задать громкость.
    n = midiOutGetNumDevs
    For i = 0 To n - 1
        r = midiOutSetVolume(i, "&H00FF00FF")
        If r = 0 Then
            ' MsgBox "id=" & Str(i) & "r=" & Str(r)
```

```
            id = i
            Exit For
        End If
    Next i

    UpDown1.Min = 0
    UpDown1.Max = 65535 ' максимальный уровень
                        ' громкости - FFFF
    UpDown1.Increment = (UpDown1.Max - UpDown1.Min) / 20
    UpDown1.Value = UpDown1.Increment * 6       ' установить
                ' громкость приблизительно 30% от максимума

    MMControl1.FileName =  Dir("*.mid")
    If MMControl1.FileName <> "*.mid" Then
        Label2.Caption = MMControl1.FileName
            MMControl1.Command = "OPEN"
            MMControl1.Command = "Play"
    Else
        Label1.Caption = "В текущем каталоге нет MIDI-файлов"
    End If

End Sub

' событие Change возникает как реакция на изменение свойства
' Value, которое, в свою очередь, автоматически изменяется
' в результате щелчка ' на одной из кнопок компонента UpDown
Private Sub UpDown1_Change()
    Dim v As Currency
    Dim s As String
    Dim r As Long

    v = UpDown1.Value
'В качестве параметра value функции midiOutSetVolume надо
'передать двойное слово, но такого типа в VB нет.
```

```
'Поэтому, формируем строку (шестнадцатеричное восьмиразрядное
'число) и передаем ее функции midiOutSetVolume.

    s = Right("000" & Hex(v), 4)
    r = midiOutSetVolume(id, "&H" & s & s)
End Sub

' обработка сигнала StatusUpdate от MMControl
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position >= MMControl1.Length Then
        MMControl1.Command = "Stop"
        MMControl1.Command = "Close"
        MMControl1.FileName = Dir ' получить имя
                                  ' следующего MIDI-файла
        If MMControl1.FileName <> vbNull Then
            Label2.Caption = MMControl1.FileName
            MMControl1.Command = "Open"
            MMControl1.Command = "Play"
        Else
            ' получить имя MIDI-файла из текущего каталога
            MMControl1.FileName = Dir("*.mid")
            Label2.Caption = MMControl1.FileName
            MMControl1.Command = "Open"
            MMControl1.Command = "Play"
        End If
    End If
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "STOP"
    MMControl1.Command = "CLOSE"
End Sub
```

Рассмотрим процесс работы программы. Сразу после запуска процедура обработки события `Load` обращается к API-функции `midiOutGetNumDevs`, которая возвращает количество MIDI-устройств, установленных в системе. Затем делается попытка изменить громкость каждого из устройств и тем самым определить идентификатор активного устройства (если устройство активно, то значение функции `midiOutSetVolume` равно нулю). После этого выполняются настройка компонента `UpDown` и загрузка из текущего каталога MIDI-файла. Громкость регулирует процедура обработки события `Change` компонента `UpDown`. Следует обратить внимание: программа спроектирована таким образом, что после воспроизведения текущего MIDI-файла делается попытка загрузить следующую мелодию, и если это удается, то автоматически активизируется процесс воспроизведения. Если загрузить следующий файл не удается, то в этом случае значение функции `Dir` равно `vbNull`, функция `Dir` вызывается снова, но уже с параметром. В результате снова загружается первый из уже проигранных файлов. Таким образом MIDI-файлы текущего каталога проигрываются непрерывно.

Просмотр видеороликов

Как было сказано ранее, компонент `MMControl` позволяет просматривать видеоролики и сопровождаемую звуком анимацию. В качестве примера использования компонента для решения этой задачи рассмотрим программу Video плеер (рис. 6.14), при помощи которой можно просмотреть клип или анимацию, представленную в AVI-формате.

Вид формы программы приведен на рис. 6.15, значения свойств компонента `MMControl1` — в табл. 6.7. Компонент `Picture1` используется в качестве экрана, на поверхности которого отображается клип. Компонент `CommonDialog1` в данной программе представляет собой стандартное диалоговое окно **Открыть файл**, которое становится доступным в результате щелчка на кнопке **Eject** и используется для выбора AVI-файла. Здесь следует обратить внимание, что управление работой видеоплеера осуществляется не при помощи кнопок компонента `MMControl`, а при помощи командных кнопок `Play` и `Eject`, поэтому свойству `Visible` компонента `MMControl` присвоено значение `False`.

Рис. 6.14. Простой видеоплеер на основе компонента `MMControl`

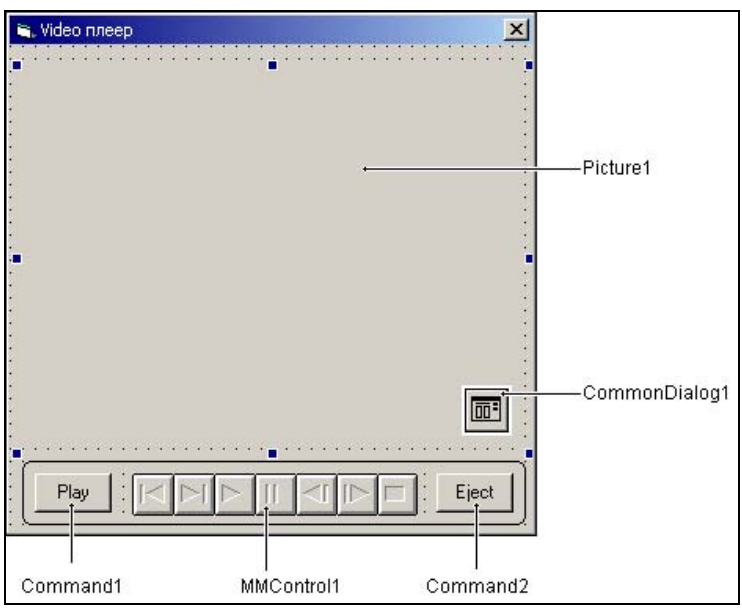


Рис. 6.15. Форма программы Video плеер

Таблица 6.7. *Значения свойств компонента* `MMControl`

Свойство	Значение	Коментарий
`DeviceType`	`AVIVideo`	
`hWndDisplay`	`Picture1.hWnd`	Свойство доступно только во время работы программы. Значение устанавливает процедура обработки события `Load`
`Visible`	`False`	Кнопки компонента не используются для управления работой плеера

Текст программы Video плеер приведен в листинге. 6.7. В начале своей работы программа настраивает компонент `MMControl`: присваивает значение свойству `hWhdDisplay` и делает попытку загрузить AVI-файл из текущего каталога. Для проверки существования файла используется функция `Dir`, которая возвращает имя файла, соответствующее маске, указанной в качестве параметра функции. Если в текущем каталоге (из которого запущена программа) есть файл с расширением avi, то компоненту `MMControl` направляются команды `Open` и `Next`, в результате чего в поле компонента `Picture1` появляется первый кадр видеоролика. Следует обратить внимание на то, что кнопка `Command1` используется как для активизации, так и для приостановки процесса воспроизведения. В зависимости от режима работы плеера на кнопке находится текст **Play** или **Pause**. Если процесс воспроизведения приостановлен, то щелчок на кнопке `Command1` активизирует процесс воспроизведения. Если плеер находится в режиме воспроизведения, то щелчок на кнопке `Command1` приостанавливает воспроизведение. В процессе воспроизведения компонент `MMControl` генерирует событие `StatusUpdate`. Процедура обработки этого события контролирует положение указателя текущей позиции и (если указатель достиг конца воспроизводимого ролика) останавливает процесс воспроизведения.

Листинг 6.7. Video плеер

```
Option Explicit

' эти константы используются для управления
' диалогом Открыть файл (Выбор AVI-файла)
Const cdlOFNHideReadOnly = &H4   ' не отображать
                                 ' CheckBox "Только чтение"
Const cdlOFNFileMustExist = &H1000 ' в поле Имя файла можно
                      ' ввести имя только существующего файла

' начало работы программы
Private Sub Form_Load()

    ' настройка MMControl
     MMControl1.hWndDisplay = Picture1.hWnd
    ' экран - компонент Picture1

    ' попробуем загрузить AVI-файл из текущего каталога
    ' MMControl1.FileName = Dir("*.avi")
    If MMControl1.FileName <> "" Then
        ' в текущем каталоге есть AVI-файл
        MMControl1.Command = "Open"
        MMControl1.Command = "Next" ' отобразить первый кадр
        Command1.Enabled = True ' кнопка Play доступна
    End If

    ' настройка CommonDialog
    CommonDialog1.DialogTitle = "Выбор AVI-файла"
    CommonDialog1.FileName = "*.avi"
    CommonDialog1.Flags = cdlOFNHideReadOnly +_
                          cdlOFNFileMustExist
End Sub
```

```
' Щелчок на кнопке Play/Pause
Private Sub Command1_Click()
    If (MMControl1.Mode = 525) Or _
       (MMControl1.Mode = 529) Then ' 525 - mciModeStop,
                                    ' 529 - mciModePause
                    ' плеер в режиме "Стоп" или "Пауза"
        MMControl1.Command = "Play"
        Command1.Caption = "Pause"
        Command2.Enabled = False
    Else
        ' плеер в режиме "Воспроизведение"
        If MMControl1.Mode = 526 Then ' 526 - mciModePlay
            MMControl1.Command = "Pause"
            Command1.Caption = "Play"
            Command2.Enabled = True
        End If
    End If
End Sub

' щелчок на кнопке Eject
' (выбор файла для просмотра)
Private Sub Command2_Click()
    CommonDialog1.FileName = "*.avi"
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "*.avi" Then
        ' пользователь выбрал файл
        MMControl1.Command = "Close"
        MMControl1.FileName = CommonDialog1.FileName
        MMControl1.Command = "Open"
        MMControl1.Command = "Next" ' отобразить первый кадр
        Command1.Enabled = True
    End If

End Sub
```

```
' обработка сигнала StatusUpdate от плеера
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position >= MMControl1.Length Then
        MMControl1.Command = "Stop"
        MMControl1.Command = "Prev"
        Command1.Caption = "Play"
        Command2.Enabled = True
    End If
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

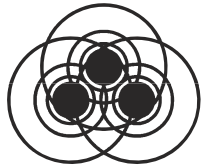
Установка программы на компьютере пользователя

Мультимидийную программу, созданную в Visual Basic, можно перенести на другой компьютер. Однако следует обратить внимание на следующий важный момент. Компонент `MMControl` — это `ActiveX` компонент (динамическая библиотека mci32.ocx). Вполне вероятно, что на компьютере пользователя этой библиотеки нет, поэтому программа работать не будет. Чтобы мультимедийная программа, использующая компонент `MMControl`, работала на другом компьютере, пользователю необходимо передать не только выполняемый (EXE) файл, но и файл mci32.ocx. Этот файл надо поместить в системный каталог операционной системы (C:\Windows\System, если на компьютере установлена операционная система Microsoft Windows 98, или в каталог C:\Windiws\System32, если на компьютере установлена другая операционная система семейства Microsoft Windows).

Причем недостаточно просто скопировать динамическую библиотеку в системный каталог — библиотеку надо *зарегистрировать* в системе. Для этого в окне **Запуск программы** (становится доступным в результате выбора команды **Пуск ▸ Выполнить**) надо набрать команду `regsvr mci32.ocx` (в Microsoft Windows 98) или `regsvr32 mci32.ocx` (в Microsoft Windows 2000/XP).

Если вы не уверены, что пользователь сможет самостоятельно выполнить установку программы, то лучше создать программу установки. О том, как это сделать, рассказывается в *главе 9*.

Глава 7



Базы данных

С точки зрения пользователя, *база данных* — это программа, которая обеспечивает работу с информацией. При запуске такой программы на экране, как правило, появляется таблица, просматривая которую можно найти нужные сведения. Если система позволяет, то пользователь может внести изменения в базу данных, например, добавить новую информацию или удалить ненужную.

С точки зрения программиста, *база данных* — это набор файлов, в которых находится информация. Разрабатывая *базу данных для пользователя*, программист создает программу, которая обеспечивает работу с файлами данных.

В состав Visual Basic включены компоненты, используя которые программист может создать программу работы практически с любой из существующих баз данных (dBase, Paradox, Microsoft Access, Microsoft SQL Server, Oracle).

База данных и СУБД

Программная система (приложение), обеспечивающая работу с базой данных (файлами данных), называется системой управления базой данных (СУБД). Следует обратить внимание на то, что вместо термина СУБД часто используется термин *база данных*, при этом файлы данных и СУБД рассматриваются как единое целое.

Локальные и удаленные базы данных

В зависимости от расположения базы данных и программы, которая манипулирует этими данными, различают *локальные* и *удаленные* базы данных.

В локальной базе файлы данных находятся на диске того компьютера, на котором работает программа манипулирования данными (СУБД). Локальные базы данных не обеспечивают одновременный доступ к информации нескольким пользователям. Несомненным достоинством локальной базы данных является высокая скорость доступа к информации. Microsoft Access, dBase, Paradox — это все локальные базы данных.

Данные удаленной базы размещают на отдельном компьютере, но доступ к которому доступен по сети. Программы работы с удаленными базами данных строят по технологии "клиент-сервер".

Клиентская часть СУБД (клиент) работает на компьютере пользователя. Она, взаимодействуя с программой-сервером, обеспечивает отображение данных, прием команд пользователя и передачу команд серверу. Серверная часть СУБД (сервер) работает на удаленном компьютере, принимает запросы (команды) от клиента, выполняет их и пересылает данные клиенту. Программа, работающая на удаленном компьютере, проектируется так, чтобы обеспечить одновременный доступ к базе данных многим пользователям. Microsoft SQL Server 2000 и Oracle — это удаленные базы данных.

Структура базы данных

База данных — это набор однородной информации (как правило, упорядоченной по некоторому критерию), она может быть представлена в "бумажном" или в электронном виде.

Типичным примером "бумажной" базы данных является каталог библиотеки — набор бумажных карточек, содержащих информацию о книгах. Информация в этой базе однородная (содержит сведения только о книгах) и упорядоченная (карточки обычно расставлены в алфавитном порядке фамилий авторов). Еще од-

ним примером "бумажной" базы данных является расписание движения поездов.

Электронная база данных представляет собой файл (или набор связанных файлов), содержащий какую-либо информацию. Этот файл часто называют *файлом данных*. Файл данных состоит из *записей*. Каждая запись содержит информацию только об одном *объекте*. Например, запись базы данных **Ежедневник** содержит информацию только об одном объекте — запланированном мероприятии или задаче.

Записи состоят из полей. Каждое поле содержит информацию об одной характеристике объекта. Например, запись базы данных **Ежедневник** может состоять из полей: **Задача**, **Дата** и **Примечание**. Содержимое полей характеризует конкретную задачу.

На бумаге базу данных удобно представить в виде таблицы. Каждая строка таблицы соответствует записи, а ячейка таблицы — полю. При этом заголовок столбца таблицы — это имя поля, а номер строки таблицы — номер записи.

Информацию компьютерных баз данных обычно выводят на экран в виде таблиц. Поэтому часто вместо словосочетания "файл данных" используют словосочетание "таблица данных" или просто "таблица".

Механизмы доступа к данным

В Microsoft Visual Basic 6.0 основной технологией доступа к данным является технология Microsoft ActiveX Data Objects (ADO — ActiveX объекты доступа к данным). Поддержка технологии ADO в Visual Basic реализована в виде набора компонентов, обеспечивающих доступ и манипулирование данными (Adodc, DataGrid и др.). Также программист может напрямую использовать ADO-объекты (`Connection`, `Command`, `Recordset` и др.).

Компоненты доступа и отображения данных

Доступ к базе данных обеспечивает компонент `Adodc`, отображение данных в виде таблицы — компонент `DataGrid` (рис. 7.1).

Чтобы эти компоненты стали доступными, их надо подключить: выбрать в меню **Project** (или в контекстном меню палитры компонентов) команду **Components** и установить во включенное состояние переключатели соответствующих этим компонентам модулей (рис. 7.2).

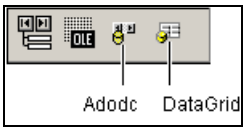


Рис. 7.1. Компонент `Adodc` обеспечивает доступ к данным, `DataGrid` — отображение данных в виде таблицы

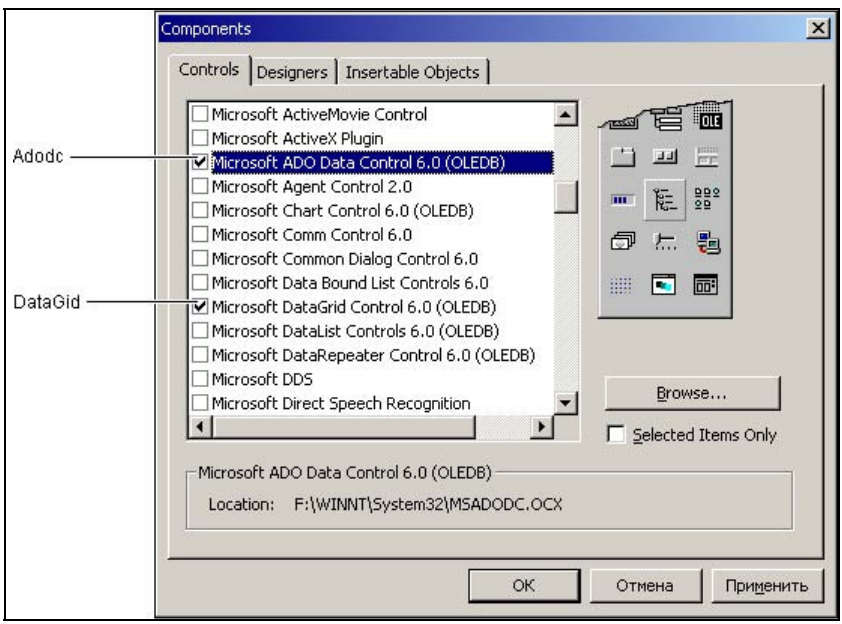


Рис. 7.2. Подключение компонентов `Adodc` и `DataGrid`

Следует обратить внимание на то, что компонент `Adodc` обращается к данным не напрямую, а через ядро баз данных Microsoft Jet и, в зависимости от вида строки соединения, через диспетчер

драйверов ODBC. Механизм доступа к данным (через диспетчер драйверов ODBC) и взаимодействие компонентов, обеспечивающих доступ и отображение данных, показан на рис. 7.3.

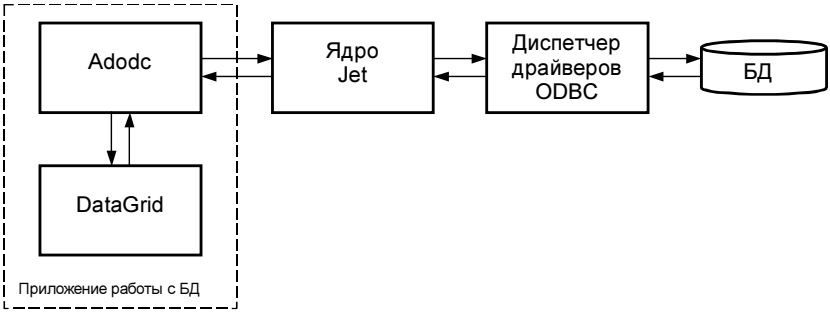


Рис. 7.3. Взаимодействие компонентов, обеспечивающих доступ и отображение данных

Технология ODBC

ODBC (Open Database Connectivity — открытое соединение с базами данных) — это технология доступа к базам данных. В основе технологии ODBC лежит идея обеспечения единого стандартного способа доступа к данным посредством универсального языка запросов SQL (Structured Query Language). Непосредственный доступ к данным обеспечивает драйвер соответствующей СУБД. Особенностью технологии ODBC является то, что взаимодействие программы, которой необходимо получить доступ к данным, с драйвером осуществляется не напрямую, а *через диспетчер драйверов* (рис. 7.4).

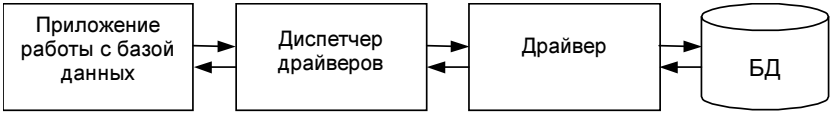


Рис. 7.4. Механизм доступа к данным посредством технологии ODBC

База данных, для доступа к которой используется технология ODBC, должна быть зарегистрирована в системе как источник данных. В процессе регистрации эта база получает уникальный идентификатор (имя) — DSN (Data Set Name — имя набора данных). В Windows регистрация базы данных в качестве источника данных ODBC осуществляется при помощи утилиты **Администратор источников данных**.

Технология ODBC поддерживается большинством СУБД, включая dBase, Microsoft Access, FoxPro, Paradox, Microsoft SQL Server, Oracle, DB2. Есть драйверы, позволяющие получить доступ к данным, находящимся в таблицах Microsoft Excel и текстовых файлах.

Строка соединения

Свойство `ConnectionString` (строка соединения) компонента `Adodc` определяет базу данных, доступ к которой обеспечивает компонент. Существуют два способа, посредством которых можно задать базу данных:

- указать в строке соединения файл базы данных. Например, строка соединения, обеспечивающая доступ к созданной в Microsoft Access базе данных (**Адресная книга**), файл которой AdrBk.mdb находится в каталоге D:\Database, выглядит следующим образом:

```
Provider=Microsoft.Jet.OLEDB.3.51;;Data
Source=D:\Database\ADRBK.MDB
```

- указать в строке соединения идентификатор базы данных, т. е. имя источника данных (DSN). Например, если база данных (**Адресная книга**) зарегистрирована в системе как источник данных ODBC под именем AB, то для того чтобы получить доступ к этой базе, свойству `ConnectionString` надо присвоить значение `DSN = AB`.

Следует обратить внимание: при использовании DSN программа работы с базой данных не знает, где находится файл этой базы и как он называется. С одной стороны, это делает программу дружественной к пользователю (файл базы данных можно поместить в любой каталог), а с другой — требует администрирования

базы данных (регистрации файла в системе как источника данных ODBC).

Программа работы с базой данных

Процесс разработки такого приложения рассмотрим на примере программы, обеспечивающей работу с базой данных Microsoft Access (условно назовем эту базу **Записная книжка**).

Создание базы данных

Перед тем как приступить к непосредственной работе в Visual Basic, надо при помощи Microsoft Access создать базу данных Записная книжка (phbk.mdb), которая содержит одну-единственную таблицу phones (табл. 7.1).

Таблица 7.1. *Характеристики таблицы phone базы данных* ***Записная книжка***

Поле (столбец)	Тип	Размер	Описание
TITLE	Char	50	Имя (фамилия, имя) или название организации (фирмы). Обязательное поле
PHONE	Char	45	Телефон и другая информация

Регистрация базы данных

В программе Записная книжка доступ к данным осуществляется через диспетчер драйверов ODBC, поэтому эту базу (phbk.mdb) надо зарегистрировать в системе.

Чтобы зарегистрировать базу данных, надо выполнить следующие действия:

1. При помощи команды **Пуск ▸ Настройка ▸ Панель управления ▸ Администрирование ▸ Источники данных (ODBC)** запустить Администратор источников данных ODBC, открыть вкладку **User DSN** и щелкнуть на кнопке **Add**.

2. На вкладке **Создать новый источник данных** выбрать драйвер, обеспечивающий доступ к данным (Microsoft Access Driver), и щелкнуть на кнопке **Готово**.
3. В окне **Установка драйвера ODBC для Microsoft Access** задать имя источника данных (`phbk`) и файл базы данных (рис. 7.5). Также рекомендуется ввести краткое описание источника данных (`База данных Записная книжка`).

В результате описанных выше действий в системе будет зарегистрирован новый источник данных, а его имя появится на вкладке **User DSN** (рис. 7.6).

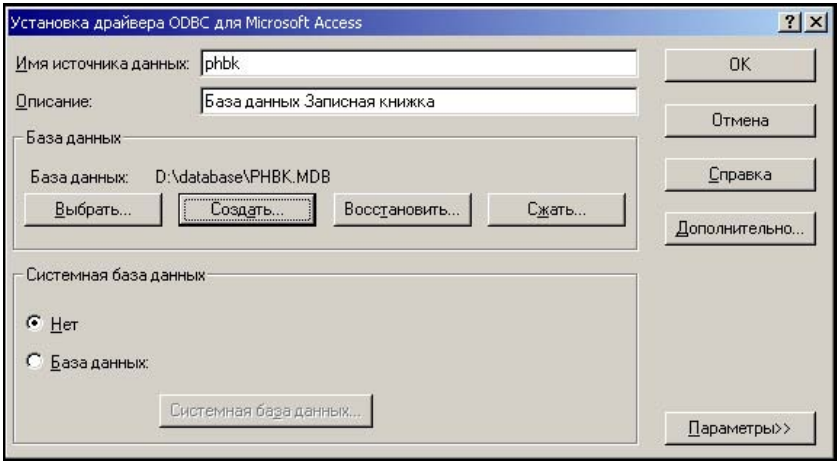


Рис. 7.5. В окне **Установка драйвера ODBC для Microsoft Access** надо задать имя источника данных

Следует обратить внимание на то, что Администратор источников данных позволяет не только зарегистрировать существующую базу, но и создать новую (для этого в окне **Установка драйвера ODBC для Microsoft Access** надо щелкнуть на кнопке **Создать**). Однако создать таблицу в базе данных одними только средствами Администратора нельзя, надо направить рассматриваемой базе данных SQL-запрос `CREATE TABLE`. Вообще задачу создания какой-либо базы данных можно возложить на программу работы с базой данных.

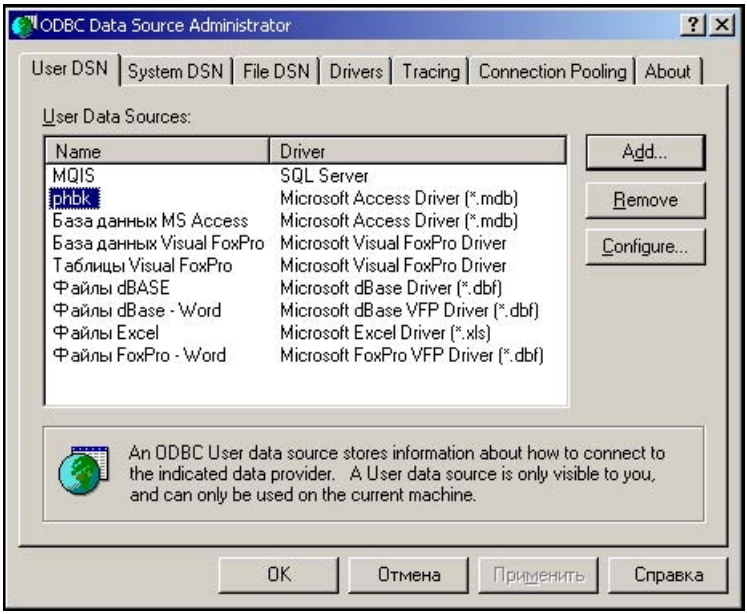


Рис. 7.6. База данных **Записная книжка** зарегистрирована в системе как источник данных `phbk`

Работа с базой данных в режиме таблицы

После того как база данных будет создана и зарегистрирована в системе как источник данных, надо запустить Visual Basic, активизировать процесс создания нового стандартного EXE-приложения (Standard EXE), подключить компоненты, обеспечивающие доступ (`Adodc`) и отображение данных (`DataGrid`), затем добавить в форму приложения компоненты `Adodc` и `DataGrid`. Вид формы в начале работы над приложением Записная книжка приведен на рис. 7.7. После этого можно приступить к настройке компонентов.

Сначала надо настроить компонент `Adodc` — задать значение свойств `ConnectionString` и `RecordSource`.

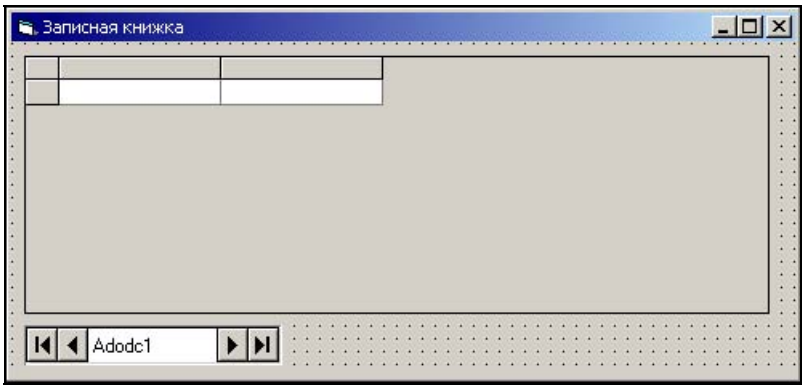


Рис. 7.7. Форма приложения Записная книжка

Свойство `ConnectionString` — *строка соединения (подключения)* содержит информацию, необходимую для подключения к базе данных. Установить значение этого свойства можно следующими способами:

- путем непосредственного ввода строки в поле значения свойства:

  `DSN=имя` (где `имя` — имя (DSN), под которым база данных зарегистрирована в системе);

- выбором источника данных на вкладке **General** окна **Property Pages** (рис. 7.8), которое становится доступным в результате щелчка на кнопке с тремя точками в строке свойства `ConnectionString` (в окне **Properties**) или в результате выбора в контекстном меню компонента `Adodc` команды **Properties**.

Свойство `RecordSource` определяет данные, с которыми работает программа. Это может быть как вся таблица данных (все записи), так и некоторая ее часть (записи, удовлетворяющие определенному критерию), а также данные, полученные из нескольких таблиц. Чтобы определить, какую информацию необходимо извлечь из базы данных при запуске программы, надо в свойство `RecordSource` записать SQL-команду, обеспечивающую выбор информации. Например, команда

```
SELECT TITLE, PHONE FROM PHONES
```

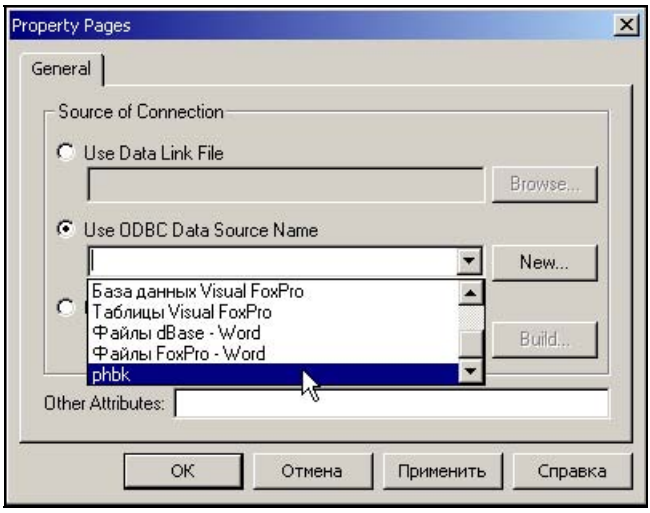


Рис. 7.8. Значение свойства `ConnectionString` можно задать путем выбора источника данных

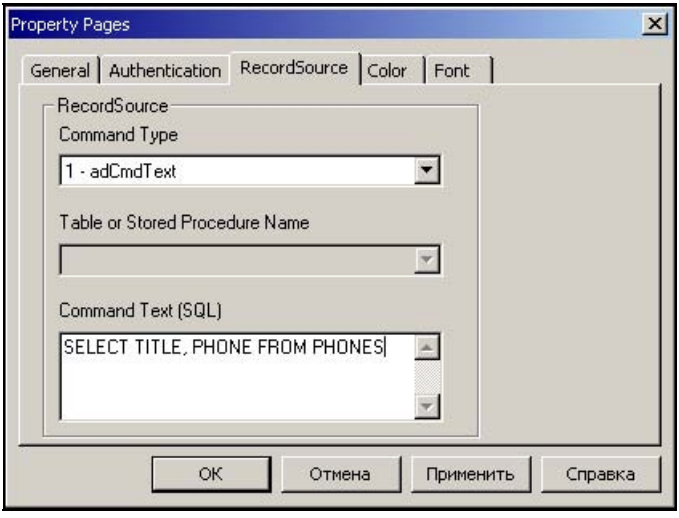


Рис. 7.9. В поле **Command Text** надо ввести SQL-команду, обеспечивающую выбор информации из базы данных

обеспечивает выбор из таблицы PHONES информации, которая находится в столбцах TITLE и PHONE. SQL-команду можно ввести непосредственно в поле значения свойства `RecordSource` или в поле **Command Text** (рис. 7.9) вкладки **RecordSource** окна **Property Pages**, которое появляется в результате выбора в контекстном меню компонента команды **Properties**.

Если SQL-команда записана с ошибками, например, неправильно указано имя поля (столбца) таблицы или имя самой таблицы, то в результате щелчка на одной из двух кнопок (**OK** или **Применить**) будет выведено сообщение об ошибке (рис. 7.10).

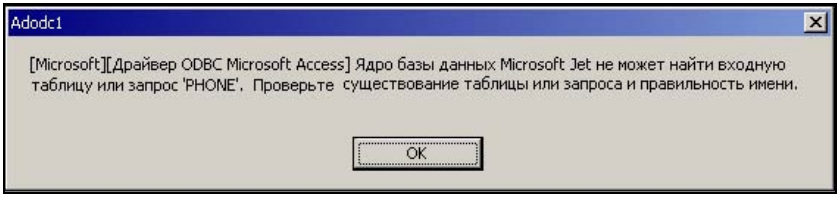


Рис. 7.10. Пример сообщения об ошибке

Значения свойств компонента `Adodc` программы работы с базой данных Записная книжка приведены в табл. 7.2.

Таблица 7.2. *Настройки компонента* `Adodc` *программы работы с БД Записная книжка*

Свойство	Значение	Примечание
`ConnectionString`	`DSN=phbk`	`phbk` — имя, под которым база данных Записная книжка (phbk.mdb) зарегистрирована в системе

`RecordSource`	`SELECT TITLE, PHONE`
	`FROM PHONES`
	или
	`SELECT * FROM PHONES`
		Отображать (выбрать) все содержимое (все записи) таблицы PHONES

Таблица 7.2 *(окончание)*

Свойство	Значение	Примечание
`Visible`	`False`	Сделать компонент невидимым. Для навигации использовать возможности компонента отображения данных

Затем надо настроить компонент `DataGrid`, который обеспечивает отображение данных в табличной форме, а также позволяет манипулировать данными (редактировать, добавлять и удалять строки таблицы). Свойства компонента `DataGrid` (табл. 7.3) определяют данные, которые отображаются в таблице.

Таблица 7.3. *Свойства компонента* `DataGrid`

Свойство	Определяет
`DataSource`	Источник данных, отображаемых в поле компонента (компонент `Adodc`)
`AllowAddNew`	Признак возможности добавления в таблицу (базу данных) новой записи
`AllowDelete`	Признак возможности удаления записи из таблицы (базы данных)
`AllowUpdate`	Признак возможности редактирования записи таблицы (базы данных)
`Caption`	Заголовок таблицы. Если текст заголовка не задан, то строка заголовка не отображается
`ColunnHeaders`	Признак отображения заголовка столбцов таблицы. По умолчанию используются имена столбцов таблицы данных. Если значение свойства равно `False`, то заголовки столбцов не отображаются
`HeadFont`	Шрифт, используемый для отображения заголовка таблицы и заголовков столбцов
`Font`	Шрифт, используемый для отображения данных

Следует обратить внимание на то, что порядок следования столбцов в поле компонента `DataGrid` соответствует порядку следования имен столбцов, указанному в команде `SELECT`, или порядку следования столбцов в исходной таблице (если после слова `SELECT` указана звездочка).

По умолчанию в заголовке столбца компонента `DataGrid` отображается название столбца таблицы данных, что не всегда удобно. Задать заголовок столбца компонента `DataGrid` можно явно на вкладке **Collunns** окна **Property Pages**, которое становится доступным в результате выбора в контекстном меню компонента `DataGrid` команды **Properties**. Текст заголовка надо ввести в поле **Caption** (рис. 7.11), предварительно выбрав в списке **Column** столбец компонента `DataGrid`, а в списке **DataField** — столбец таблицы данных.

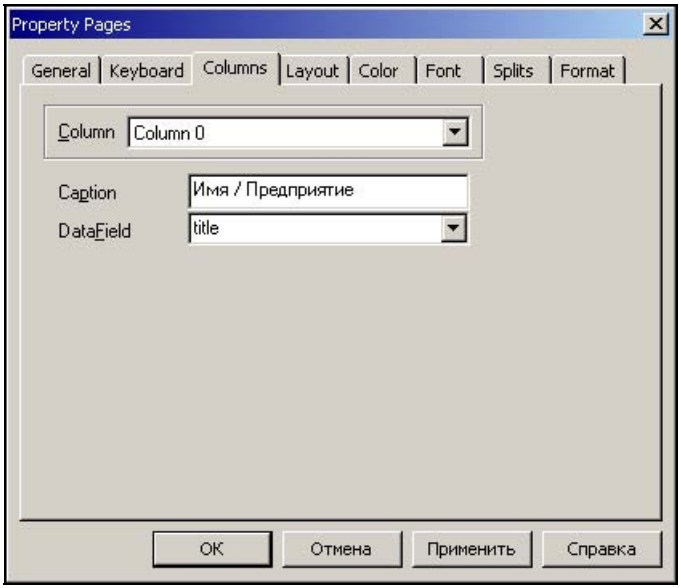


Рис. 7.11. Текст заголовка столбца надо ввести в поле **Caption**

Значения свойств компонента `DataGrid` приведены в табл. 7.4, окончательный вид формы — на рис. 7.12.

Таблица 7.4. *Значения свойств компонента* `DataGrid1`

Свойство	Значение
`DataSource`	`Adodc1`
`AllowAddNew`	`True`
`AllowDelete`	`True`
`AllowUpdate`	`True`

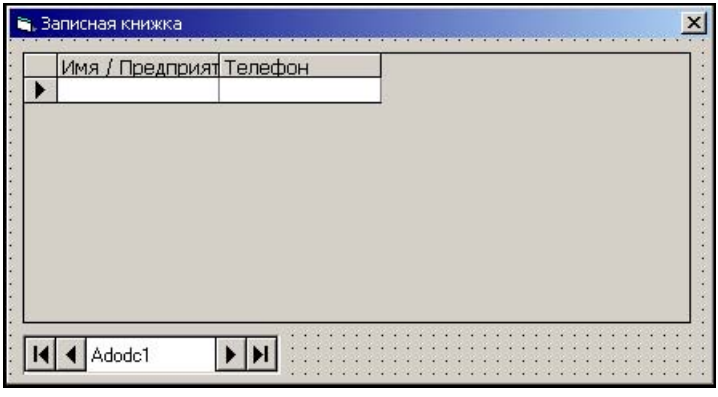


Рис. 7.12. Вид формы
после настройки компонента `DataGrid`

На этом настройка компонентов приложения заканчивается. В принципе простейшая программа работы с базой данных готова. Программу можно запустить и начать работу с базой данных.

После завершения работы по созданию формы приложения можно приступить к написанию процедур обработки событий. В простейшем случае надо создать процедуру обработки события `Error` для компонента `Adodc`, а также процедуру обработки события `Initialize` формы (листинг 7.1). Процедура обработки события `Error` обеспечивает вывод информационного сообщения в случае возникновения ошибки при обращении к базе данных, а процедура обработки события `Initialize` устанавливает ширину столбцов таблицы.

Листинг 7.1. Процедуры обработки событий `Error` и `Initialize`

```
' Если БД не зарегистрирована в системе как источник данных,
' то возникает ошибка.
Private Sub Adodc1_Error(ByVal ErrorNumber As Long, _
    Description As String, ByVal Scode As Long,_
    ByVal Source As String, _
    ByVal HelpFile As String, ByVal HelpContext As Long, _
    fCancelDisplay As Boolean)

    Dim msg As String

    If Scode = -2147467259 Then
       msg = "Ошибка доступа к базе данных. " + _
            "Убедитесь, что база данных зарегистрирована " + _
            "в системе как источник данных phbk." + vbCrLf
    End If

    MsgBox msg + Description, vbExclamation +_
    vbOKOnly, "Записная книжка"

    fCancelDisplay = True ' не отображать стандартное
                          ' сообщение об ошибке

End Sub

Private Sub Form_Initialize()
    Form1.ScaleMode = vbPixels
    DataGrid1.Columns(0).Width = 270
End Sub
```

Окно программы работы с базой данных Записная книжка приведено на рис. 7.13. Программа позволяет редактировать данные, добавлять и удалять строки.

Чтобы изменить содержимое ячейки таблицы, надо переместить курсор в нужную ячейку (при помощи клавиш перемещения курсора или щелкнув в ячейке кнопкой мыши), нажать клавишу <F2> и внести изменения.

Чтобы удалить строку, надо щелчком мыши в служебном столбце выделить эту строку и нажать клавишу <Del>. Вести новую информацию можно только в последнюю (пустую строку) таблицы. Следует обратить внимание на то, что щелчком в заголовке столбца можно выполнить сортировку данных.

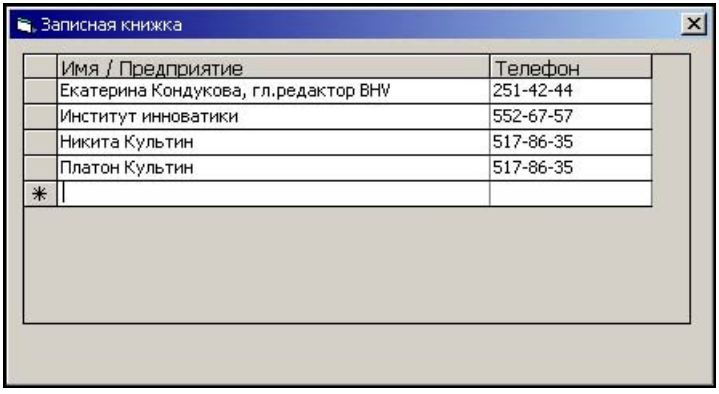


Рис. 7.13. Окно программы Записная книжка

Выбор информации из базы данных

При работе с базой данных пользователя, как правило, интересует не все ее содержимое, а некоторая конкретная информация. Найти нужные сведения можно последовательным просмотром записей. Однако такой способ поиска неудобен и малоэффективен.

Большинство систем управления базами данных позволяют выполнять выборку нужной информации путем выполнения *запросов*. Пользователь формулирует запрос, указывая критерий, которому должна удовлетворять интересующая его информация, а система выводит записи, удовлетворяющие этому запросу.

Выбрать необходимую информацию из базы данных можно, задав критерий отбора записей в команде **SELECT** компонента Adodc.

В общем виде SQL-команда, обеспечивающая выбор информации из базы данных (таблицы), выглядит так:

```
SELECT СписокПолей FROM Таблица
WHERE(Критерий)
ORDER BY СписокПолей
```

Параметр Таблица указывает таблицу базы данных, в которой находится необходимая информация.

Параметр Критерий определяет критерий (условие) отбора записей (строк) из указанной таблицы.

Параметр СписокПолей (следующий за словом **SELECT**) задает столбцы таблицы, содержимое которых интересует пользователя (если надо выбрать все столбцы таблицы, то вместо списка полей можно указать звездочку).

Параметр СписокПолей (следующий за словом **ORDER BY**) задает поля, содержимое которых используется в качестве критерия для упорядочивания отобранных записей.

Например, запрос

```
SELECT When_, What_ FROM Tasks
WHERE ( When_ = "05.06.2005" )
ORDER BY When_
```

обеспечивает выборку из базы данных Ежедневник (из таблицы Tasks) тех записей, у которых в поле When_ находится текст 05.06.2005, т. е. формирует список мероприятий, назначенных на 5 июня 2005 года (предполагается, что столбец What_ таблицы Tasks содержит названия задач (мероприятий), а столбец When_ — даты, на которые мероприятия запланированы).

Другой пример. Запрос

```
SELECT When_, What_ FROM Tasks
WHERE ( When_ >= "31.05.2005" ) AND_
      ( When_ <= "06.06.2005" )
ORDER BY When_
```

формирует список дел, назначенных на неделю (с 31 мая по 6 июня 2005 года).

В критерии запроса вместо конкретного значения поля (константы) можно указать шаблон. Например, шаблон `Сергей%` обозначат все строки, которые начинаются словом Сергей, а `%Иванов%` — все строки, в которых есть подстрока Иванов (например, Иванов Михаил, Елена Иванова и т. д.). При использовании шаблонов надо применять оператор **LIKE**. Например, запрос

```
SELECT * FROM PHONES
WHERE NAME LIKE "Ива%"
```

выводит все записи таблицы PHONES, у которых в поле NAME находится текст, начинающийся с символов `Ива`.

Запрос может быть сформирован как во время разработки формы, так и во время работы программы.

Следующая программа (Записная книжка-2) демонстрирует формирование запроса во время работы программы. Программа позволяет найти в базе данных телефон нужного человека или организации. Форма программы приведена на рис. 7.14.

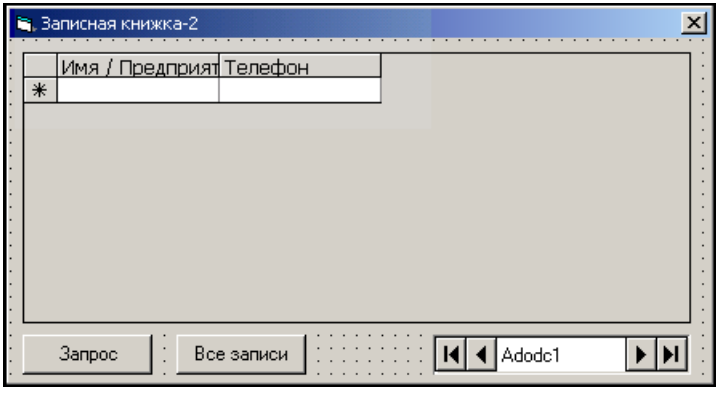


Рис. 7.14. Форма программы Записная книжка-2

После того как в форму будут добавлены компоненты `Adodc` и `DataGrid`, их надо настроить на работу с конкретной базой данных. Затем необходимо создать процедуры обработки событий `Load` и `Closing` формы, а также события `Click` кнопки **Все записи**. Модуль главной формы приведен в листинге 7.2.

Листинг 7.2. Модуль главной формы программы Записная книжка-2

```
Option Explicit
Public KeyWord As String ' критерий запроса

Private Sub Form_Initialize()
    Form1.ScaleMode = vbPixels
    DataGrid1.Columns(0).Width = 270
End Sub

Private Sub Adodc1_Error(ByVal ErrorNumber As Long, _
    Description As String, ByVal Scode As Long,_
    ByVal Source As String, ByVal HelpFile As String,_
    ByVal HelpContext As Long, fCancelDisplay As Boolean)

    Dim msg As String

    If Scode = -2147467259 Then
        msg = "Ошибка доступа к базе данных. " + _
            "Убедитесь, что база данных зарегистрирована " + _
            "в системе  как источник данных phbk." + vbCrLf

        Command1.Enabled = False
        Command2.Enabled = False
    End If
```

```
    MsgBox msg + Description, vbExclamation + vbOKOnly,_
    "Записная книжка-2"

    fCancelDisplay = True ' не отображать стандартное
                          ' сообщение об ошибке

End Sub
```

Для ввода параметра критерия запроса (имени человека или названия организации) в рассматриваемой программе необходимо щелкнуть мышью на кнопке **Запрос**, после чего активизируется диалоговое окно **Запрос** (рис. 7.15).

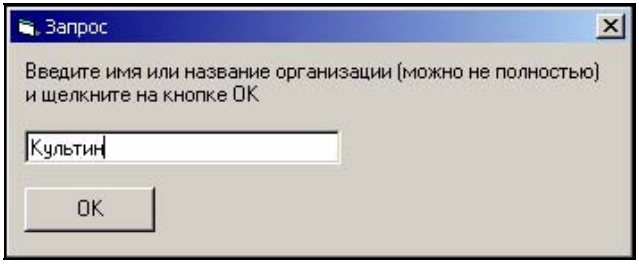


Рис. 7.15. Для ввода параметра критерия запроса используется окно **Запрос**

Чтобы программа могла вывести окно, отличное от главного, в проект надо добавить форму и выполнить ее настройку. А чтобы добавить в проект форму, необходимо сначала в меню **Project** выбрать команду **Add Form**, затем (в окне **Add Form**) — тип формы (в данном случае **Form**).

Настройка формы производится обычным образом. Значения свойств формы `Form2` разрабатываемого приложения приведены в табл. 7.5, а ее вид — на рис. 7.16.

Таблица 7.5. *Значение свойств формы* `Form2`

Свойство	Значение	Комментарий
`BorderStyle`	`FixedDialog`	Тонкая граница (нельзя изменить размер окна), нет кнопок **Развернуть** и **Свернуть** в заголовке окна
`StartPosition`	`Center`	Положение окна на экране — по центру относительно главного окна программы

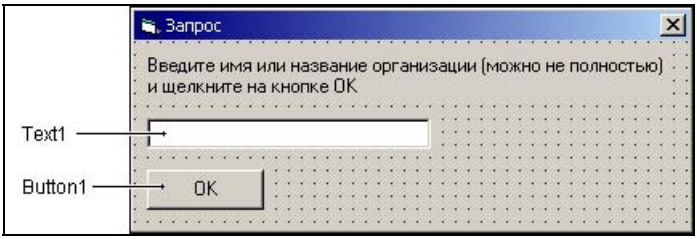


Рис. 7.16. Форма **Запрос**

После того как будет создана и настроена форма **Запрос** (`Form2`), в модуль главной формы надо добавить процедуры обработки события `Click` на кнопках **Запрос** и **Все записи** (листинг 7.3), а в модуль формы `Form2` — процедуру обработки события `Click` на кнопке **OK** (листинг 7.4). Следует обратить внимание на то, что передача критерия запроса из формы запроса в главную форму осуществляется через глобальную переменную `KeyWord`, которая объявлена в модуле главной формы как `Public`.

Листинг 7.3. Процедуры обработки события `Click` на кнопках Запрос и Все записи

```
' щелчок на кнопке "Запрос"
Private Sub Command1_Click()
    KeyWord = ""

    Form2.Show (vbModal) ' отобразить окно Запрос
```

```
    If KeyWord <> "" Then
        ' пользователь ввел критерий запроса
        ' сформировать SQL-запрос
        Adodc1.RecordSource = _
           "SELECT * FROM phones WHERE Title LIKE " + _
           Chr(39) + "%" + KeyWord + "%" + Chr(39) + _
           "ORDER BY title"
        Adodc1.Refresh ' выполнить запрос

    End If
End Sub

' щелчок на кнопке "Все записи"
Private Sub Command2_Click()
    Adodc1.RecordSource = _
           "SELECT * FROM phones ORDER BY title"
    Adodc1.Refresh ' выполнить запрос
End Sub
```

Листинг 7.4. Процедура обработки события `Click` на кнопке OK

```
Private Sub Command1_Click()
    Form1.KeyWord = Text1.Text  ' передать критерий запроса
                                ' в модуль главной формы
                                '(Form1)
    Form2.Hide   ' скрыть окно Запрос
End Sub
```

Работа с базой данных в режиме формы

Существуют два режима работы с базой данных: режим таблицы и режим формы.

В режиме таблицы база данных отображается в виде таблицы, что позволяет видеть несколько записей одновременно.

Если записи базы данных состоят из большого количества полей или информация распределена по нескольким таблицам, то работать с такой базой в режиме *таблицы* неудобно — размер экрана (ширина области отображения таблицы) не позволяет видеть всю информацию одновременно (чтобы увидеть содержимое поля приходится изменять ширину столбца или прокручивать окно по горизонтали). В этом случае удобней работать с базой данных в режиме *формы*. В режиме формы на экране отображается только одна запись, что позволяет видеть содержимое всех полей одновременно (рис. 7.17).

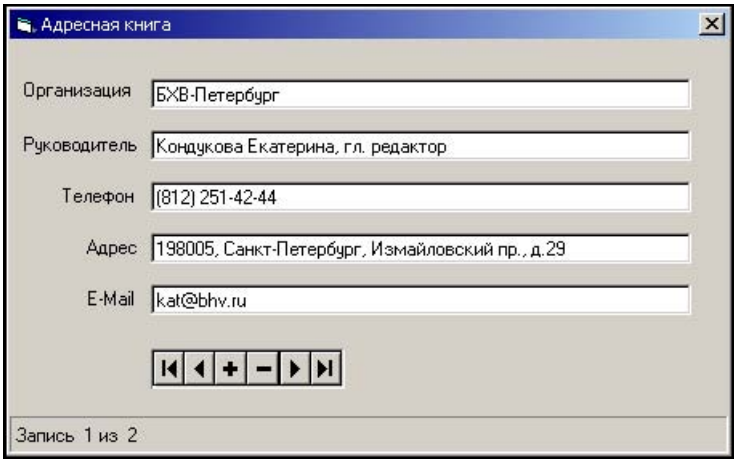


Рис. 7.17. Работа с базой данных в режиме формы

Довольно часто режимы комбинируют. Краткую информацию (содержимое ключевых полей) выводят в виде таблицы, а при необходимости видеть содержимое всех полей выполняют переключение в режим формы. Режим формы также удобен для ввода информации в базу данных.

Рассмотрим программу, демонстрирующую работу с базой данных в режиме формы. Программа обеспечивает выполнение основных операций (просмотр, добавление, удаление записей)

с базой данных Microsoft Access Адресная книга (ArdBk.mdb), которая состоит из одной-единственной таблицы Contacts (табл. 7.6).

Таблица 7.6. *Поля таблицы Contacts базы данных Адресная книга*

Поле	Тип	Размер	Комментарий
Title	Строковый	50	Название организации
Phone	Строковый	50	Телефон
Manager	Строковый	50	Руководитель (контактное лицо)
Address	Строковый	50	Адрес
Email	Строковый	50	Адрес электронной почты

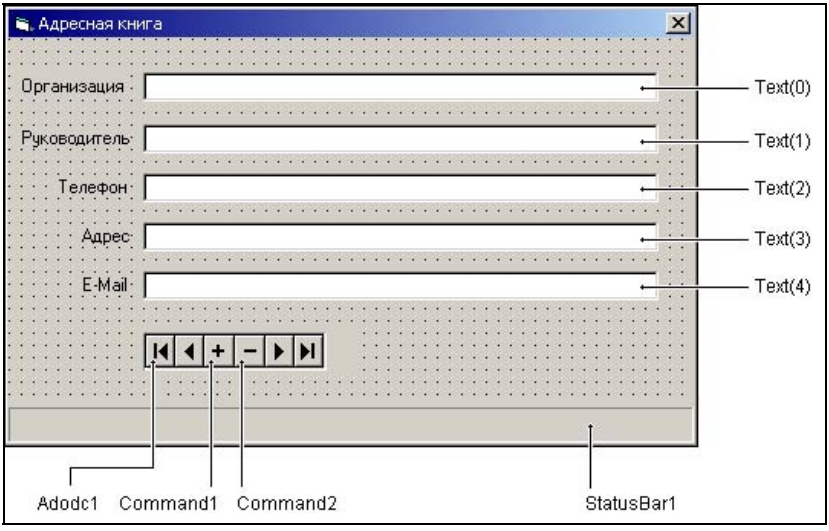


Рис. 7.18. Фома программы Адресная книга

Форма программы приведена на рис. 7.18. Кнопки, обеспечивающие добавление (`Command1`) и удаление (`Command2`) записей, помещены в поле компонента `Adodc1`. Это обычные `CommandButton` графические кнопки. Рисунок для кнопок создан в редакторе Paint. Компонент `StatusBar` используется для отображения ин-

формации о состоянии базы данных, во время работы программы в строке состояния отображается номер активной в данный момент записи. Чтобы компонент был доступен во время разработки формы, надо подключить Microsoft Windows Common Control 6.0 (команда **Project ▸ Components**).

Следует обратить внимание на то, что для отображения содержимого записи (строки таблицы) используются не отдельные компоненты `TextBox`, а массив компонентов. Замена отдельных компонентов массивом компонентов позволяет значительно сократить текст программы — десять процедур обработки событий (`KeyPress` и `GotFocus` для каждого из пяти компонентов `TextBox`) заменены двумя, каждая из которых обрабатывает соответствующее событие для всех компонентов массива.

Чтобы создать массив компонентов, надо:

1. Поместить на форму компонент и выполнить его настройку, в т. ч. изменить имя так, чтобы оно соответствовало имени создаваемого массива компонентов (в рассматриваемом примере имя `Text1` заменено на `Text`).
2. В меню **Edit** выбрать команду **Copy**.
3. В меню **Edit** выбрать команду **Paste**.
4. В окне запроса **You already have a control named Do you want to create a control array?** (Элемент управления с именем уже есть. Вы хотите создать массив компонентов?) сделать щелчок на кнопке **Да** (рис. 7.19). В результате в форму будет добавлен компонент.
5. Повторить шаг 3 нужное количество раз.

Рис. 7.19. Чтобы создать массив компонентов, надо щелкнуть на кнопке **Да**

После того как массив компонентов будет сформирован, можно выполнить настройку отдельных компонентов массива и создать процедуры обработки событий. Настройка компонентов выполняется обычным образом. События, которые возникают для каждого элемента массива компонентов, обрабатываются соответствующей, единой для всех компонентов массива, процедурой. Процесс создания процедуры обработки события для компонентов массива ничем не отличается от процесса создания процедуры обработки события для отдельного компонента. В заголовке процедуры обработки события, созданной для массива компонентов, помимо других параметров есть параметр `Index`. Через этот параметр в процедуру передается номер компонента (элемент массива), на котором произошло событие.

В качестве примера в листинге 7.5 приведена процедура обработки события `KeyPress` для массива компонентов `TextBox` формы программы доступа к базе данных Адресная книга. Процедура обрабатывает нажатие клавиши в любом из полей `Text(0)`—`Text(4)` (при нажатии клавиши <Enter> переводит курсор в следующее по порядку поле ввода).

Листинг 7.5. Пример обработки события для массива компонентов

```
' нажатие клавиши в поле редактирования
Private Sub Text_KeyPress(Index As Integer,_
            KeyAscii As Integer)
    If KeyAscii = 13 Then ' пользователь нажал <Enter>
        ' курсор в следующее поле ввода
        If Index < 4 Then Text(Index + 1).SetFocus
    End If
End Sub
```

Для отображения и редактирования записей базы данных используются компоненты `Text(0)`—`Text(4)` (см. рис. 7.18). Чтобы в поле редактирования (компонента `TextBox`) появилось содержимое поля записи базы данных, надо *связать* поле редактирования и соответствующее поле записи: присвоить значения

свойствам `DataSource` и `DataField` поля редактирования. Для этого надо выбрать компонент `TextBox`, в окне **Properties** выбрать источник данных (присвоить значение свойству `DataSource`), а затем — поле таблицы данных (рис. 7.20).

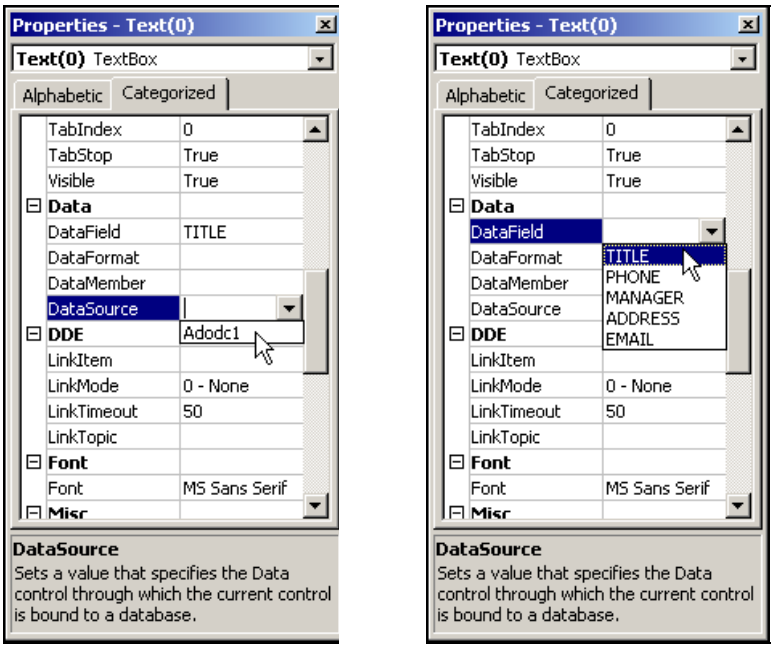


Рис. 7.20. Связывание поля редактирования и поля записи базы данных

Значения свойств компонентов приведены в табл. 7.7 и 7.8, текст модуля формы — в листинге 7.6.

Таблица 7.7. *Значения свойств компонента* `Adodc`

Свойство	Значение
`ConnectionString`	`DSN=adrbk`
`RecordSource`	`SELECT * FROM CONTACTS ORDER BY TITLE`

Таблица 7.8. *Значения свойств компонентов* `Text(0)—Text(4)`

Свойство	Значение
Text(0).DataSource	Adodc1
Text(0).DataField	TITLE
Text(1).DataSource	Adodc1
Text(1).DataField	MANAGET
Text(2).DataSource	Adodc1
Text(2).DataField	PHONE
Text(3).DataSource	Adodc1
Text(3).DataField	ADDRESS
Text(4).DataSource	Adodc1
Text(4).DataField	EMAIL

Листинг 7.6. Модуль главной формы программы работы с базой данных Адресная книга

```
Option Explicit

' начало работы программы
Private Sub Form_Load()
    ' ограничение количества символов, которое можно
    ' ввести в поле редактирования
   Text(0).MaxLength = 50
   Text(1).MaxLength = 50
   Text(2).MaxLength = 50
   Text(3).MaxLength = 50
   Text(4).MaxLength = 50

  ' открыть базу данных
  ' Adodc1.CommandType = adCmdText
  ' Adodc1.RecordSource = "SELECT * FROM adrbk ORDER BY title"
  ' Adodc1.Refresh
```

```
   ' если в базе данных нет записей
     If Adodc1.Recordset.RecordCount = 0 Then

   ' если в базе данных нет записей и пользователь
   ' попытался сохранить данные, предварительно не нажав
   ' кнопку "Добавить запись", то в программе возникнет
   ' ошибка доступа к записи. Чтобы этого не произошло,
   ' предварительно добавляется запись. При этом если
   ' пользователь все-таки нажмет кнопку "Добавить", то двух
   ' новых записей не появится, потому что если ни одно из
   ' полей не заполнено, то запись не добавляется.

        Adodc1.Recordset.AddNew
     End If

End Sub

' щелчок на кнопке "+" (Добавить запись)
Private Sub Command1_Click()
    Adodc1.Recordset.AddNew
    Text(0).SetFocus
End Sub


' щелчок на кнопке "-" (Удалить запись)
Private Sub Command3_Click()
    Dim r
    r = MsgBox("Удалить запись?", vbQuestion + vbOKCancel, _
               "База данных Адресная книга")
    If r = vbOK Then

       If Adodc1.Recordset.RecordCount <> 0 Then
          Adodc1.Recordset.Delete
```

```
        If Adodc1.Recordset.EOF Then
            Adodc1.Recordset.MoveNext
        Else
            Adodc1.Recordset.MoveLast
        End If
    End If

    If Adodc1.Recordset.RecordCount = 0 Then
        Adodc1.Recordset.AddNew
    End If
  End If
End Sub

' курсор перешел в поле редактирования
Private Sub Text_GotFocus(Index As Integer)
    Text(Index).Text = RTrim(Text(Index).Text)
    ' выделить текст, который находится в поле редактирования
    Text(Index).SelStart = 0
    Text(Index).SelLength = Len(Text(Index).Text)
End Sub

' нажатие клавиши в поле редактирования
Private Sub Text_KeyPress(Index As Integer,_
            KeyAscii As Integer)
    If KeyAscii = 13 Then ' пользователь нажал <Enter>
        ' курсор в следующее поле ввода
        If Index < 4 Then Text(Index + 1).SetFocus
    End If
End Sub

' индикация номера текущей записи
Private Sub Adodc1_MoveComplete(ByVal adReason As_
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus_
As ADODB.EventStatusEnum, ByVal pRecordset As_
ADODB.Recordset)
```

```
    StatusBar1.SimpleText = "Запись " &_
              Str(Adodc1.Recordset.AbsolutePosition) & _
              " из " & Str(Adodc1.Recordset.RecordCount)

End Sub

' завершение работы с программой
Private Sub Form_Unload(Cancel As Integer)
    If Text(0).DataChanged Or Text(1).DataChanged Or_
       Text(2).DataChanged Or Text(3).DataChanged Or_
       Text(4).DataChanged Then_
       Adodc1.Recordset.UpdateBatch (adAffectCurrent)
    End If
End Sub
```

Довольно часто режим формы комбинируют с режимом таблицы. На рис. 7.21 приведена форма приложения, демонстрирующего работу с базой данных Адресная книга, в которую добавлен компонент `DataGrid` (значения свойств компонента приведены в табл. 7.9), обеспечивающий просмотр базы еще и в режиме таблицы.

Следует обратить внимание: работа компоненента `DataGrid1` и полей редактирования синхронизирована. Кнопка `Command3` используется для управления отображением таблицы. В начале работы программы таблица не видна (рис. 7.22), она становится доступной (рис. 7.23), если пользователь сделает щелчок на кнопке **Таблица**. Повторный щелчок на этой кнопке скроет таблицу. Процедура обработки события `Clik` на кнопке **Таблица** приведена в листинге 7.7.

Таблица 7.9. *Значения свойств компонента* `DataGrid1`

Свойство	Значение	Свойство	Значение
DataSource	Adodc1	AllowDelete	True
AllowAddNew	True	AllowUpdate	True

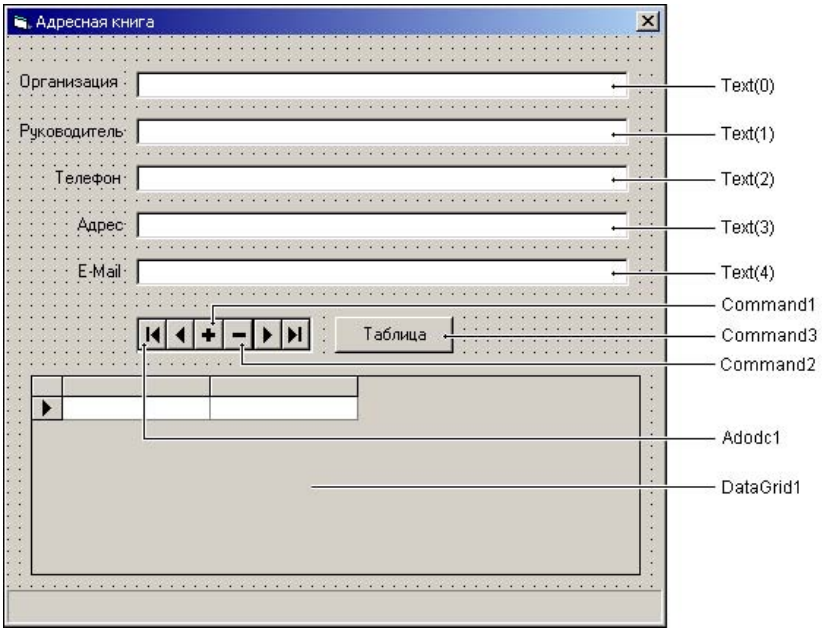


Рис. 7.21. Форма программы Адресная книга

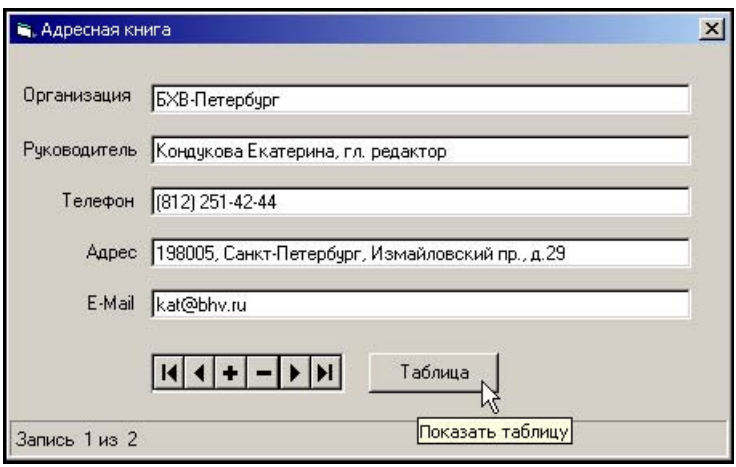


Рис. 7.22. Работа с базой данных в режиме формы

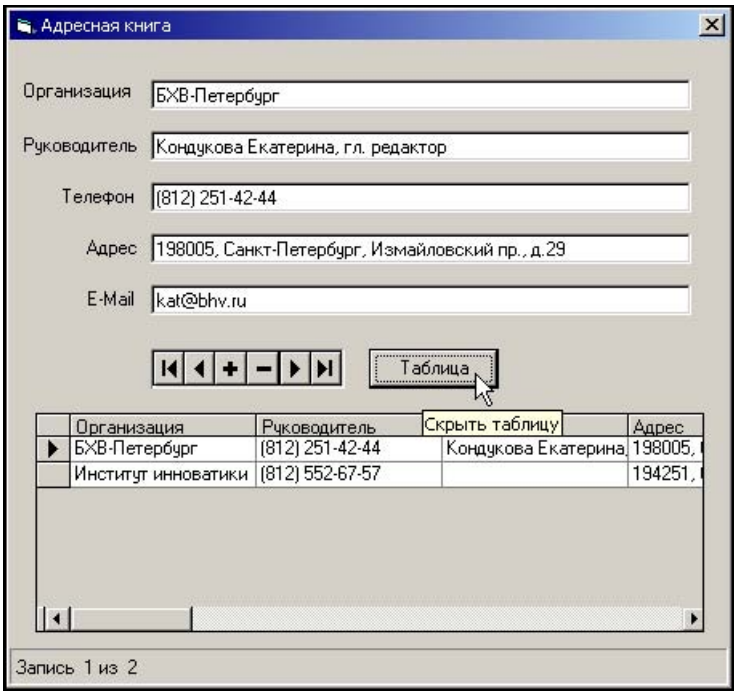


Рис. 7.23. Щелчок на кнопке **Таблица** скроет таблицу

Листинг 7.7. Обработка события `Click` на кнопке Таблица

```
' щелчок на кнопке "Таблица"
Private Sub Command3_Click()
    ' изменить высоту окна
    If DataGrid1.Visible Then
        ' скрыть таблицу
        DataGrid1.Visible = False
        Form1.Height = Form1.Height - _
        Form1.ScaleY(DataGrid1.Height + 8, vbPixels, vbTwips)
        Command3.ToolTipText = "Показаь таблицу"
    Else
        ' показать таблицу
```

```
        DataGrid1.Visible = True
        Form1.Height = Form1.Height + _
        Form1.ScaleY(DataGrid1.Height + 8, vbPixels, vbTwips)
        Command3.ToolTipText = "Скрыть таблицу"
    End If
End Sub
```

Для того чтобы в начале работы программы таблица не отображалась, в процедуру `Load` (обработки события загрузки формы) добавлены инструкции, которые устанавливают требуемый размер окна (листинг 7.8). Эта же процедура задает заголовки таблицы.

Листинг 7.8. Процедура обработки события `Load`

```
' начало работы программы
Private Sub Form_Load()
    ' ограничение количества символов, которое можно
    ' ввести в поле редактирования
   Text(0).MaxLength = 50
   Text(1).MaxLength = 50
   Text(2).MaxLength = 50
   Text(3).MaxLength = 50
   Text(4).MaxLength = 50

  ' открыть базу данных
  ' Adodc1.CommandType = adCmdText
  ' Adodc1.RecordSource = "SELECT * FROM adrbk ORDER BY
  ' title"
  ' Adodc1.Refresh

  ' если в базе данных нет записей
    If Adodc1.Recordset.RecordCount = 0 Then

  ' если в базе данных нет записей и пользователь
  ' попытался сохранить данные, предварительно не нажав
```

```
    ' кнопку "Добавить запись", то в программе возникнет
    ' ошибка доступа к записи. Чтобы этого не произошло,
    ' предварительно добавляется запись. При этом если
    ' пользователь все-таки нажмет кнопку "Добавить", то двух
    ' новых записей не появится, т. к. если ни одно из полей
    ' не заполнено, то запись не добавляется.

        Adodc1.Recordset.AddNew
    End If

    Command1.ToolTipText = "Добавить запись"
    Command2.ToolTipText = "Удалить запись"

    ' настройка компонента DataGrid
    DataGrid1.Columns(0).Caption = "Организация"
    DataGrid1.Columns(1).Caption = "Руководитель"
    DataGrid1.Columns(2).Caption = "Телефон"
    DataGrid1.Columns(3).Caption = "Адрес"
    DataGrid1.Columns(4).Caption = "E-Mail"

    ' скрыть таблицу (компонент DataGrid)
    DataGrid1.Visible = False
    Form1.Height = Form1.Height - _
    Form1.ScaleY(DataGrid1.Height + 8, vbPixels, vbTwips)
    Command3.ToolTipText = "Показать таблицу"

End Sub
```

Еще раз о создании базы данных

При разработке программ работы с базами данных Записная книжка и Адресная книга предполагалось, что эти базы (файлы phbk.mdb и, соответственно, adrbk.mdb) существуют и зарегистрированы в системе. Процесс регистрации уже был подробно

описан. Теперь рассмотрим, как можно создать базу данных без использования соответствующей СУБД, в частности, базу данных в формате Microsoft Access без Microsoft Access.

Создание файла базы данных

Создание файла базы данных обеспечивает метод `Create` объекта `Catalog`. Объект `Catalog` является объектом ADO, но, в отличие от других ранее рассмотренных объектов (`Connection`, `Command` и `Recordset`), которые находятся в объектной библиотеке Microsoft ActiveX Data Objects 2.1 (ADODB), объект `Catalog` располагается в библиотеке Microsoft Data Ext. 2.1 for DLL and Security (ADO Ext). Поэтому в проект надо добавить ссылку на библиотеку ADO Ext (рис. 7.24).

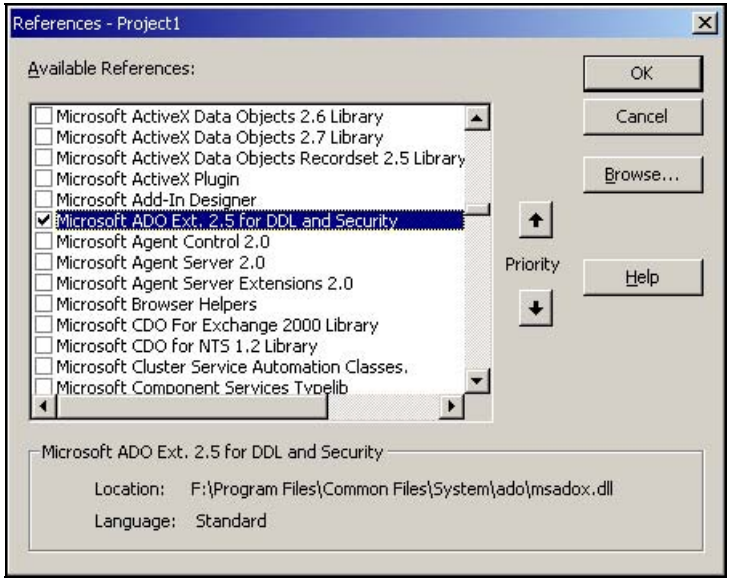


Рис. 7.24. Для доступа к объекту `Catalog`, обеспечивающему создание БД, в проект надо добавить ссылку на библиотеку ADO Ext

Чтобы создать файл базы данных, надо вызвать метод `Create`, указав в качестве параметра строку соединения. Например, в результате выполнения инструкции

```
aCatalog.Create
("Provider=Microsoft.Jet.OLEDB.3.51;Data
Source=D:\Database\AdrBk.mdb")
```

в каталоге D:\Database будет создан файл AdrBk.mdb — база данных формата Microsoft Access. Точнее, будет создан файл базы данных Microsoft Jet, формат которой использует СУБД Microsoft Access. Следует обратить внимание на то, что метод `Create` не только создает файл базы данных, но и открывает соединение, доступ к которому можно получить через свойство `ActiveConnection`.

Создание таблицы

Создать таблицу в базе данных можно, направив базе данных SQL-команду (запрос) `CREATE TABLE`, которая в общем виде выглядит так:

`CREATE TABLE` Таблица (Поле$_1$ Тип$_1$, Поле$_2$ Тип$_2$, ... Поле$_k$ Тип$_k$)

где:

- Таблица — имя таблицы, которая будет создана в результате выполнения SQL-команды `CREATE`;
- Поле$_i$ — имя *i*-го столбца таблицы;
- Тип$_i$ — тип *i*-го столбца таблицы.

Например, в результате выполнения команды

```
CREATE TABLE Contacts (Title CHAR(50), Phone CHAR(50),_
    Manager CHAR(50), Address CHAR(50), Email CHAR(50))
```

в базе данных, которой адресован запрос, будет создана таблица `Contacts`. Следует обратить внимание: таблица будет создана только в том случае, если в базе данных таблицы с указанным именем нет.

Параметр, указанный после имени поля (столбца), задает тип данных столбца таблицы. Помимо типа `CHAR` (символьный)

можно задать NUMBER (числовой), CURRENCY (денежный), DATE (дата) или TIME (время). Например, команда

```
CREATE TABLE Expenses (Sum CURRENCY, aDate DATE,_
                       Comment CHAR(50))
```

создает в базе данных таблицу Expenses (расходы), которая состоит из столбцов Sum (сумма), aDate (дата) и Comment (комментарий).

Если поле записи является обязательным (т. е. обязательно должно содержать информацию), то после идентификатора типа поля надо указать параметр NOT NULL. Например, очевидно, что в таблице Contacts поле Title обязательно должно быть заполнено. Поэтому команду, обеспечивающую создание таблицы, следует написать так:

```
CREATE TABLE Contacts (Title CHAR(50) NOT NULL,_
 Phone CHAR(50), Manager CHAR(50), Address CHAR(50),_
 Email CHAR(50))
```

Добавление информации

Чтобы добавить в таблицу информацию (запись), надо направить базе данных команду **INSERT INTO**, указав имя таблицы, имена и значения полей добавляемой записи. Например, команда:

```
INSERT INTO Expenses (Sum, aDate,Comment)_
            VALUES ('30.08.2004', '864', 'Транспортный налог')
```

добавляет в таблицу информацию об уплате налога на использование автомобиля.

Следует обратить внимание на то, что если при создании таблицы поле было объявлено как обязательное, то его значение обязательно должно быть указано. Если значение обязательного поля не указано, то запись в таблицу добавлена не будет.

Удаление таблицы

Иногда возникает необходимость удалить из базы данных таблицу. Сделать это можно, направив к базе данных SQL-команду **DROP TABLE**.

Например, команда

```
DROP TABLE Expenses
```

удаляет из базы данных таблицу `Expenses`.

Создание базы данных

Следующая программа (ее форма представлена на рис. 7.25) демонстрирует, как можно создать базу данных (файл базы данных) и поместить в нее таблицу, не прибегая к помощи СУБД. Текст программы приведен в листинге 7.9.

Следует обратить внимание: для доступа к объектам ADODB (`Connection` и `Command`) и ADOX (`Catalog`) в проект надо поместить ссылки на библиотеки Microsoft ActiveX Data Objects 2.1 (ADODB) и Microsoft ADO Ext. 2.1 for DLL and Security (ADOX).

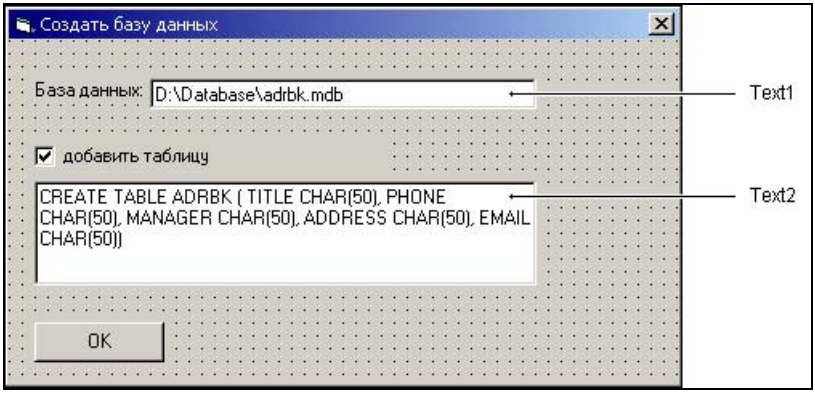


Рис. 7.25. Форма программы Создать базу данных

Листинг. 7.9. Создание базы данных

```
Option Explicit

' Для доступа к объектам ADODB и ADOX в проекте должны быть
' ссылки на:
```

```
'  - Microsoft ActiveX Data Objects 2.1 Library (ADODB)
'  - Microsoft ADO Ext. 2.5 fo DLL and Security (ADOX)

Private Sub Command1_Click()

    Dim ConnStr As String '  строка соединения

    Dim aCatalog As New ADOX.Catalog

    Dim conn As New ADODB.Connection
    Dim cmd As New ADODB.Command

    ConnStr = "Provider=Microsoft.Jet.OLEDB.3.51;_
               Data Source=" + Text1.Text

    ' создать базу данных (mdb-файл)
    On Error GoTo er1
    aCatalog.Create (ConnStr)
    MsgBox "База данных " + Text1.Text + " создана.", _
            vbInformation + vbOKOnly, "Создать базу данных"

    Set conn = aCatalog.ActiveConnection

    If Check1.Value = Checked Then
       ' добавить в созданную базу данных таблицу
       ' соединение уже открыто

       Set cmd.ActiveConnection = conn

       cmd.CommandText = Text2.Text
       On Error GoTo er2
       cmd.Execute ' выполнить команду

       MsgBox "SQL-команда: " + vbCr + Text2.Text +_
      "выполнена.", vbInformation, "Выполнить SQL-команду"
```

```
        ' закрыть соединение
        conn.Close
    End If

    Exit Sub

er1: ' ошибка создания базы данных
    MsgBox "Ошибка создания базы данных." + vbCr + Error, _
           vbExclamation + vbOKOnly, "Ошибка"
    Exit Sub


er2: ' ошибка выполнения SQL-команды
    MsgBox "Ошибка выполнения SQL-команды." + vbCr + Error, _
           vbExclamation + vbOKOnly, "Ошибка"
    conn.Close
    Exit Sub

End Sub

Private Sub Check1_Click()
    If Check1.Value = Checked Then
        Text2.Enabled = True
        Text2.SetFocus
    Else
        Text2.Enabled = False
    End If
End Sub

Private Sub Text1_GotFocus()
    Text1.SelStart = 0
    Text1.SelLength = Len(Text1.Text)
End Sub
```

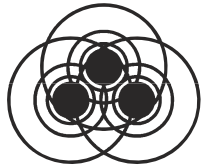
```
Private Sub Text2_GotFocus()
    Text2.SelStart = 0
    Text2.SelLength = Len(Text2.Text) - 1
End Sub
```

Установка программы работы с базой данных на компьютере пользователя

Часто возникает необходимость перенесения созданной базы данных на другой компьютер. В отличие от процесса установки обычной (простой) программы, когда, как правило, достаточно скопировать только выполняемый файл (EXE), при переносе программы управления базой данных на компьютер пользователя надо установить не только саму программу управления базой данных, но и, возможно, другие библиотеки, обеспечивающие доступ к базе данных, а также саму базу данных. Кроме того, базу данных надо *зарегистрировать* в системе пользователя.

Для установки на другом компьютере программы, созданной в Visual Basic, рекомендуется использовать специальную программу установки, созданную при помощи утилиты Package & Deployment Wizard, которая входит в комплект поставки Visual Basic.

Глава 8



Справочная система

Каждая программа должна иметь справочную систему, используя которую пользователь может получить информацию о назначении программы и о том, как с ней работать (выполнить ту или иную операцию).

В настоящее время на практике используются три формата отображения справочной информации: WinHelp, HTML Help 1.*x* и MS Help 2.*x*.

Формат WinHelp, разработанный еще во времена Windows 3.*x*, считается устаревшим, хотя до недавнего времени во многих приложениях использовался именно этот формат. Справочная система формата WinHelp представляет собой набор HLP-файлов. Отображение справочной информации (рис. 8.1) обеспечивает программа Winhelp, являющаяся составной частью операционной системы. Создать справочную систему формата WinHelp можно при помощи Microsoft Help Workshop.

Большинство современных программ отображают справочную информацию в формате HTML Help 1.*x* (рис. 8.2).

Основой HTML Help 1.*x* справочной системы являются CHM-файлы — это *компилированные* HTML-документы, полученные путем компиляции (объединения) в один файл всех HTML-файлов, содержащих справочную информацию. Отображение справочной информации этого формата также обеспечивает операционная система (программа hh.exe). Справочную систему формата HTML Help 1.*x* можно создать при помощи HTML Help Workshop.

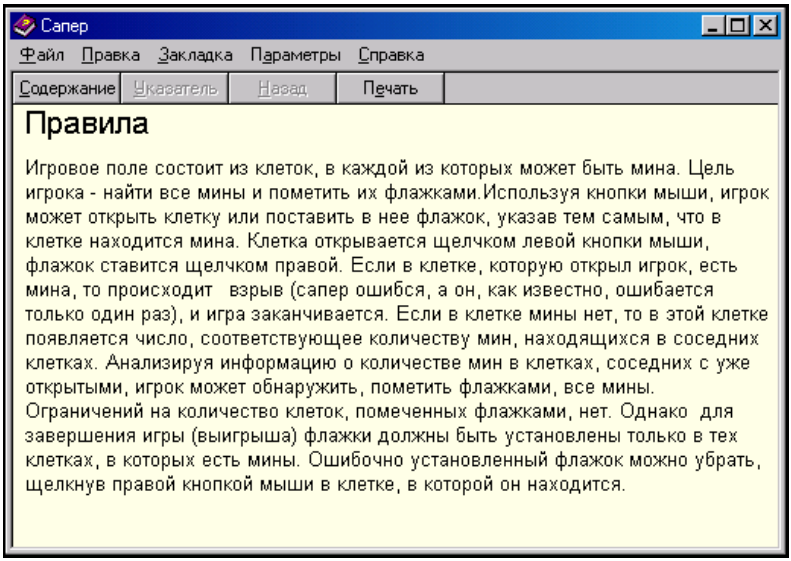


Рис. 8.1. Пример справочной системы формата WinHelp

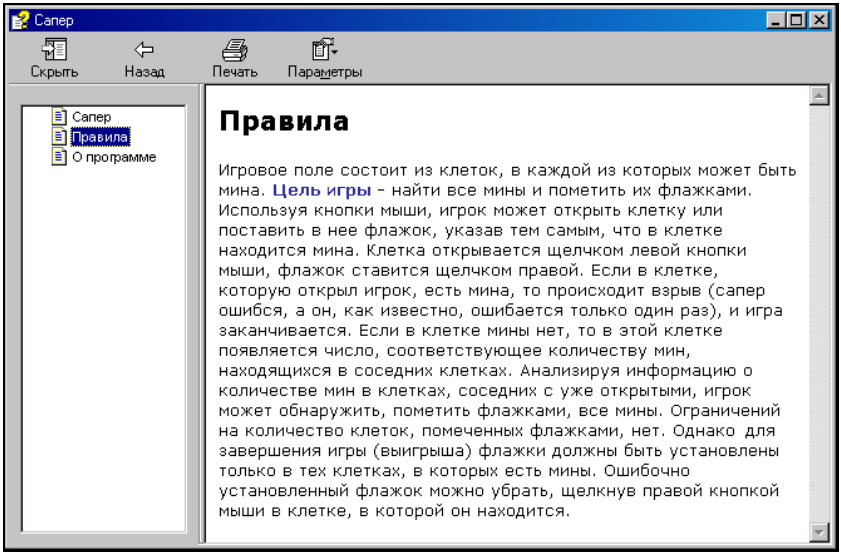


Рис. 8.2. Пример справочной системы формата HTML Help 1.*x*

Появление технологии .NET привело к возникновению нового формата отображения справочной информации — MS Help 2.*x* (рис. 8.3), который используется в .NET продуктах фирмы Microsoft. Типичным примером справочной системы в таком формате является справочная система Microsoft .NET Framework SDK. Отличительной особенностью справочной системы формата MS Help 2.*x* является то, что информация может находиться в любой точке системы (в т. ч. и на другом компьютере). Отображение этой информации обеспечивает программа Document Explorer (dexplore.exe), а сам процесс получения справочной информации весьма напоминает процесс доступа к узлу Web.

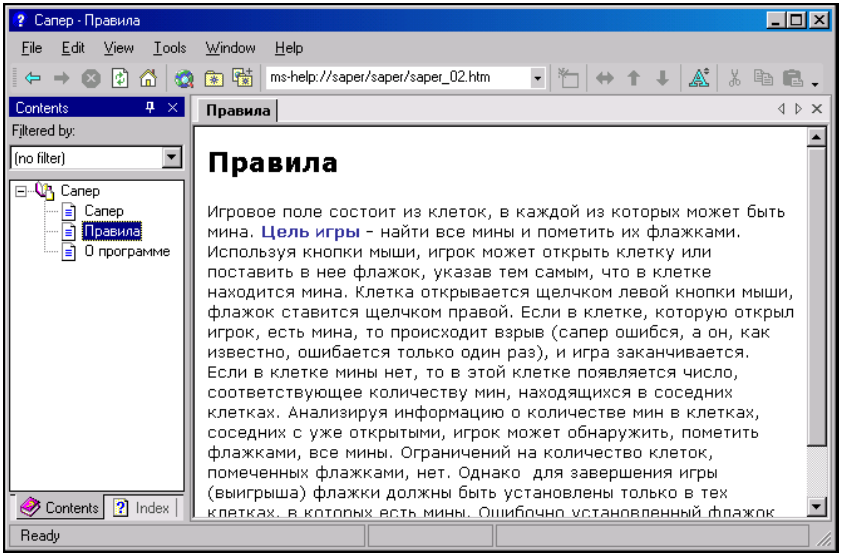


Рис. 8.3. Пример справочной системы формата MS Help 2.*x*

Следует обратить внимание: открыть справочную информацию щелчком на имени файла (как это можно было сделать с HLP- или CHM-файлом) нельзя. Кроме того, справочная информация должна быть зарегистрирована в системе, что делается при помощи утилиты hxreg, входящей в MS Help SDK.

Создание справочной системы формата MS Help 2.*x* производится с помощью утилиты hxcomp, которая также входит в MS Help 2 SDK. Утилита hxconv позволяет создать справочную систему формата MS Help 2.*x* путем конвертации существующей справочной системы формата HTML Help 1.*x*.

Справочная система HTML Help 1.*x*

Основой справочной системы формата HTML Help 1.*x* являются *компилированные* HTML-документы, т. е. CHM-файлы (файлы с расширением chm), которые получаются путем компиляции (объединения) HTML-файлов, содержащих справочную информацию, а также файлов иллюстраций в один файл (рис. 8.4).

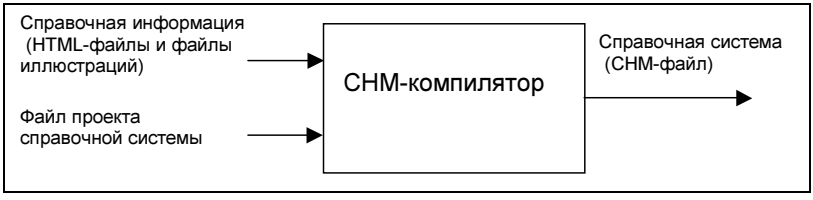


Рис. 8.4. Процесс создания CHM-файла

Процесс создания справочной системы состоит из двух этапов. Сначала надо подготовить справочную информацию, т. е. набрать текст разделов, поместив каждый из них в отдельный HTML-файл. После того как справочная информация будет готова, надо выполнить компиляцию — преобразовать эту информацию в справочную систему (CHM-файл) (можно при помощи CHM-компилятора, входящего в состав Microsoft HTML Help Workshop).

Подготовка справочной информации

Исходным "материалом" для CHM-компилятора Microsoft HTML Help Workshop является справочная информация, представленная в виде набора HTML-файлов.

Подготовить справочную информацию в HTML-формате (создать HTML-файл) можно при помощи любого редактора текста. Наиболее подходящим является Microsoft Word, поскольку этот редактор позволяет сохранить набранный текст в HTML-формате. Можно воспользоваться и редактором, встроенным в Microsoft HTML Help Workshop, но для этого надо знать язык HTML (по крайней мере, его основы).

В простейшем случае вся справочная информация может находиться в одном HTML-файле. Однако если для навигации по справочной системе предполагается использовать вкладку **Содержание**, в которой будут перечислены разделы справочной информации, то информацию каждого раздела нужно поместить в отдельный HTML-файл.

Подготовка справочной информации в Microsoft Word

Чтобы приступить к набору текста раздела, надо в меню **Файл** выбрать команду **Создать ▸ Web-страница**. После того как текст будет набран, заголовок раздела следует оформить стилем **Заголовок 1**, а заголовки подразделов (если они есть) — стилем **Заголовок 2**.

Если для навигации по справочной системе предполагается использовать не только вкладку **Содержание**, но и гиперссылки, то следующее, что надо сделать, — это вставить закладки в те точки документа, в которые предполагаются переходы из других разделов справочной системы.

Чтобы вставить закладку, нужно установить курсор в точку текста, в которой должна быть закладка, из меню **Вставка** выбрать команду **Закладка** и в поле **Имя закладки** диалогового окна **Закладка** ввести имя закладки. Следует обратить внимание на то, что имя закладки должно отражать суть предполагаемого перехода, соответствовать содержимому помеченного фрагмента текста. В имени закладки пробел использовать нельзя (вместо пробела можно поставить символ подчеркивания). Заголовки, оформленные стилем **Заголовок**, помечать закладками не надо. Таким образом, если в создаваемой справочной системе предполагаются переходы только к заголовкам разделов справочной информации, то закладки в документ вставлять не надо.

После того как будут готовы HTML-файлы для всех разделов справочной информации, можно приступить к расстановке ссылок, обеспечивающих навигацию (перемещение между разделами) по справочной системе. Следует обратить внимание: если для навигации по справочной системе предполагается использовать только вкладку **Содержание**, то ссылки вставлять не надо.

Различают ссылки, обеспечивающие навигацию внутри раздела, и ссылки, обеспечивающие переход к другому разделу справочной системы.

Чтобы вставить ссылку, обеспечивающую навигацию внутри раздела, надо выделить фрагмент текста (слово или фразу), при выборе которого должен быть выполнен переход. Потом из меню **Вставка** надо выбрать команду **Гиперссылка**, в появившемся окне **Добавление гиперссылки** сначала щелкнуть на кнопке **Связать с местом в этом документе**, затем — выбрать закладку или заголовок, к которому должен быть выполнен переход.

Если нужно вставить в документ ссылку на раздел справки, который находится в другом файле, то в диалоговом окне **Добавление гиперссылки** нужно сначала щелкнуть на кнопке **Связать с имеющимся файлом**, затем на кнопке **Файл**, и в появившемся стандартном окне выбрать имя нужного HTML-файла. Если надо вставить ссылку на закладку, которая находится в другом файле, то после выбора файла необходимо сделать щелчок на кнопке **Закладка** и в появившемся окне выбрать нужную закладку.

Подготовка справочной информации в HTML Help Workshop

Использование HTML-редактора, входящего в состав HTML Help Workshop, предполагает знание основ языка гипертекстовой разметки (далее приведены краткие сведения об HTML, которых достаточно для создания вполне приличной справочной системы).

Чтобы создать HTML-файл, надо запустить HTML Help Workshop, из меню **File** выбрать команду **New ▸ HTML File** и в появившемся окне **HTML Title** задать название раздела справки, текст которого будет находиться в создаваемом файле (рис. 8.5).

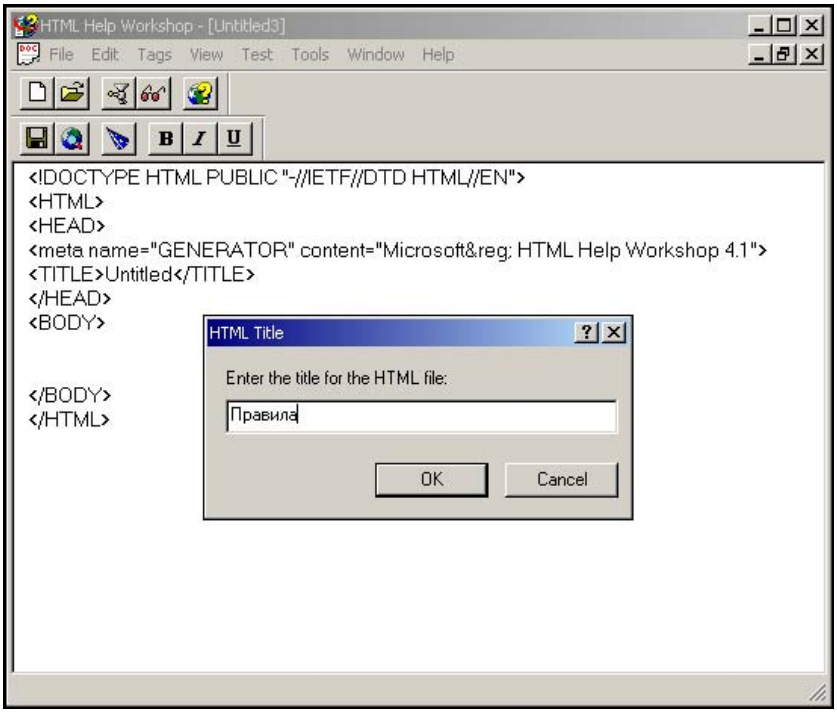


Рис. 8.5. Начало работы над новым HTML-файлом

После щелчка на кнопке **OK** становится доступным окно HTML-редактора, в котором находится шаблон HTML-документа. В этом окне после строки `<BODY>` можно набирать текст.

Основы HTML

HTML-документ представляет собой текст, в который помимо обычного текста включены специальные последовательности символов — *теги*. Тег начинается символом < и заканчивается символом >. Теги используются программами отображения HTML-документов для форматирования текста в окне просмотра (сами теги не отображаются).

Большинство тегов парные. Например, пара тегов `<H2> </H2>` сообщает программе отображения HTML-документа, что текст, который находится между этими тегами, является заголовком

второго уровня и должен быть отображен соответствующим стилем.

В табл. 8.1 представлен минимальный набор тегов, используя которые можно подготовить HTML-файл с целью дальнейшего его преобразования в CHM-файл справочной системы.

Таблица 8.1. *HTML-теги*

Тег	Пояснение
<TITLE> *Название* </TITLE>	Задает название HTML-документа. Программы отображения HTML-документов, как правило, выводят название документа в заголовке окна, в котором документ отображается. Если название не задано, то в заголовке окна будет выведено название файла
<BODY BACKGROUND = "*Файл*"	Параметр BACKGROUND задает фоновый рисунок,
BGCOLOR="*Цвет*"	BGCOLOR — цвет фона,
TEXT="*Цвет*">	TEXT — цвет символов HTML-документа
<BASEFONT FACE="*Шрифт*"	Задает основной шрифт, который используется для отображения текста:
SIZE=*n*>	FACE — название шрифта, SIZE — размер в относительных единицах. По умолчанию значение параметра SIZE равно 3. Размер шрифта заголовков (см. тег <H>) берется от размера, заданного параметром SIZE
<H1> </H1>	Определяет текст, находящийся между тегами <H1> и </H1>, как заголовок уровня 1. Пара тегов <H2></H2> определяет заголовок второго уровня, а пара <H3></H3> — третьего
 	Конец строки. Текст, находящийся после этого тега, будет выведен с начала новой строки

Таблица 8.1 *(окончание)*

Тег	Пояснение
<P> </P>	Текст, находящийся внутри этих тегов, является параграфом
<B> </B>	Текст, находящийся внутри этой пары тегов, будет выделен полужирным
<I> </I>	Текст, находящийся внутри этой пары тегов, будет выделен курсивом
<A NAME="*Закладка*"> </A>	Помечает фрагмент документа закладкой. Имя закладки задает параметр NAME. Это имя используется для перехода к закладке
<A HREF="*Файл.htm#Закладка*"> </A>	Выделяет фрагмент документа как гиперссылку. При выборе этой гиперссылки происходит перемещение к закладке, имя которой указано в параметре HREF
<IMG SRC="*Иллюстрация*">	Выводит иллюстрацию, имя файла которой указано в параметре SRC
<!-- -->	Комментарий. Текст, находящийся между дефисами, на экран не выводится

Набирается HTML-текст обычным образом. Теги можно набирать как прописными, так и строчными буквами. Однако чтобы лучше была видна структура документа, рекомендуется записывать все теги строчными (большими) буквами. Следующее, на что надо обратить внимание, — это то, что программы отображения HTML-документов игнорируют "лишние" пробелы и другие "невидимые" символы (табуляция, новая строка). Чтобы фрагмент документа начинался с новой строки, в конце предыдущей строки надо поставить тег
, а чтобы между строками текста появилась пустая строка, в HTML-текст надо вставить два тега
 подряд.

Работая с HTML-редактором в программе HTML Help Workshop, в процессе набора HTML-текста, можно увидеть, как будет выглядеть набираемый текст. Для этого надо из меню **View** выбрать команду **In Browser** или щелкнуть на командной кнопке с изображением стандартного значка Internet Explorer.

В качестве примера на рис. 8.6 приведен текст раздела **Правила** справочной системы программы **Сапер**.

```
<HTML>
<TITLE>Правила</TITLE>
<BODY BGCOLOR=#FFFFFF BACKGROUND ="">
<BASEFONT FACE="Arial" SIZE=2>
<A NAME="Правила"><H2>Правила</H2></A><P>Игровое поле состоит из клеток, в каждой из которых может быть мина. Цель игры - найти все мины и пометить их флажками.</P>
<P>Используя кнопки мыши, игрок может открыть клетку или поставить в нее флажок, указав тем самым, что в клетке находится мина. Клетка открывается щелчком левой кнопки мыши, флажок устанавливается щелчком правой. Если в клетке, которую открыл игрок, есть мина, то происходит взрыв (сапер ошибся, а он, как известно, ошибается только один раз), и игра заканчивается. Если в клетке мины нет, то в этой клетке появляется число, соответствующее количеству мин, находящихся в соседних клетках. Анализируя информацию о количестве мин в клетках, соседних с уже открытыми, игрок может обнаружить (т. е. пометить флажками) все мины. Ограничений на количество клеток, помеченных флажками, нет. Однако для завершения игры (выигрыша) флажки должны быть установлены только в тех клетках, в которых есть мины. Ошибочно установленный флажок можно убрать, щелкнув правой кнопкой мыши в клетке, в которой он находится.</P>
<I>См.</I><BR>
<A HREF="saper_01.htm#Сапер">Сапер</A><BR>
<A HREF="saper_03.htm#О_программе">О программе</A><BR>
</BODY>
</HTML>
```

Рис. 8.6. HTML-текст раздела справочной системы

Создание справочной системы

После того как справочная информация будет подготовлена, можно приступить к непосредственному ее созданию. Для этого надо запустить HTML Help Workshop, из меню **File** выбрать команду **New ▸ Project**, в окне **New Project** задать имя файла проекта создаваемой справочной системы (рис. 8.7). После щелчка на кнопке **Далее** в этом, а затем и в следующем окне, активизируется окно **HTML Help Workshop**, которое должно выглядеть так, как показано на рис. 8.8.

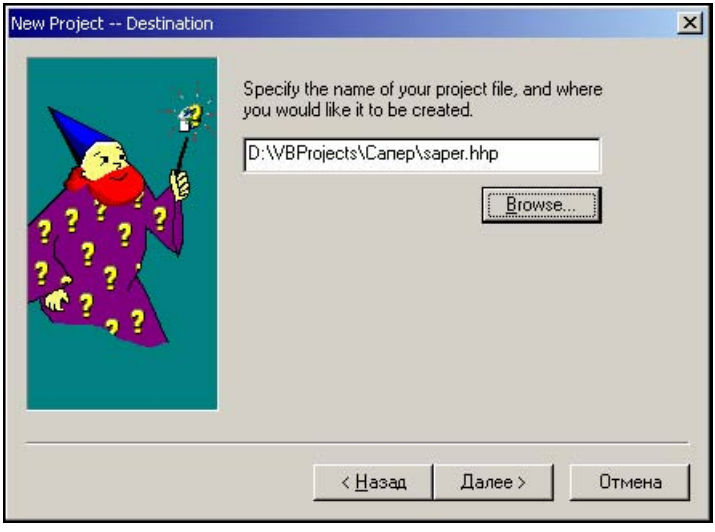


Рис. 8.7. Начало работы над новым проектом

Первое, что надо сделать, — сформировать раздел **[FILES]**, в котором перечисляются файлы, содержащие справочную информацию. В начале работы над новым проектом этот раздел не содержит ни одного элемента, поэтому на вкладке **Project** не отображается. Чтобы сформировать раздел **[FILES]**, надо щелкнуть на кнопке **Add/Remove topic files**, на экране появится окно **Topic Files** (рис. 8.9). В этом окне надо щелкнуть на кнопке **Add** и в стандартном диалоговом окне **Открыть** выбрать HTML-файл раздела справки.

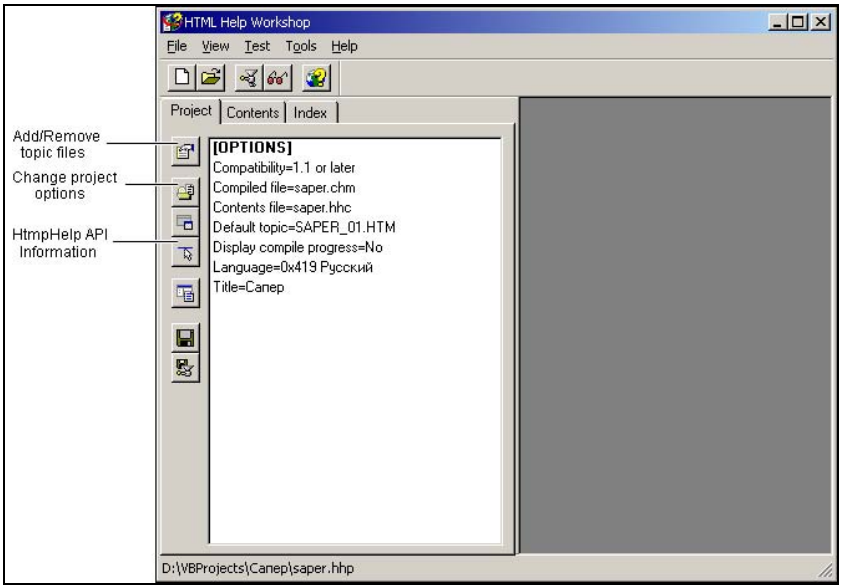


Рис. 8.8. Окно **HTML Help Workshop** в начале работы над новым проектом

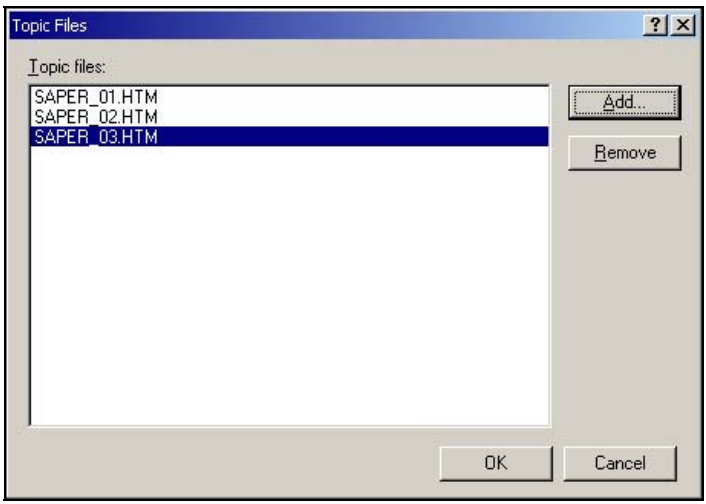


Рис. 8.9. Диалоговое окно **Topic Files**

Если справочная информация распределена по нескольким файлам, то операцию добавления нужно повторить несколько раз. После того как в диалоговом окне **Topic Files** будут перечислены все необходимые для создания справочной информации HTML-файлы, нужно щелкнуть на кнопке **OK**. В результате этих действий в файле проекта появится раздел **[FILES]**, в котором будут перечислены HTML-файлы, необходимые для создания справочной системы (рис. 8.10).

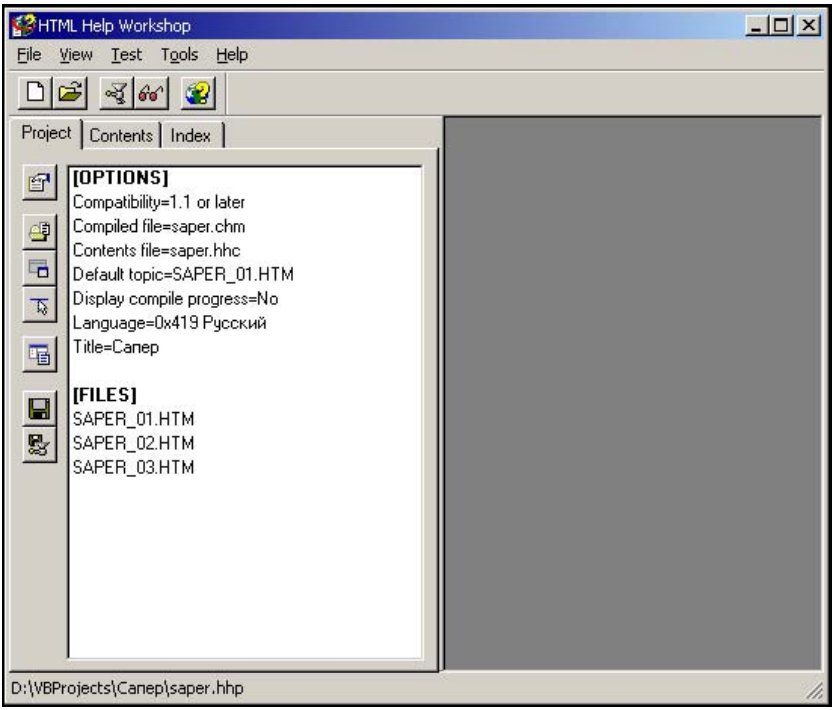


Рис. 8.10. В разделе **[FILES]** перечислены файлы, используемые для создания CHM-файла

Следующее, что надо сделать, — это задать главный (стартовый) раздел и заголовок окна справочной системы. Заголовок и имя файла главного раздела вводятся соответственно в поля **Title** и **Default file** вкладки **General** диалогового окна **Options** (рис. 8.11),

которое появляется в результате щелчка на кнопке **Change project options** (см. рис. 8.8).

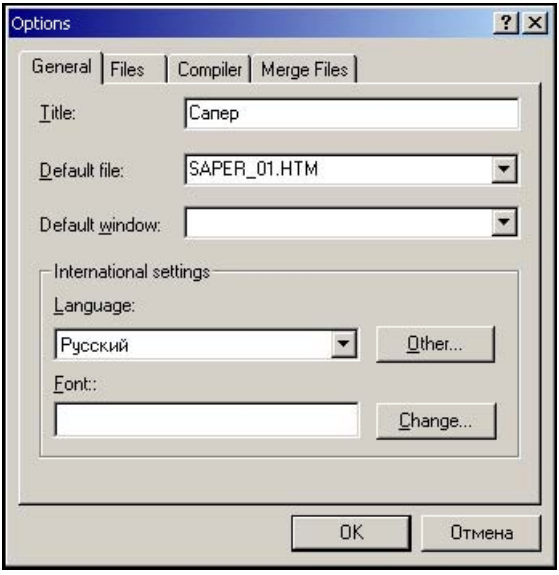


Рис. 8.11. В диалоговом окне **Options** надо задать заголовок окна справочной системы и файл главного раздела

Чтобы программа, использующая справочную систему, могла получить доступ к конкретному разделу справочной информации, необходимо для каждого раздела определить *идентификатор раздела* (TopicID). Для этого необходимо при помощи редактора текста (например, Блокнота) сначала создать H-файл, содержащий строки вида:

```
#define Файл Идентификатор ; комментарий
```

где:

- `Файл` — имя HTML-файла (без расширения), в котором находится справочная информация раздела;
- `Идентификатор` — идентификатор раздела справочной информации (целое число);
- `комментарий` — необязательный пояснительный текст.

В качестве примера в листинге 8.1 приведено содержимое Н-файла справочной системы программы Сапер.

Листинг 8.1. Файл saper.h содержит идентификаторы разделов справочной информации

```
#define saper_01 1; Сапер (общая информация)
#define saper_02 2; Правила
#define saper_03 3; О программе
```

После того как Н-файл будет готов, надо щелчком мыши на кнопке **HtmlHelp API Information** (см. рис. 8.8) открыть одноименное окно, в котором щелкнуть на кнопке **Header files** и выбрать в стандартном окне созданный Н-файл. В результате этих действий на вкладке **MAP** появится ссылка на Н-файл (рис. 8.12), а после того как окно будет закрыто — раздел (секция) **[MAP]** в окне проекта.

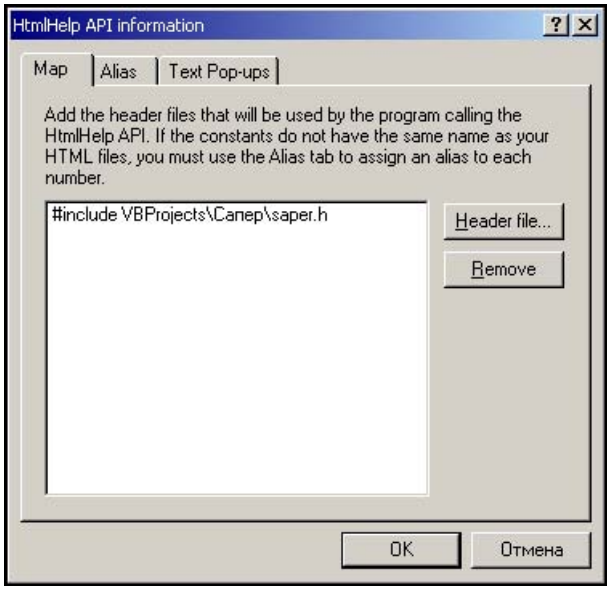


Рис. 8.12. Диалоговое окно **HtmlHelp API Information**

Если для навигации по справочной системе предполагается использовать вкладку **Contents** (Содержание), то надо создать *файл контекста*. Чтобы это сделать, нужно щелкнуть на вкладке **Contents**, подтвердить создание нового файла и задать имя файла контекста, в качестве которого можно использовать имя проекта. В результате станет доступной вкладка **Contents** (рис. 8.13), в которую нужно ввести содержание — названия разделов справочной системы.

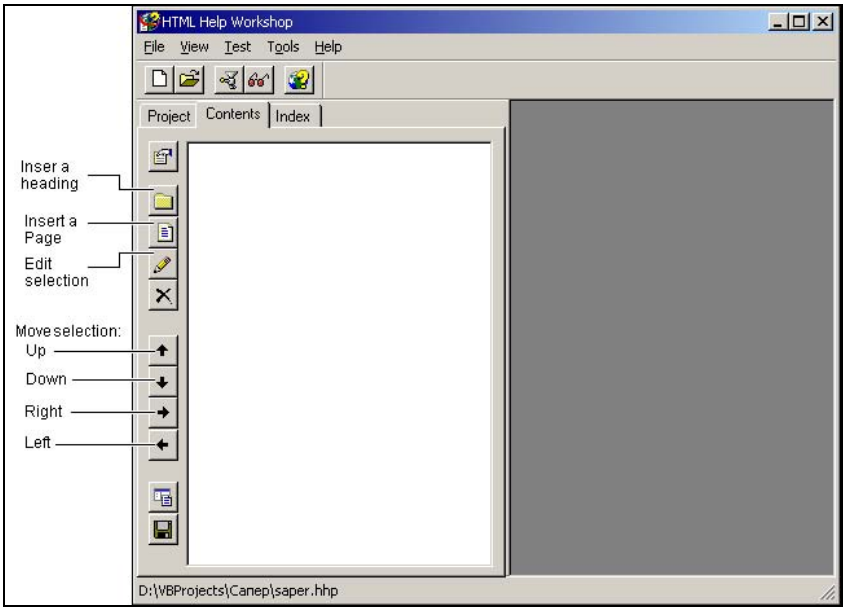


Рис. 8.13. Вкладка **Contents** диалогового окна **HTML Help Workshop**

Чтобы на вкладку **Contents** добавить элемент, соответствующий разделу справочной системы, нужно щелкнуть на кнопке **Insert a heading**, в поле **Entry title** появившегося диалогового окна **Table of Contents Entry** (рис. 8.14) ввести название раздела и щелкнуть на кнопке **Add**. На экране появится окно **Path or URL** (рис. 8.15). В поле **HTML titles** этого окна перечислены названия разделов (заголовки HTML-файлов) справочной информации, которая находится во включенных в проект файлах (имена этих

файлов указаны в разделе **[FILES]** вкладки **Project**). Если вместо названия раздела справочной информации указано имя файла, то это значит, что в файле нет тега <TITLE>. Выбрав нужный файл, нужно щелкнуть на кнопке **OK**. В результате перечисленных действий на вкладке **Contents** появится строка с названием раздела справочной информации.

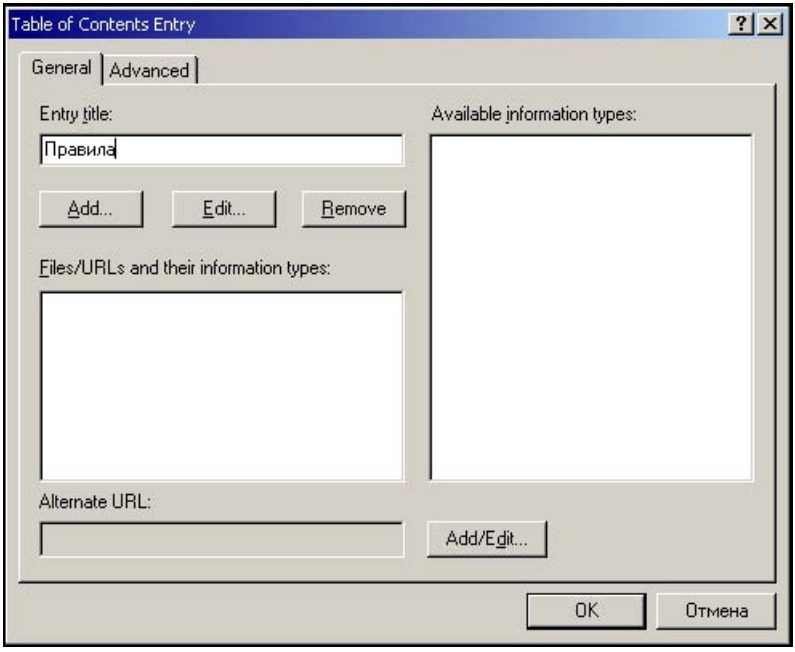


Рис. 8.14. Формирование вкладки **Contents** — добавление элемента в список разделов

Если нужно изменить значок, соответствующий добавленному разделу, то следует щелкнуть на кнопке **Edit selection** и, используя список **Image index** вкладки **Advanced** окна **Table of Contents**, выбрать нужный значок (обычно рядом с названием раздела или подраздела изображена книжка).

Подраздел добавляется точно так же, как и раздел, но после его добавления нужно щелкнуть на кнопке **Move selection right**, в

результате чего уровень заголовка понизится, т. е. раздел станет подразделом.

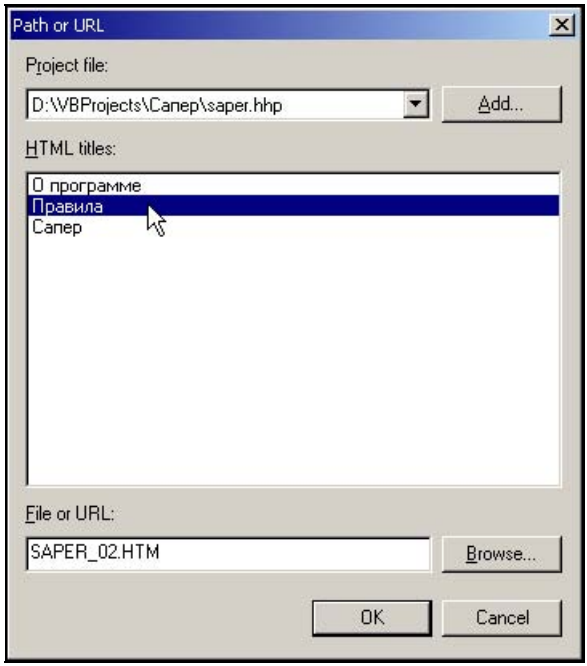


Рис. 8.15. Выбор файла, соответствующего элементу списка разделов

Элементы содержания, соответствующие темам справочной информации, добавляются аналогичным образом, но процесс начинается щелчком на кнопке **Insert a page**.

Иногда возникает необходимость изменения порядка следования элементов списка содержания или уровня иерархии элемента списка. Сделать это можно при помощи командных кнопок, на которых изображены стрелки. Кнопки **Move selection up** и **Move selection down** перемещают выделенный элемент списка, соответственно, вверх и вниз. Кнопка **Move selection right** перемещает выделенный элемент вправо, т. е. делает его подчиненным пре-

дыдущему элементу списка. Кнопка **Move selection left** выводит элемент из подчиненности предыдущему элементу.

В качестве примера на рис. 8.16 приведена вкладка **Contents** справочной системы программы Сапер.

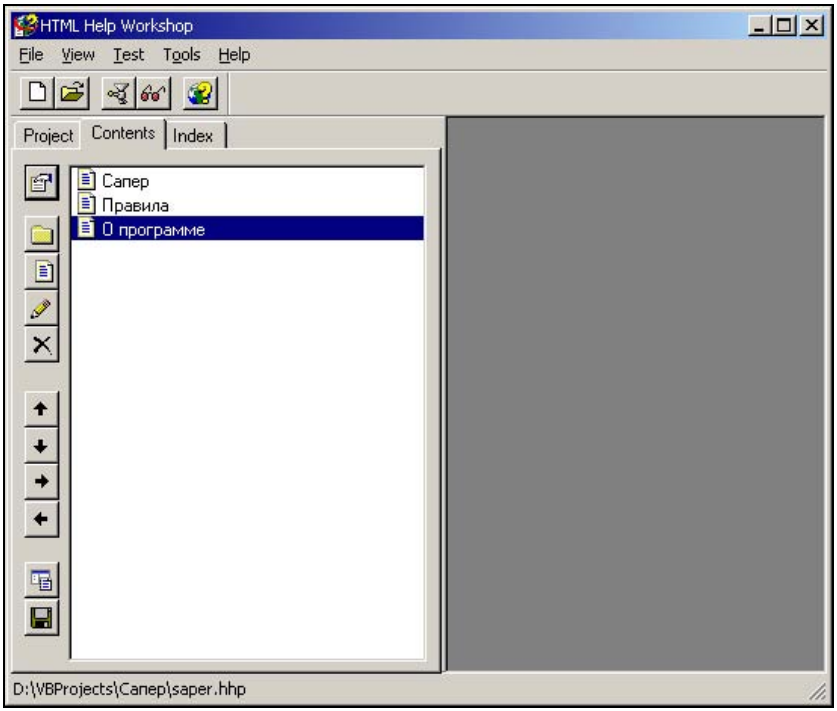


Рис. 8.16. Вкладка **Contents** содержит названия разделов справочной системы

Компиляция

После того как будут определены файлы, в которых находится справочная информация (сформирован раздел **[FILES]**), подготовлена информация для формирования вкладки **Contents** (создан файл контекста), можно выполнить компиляцию, т. е. преобразовать исходную справочную информацию в файл справочной системы (CHM-файл).

Чтобы выполнить компиляцию, надо в меню **File** выбрать команду **Compile**, затем в появившемся окне **Create a compiled file** установить переключатель **Automatically display compiled help file when done** (после компиляции показать созданный файл справки) и щелкнуть на кнопке **Compile** (рис. 8.17). В результате этого будет создан файл справки, а на экране появится окно справочной системы, в котором будет выведена информация главного раздела.

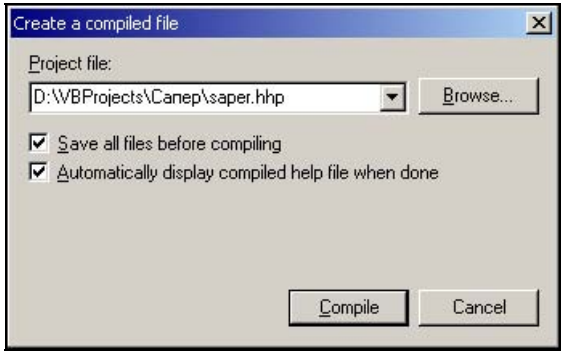


Рис. 8.17. Диалоговое окно **Create a compiled file**

Отображение справочной информации

Существуют два способа активизации процесса отображения справочной информации во время работы программы. Первый — нажать клавишу <F1>, а второй — выбрать команду в меню **Справка** или сделать щелчок на соответствующей командной кнопке.

Чтобы окно справочной системы появилось в результате нажатия клавиши <F1>, надо записать имя файла справочной информации в поле **Help File Name** вкладки **General** окна **Project Properties** (рис. 8.18), которое становится доступным в результате выбора в меню **Project** команды **Properties**.

Пользователь может нажать клавишу <F1> в разные моменты работы с программой, например, во время ввода исходных данных, когда курсор находится в одном из полей редактирования,

или сразу после запуска программы. Очевидно, что справочная информация, отображаемая в окне справочной системы, при этом должна быть разной. Чтобы в результате нажатия клавиши <F1> (в зависимости от того, какой компонент формы в данный момент активен) появился конкретный раздел справочной информации, необходимо в свойство `HelpContextID` компонента записать идентификатор раздела справочной информации. Вспомните, что идентификаторы разделов справочной информации определены в H-файле, а ссылка на этот файл находится в MAP-секции файла проекта справочной системы. Следует обратить внимание на то, что если значение свойства `HelpContextID` компонента не задано, то в окне справочной системы отображается содержимое раздела, заданного свойством `HelpContextID` формы или проекта.

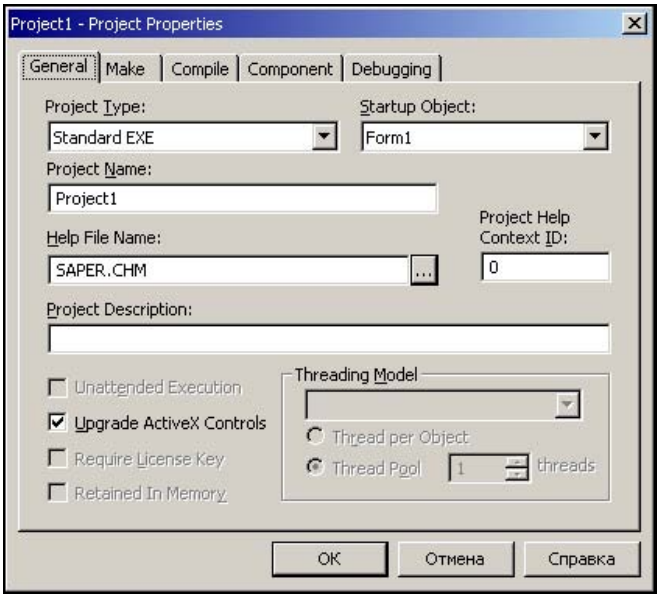


Рис. 8.18. Имя файла справочной информации надо ввести в поле **Help File Name**

Если справочная информация должна появиться в результате щелчка на командной кнопке или после выбора команды в ме-

ню, то необходимо создать процедуру обработки соответствующего события, обеспечивающую запуск утилиты hh.exe. В качестве параметров утилите hh.exe надо передать: идентификатор раздела справочной информации, указав перед ним ключ -mapid, имя файла справочной информации и идентификатор типа окна справочной информации. Запуск утилиты hh.exe, как и любой другой программы, можно выполнить при помощи инструкции Shell, указав в качестве параметров командную строку, обеспечивающую запуск программы (в том виде, как она набирается в стандартном окне **Запуск программы**), а также тип окна, в котором будет работать программа. Например, инструкция, обеспечивающая запуск программы hh.exe и отображение в ее окне справочной информации пятого раздела справочной информации, которая находится в файле konvertor.chm, выглядит так:

```
Shell "hh.exe -mapid 1 konvertor.chm", vbNormalFocus
```

Справочная информация может стать доступной и в результате щелчка на кнопке **Справка** в окне сообщения, которое выводит процедура (функция) MsgBox. Чтобы пользователь мог воспользоваться этим способом доступа к справочной информации, в инструкции вызова функции MsgBox надо указать, что в окне сообщения необходимо отобразить кнопку **Справка**, задать имя файла справочной информации и идентификатор раздела.

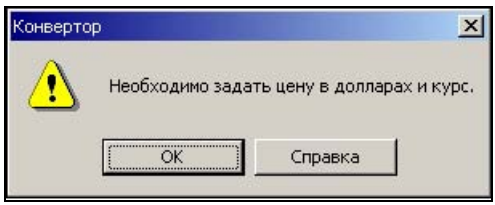


Рис. 8.19. Справочная информация появится в результате щелчка на кнопке **Справка**

В качестве примера на рис. 8.19 приведено окно сообщения, которое отображает процедура

```
MsgBox "Необходимо задать цену в долларах и курс.", _
        vbExclamation + vbMsgBoxHelpButton, _
```

```
"Конвертор", _
"konvertor.chm", 2
```

Следующая программа (ее форма приведена на рис. 8.20) демонстрирует все перечисленные способы доступа к справочной информации.

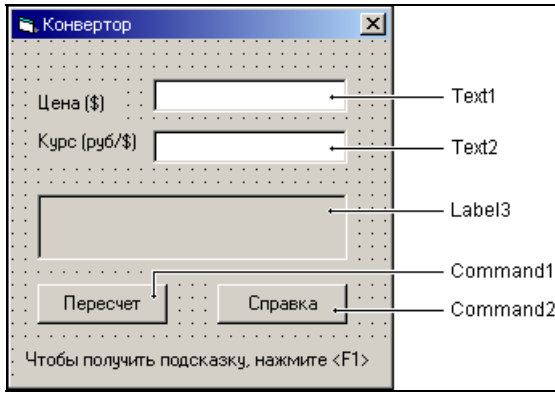


Рис. 8.20. Форма программы Конвертор

Справочная информация программы Конвертор состоит из пяти разделов (рис. 8.21).

Разделы **Цена** и **Курс** отображаются в результате нажатия клавиши <F1>, если курсор находится, соответственно, в поле **Цена** или **Курс**. Раздел **Конвертор** (общая информация о программе) отображается, если пользователь сделал щелчок на кнопке **Справка** или нажал клавишу <F1>, при условии, что курсор не находится ни в одном из полей ввода исходных данных.

Раздел **Ввод чисел** отображается в результате щелчка на кнопке **Справка** в окне сообщения, которое появляется, если исходные данные введены неверно. Значения свойств компонентов, обеспечивающих отображение справочной информации, приведены в табл. 8.2, текст программы — в листинге 8.2, файл konvertor.h — в листинге 8.3.

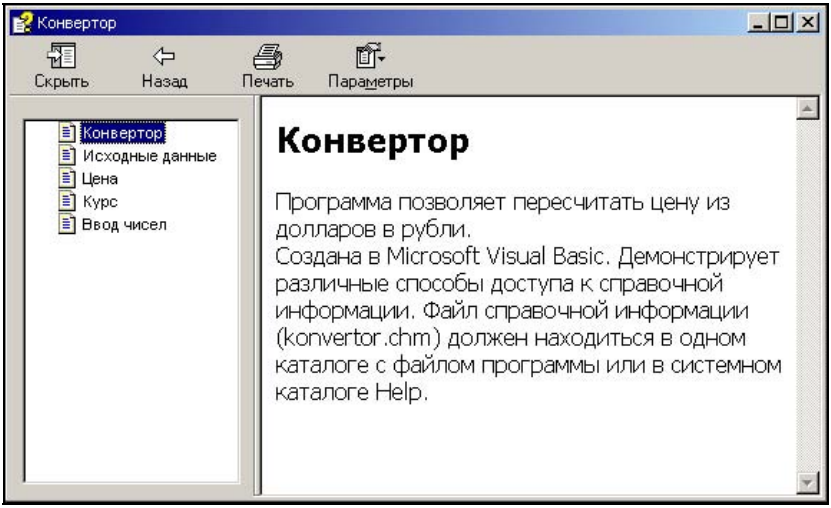


Рис. 8.21. Справочная информация программы Конвертор, состоящая из пяти разделов

Таблица 8.2. *Свойства, обеспечивающие отображение справочной информации*

Свойство	Значение	Замечание
`Help File Name`	`konvertor.chm`	Устанавливается в окне **Project Properties** (команда **Project ▸ Properties**)
`Form1.HelpContextID`	`1`	Раздел **Конвертор**
`Text1.HelpContextID`	`3`	Раздел **Цена**
`Text2.HelpContextID`	`4`	Раздел **Курс**

Листинг 8.2. Конвертор (вывод справочной информации)

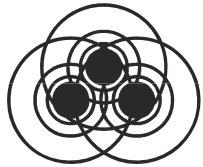
```
' щелчок на кнопке Пересчет
Private Sub Command1_Click()
    Dim usd As Single ' цена в долларах
```

```
    Dim k As Single    ' курс
    Dim rub As Single ' цена в рублях

    usd = Val(Text1.Text)
    k = Val(Text2.Text)

    If usd = 0 Or k = 0 Then
       MsgBox "Необходимо задать цену в долларах и курс.", _
            vbExclamation + vbMsgBoxHelpButton, "Конвертор", _
            "konvertor.chm", 2
    Else
      rub = usd * k
      Label3.Caption = Text1.Text & "$ - " & Str(rub) & "руб."
    End If

End Sub

' щелчок на кнопке Справка
Private Sub Command2_Click()
    Shell "hh.exe -mapid 0 konvertor.chm", vbNormalFocus
End Sub

' нажатие клавиши в поле Цена
' при нажатии функциональной клавиши, например F1,
' событие KeyPress не возникает)
Private Sub Text1_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then Text2.SetFocus
End Sub

' нажатие клавиши в поле Курс
Private Sub Text2_KeyPress(KeyAscii As Integer)
    If KeyAscii = 13 Then Command1.SetFocus
End Sub
```

Листинг 8.3. konvertor.h

```
#define konvertor_01 1;      Конвертор
#define konvertor_02 2;      Исходные данные
#define konvertor_03 3;      Цена
#define konvertor_04 4;      Курс
#define konvertor_05 5;      Ввод чисел
```

Глава 9



Программа установки

Чтобы установить созданную в Visual Basic программу на другой компьютер, сначала надо скопировать все необходимые файлы на промежуточный носитель (дискету или флэш-накопитель). Затем на компьютере пользователя надо создать каталог для устанавливаемой программы, скопировать в созданный каталог файлы с промежуточного носителя и добавить в меню **Пуск** команду, обеспечивающую запуск установленной программы. Очевидно, что большинство пользователей не смогут самостоятельно выполнить установку программы. Поэтому для установки приложения на компьютер пользователя разработана специальная программа, которая не только копирует всю необходимую информацию с промежуточного носителя, но и выполняет необходимые настройки. Обычно программа установки находится в корневом каталоге носителя (называется setup.exe), и, если промежуточным носителем является CD, то она, как правило, запускается автоматически.

Создать программу установки можно точно так же, как и любую другую программу. Однако лучше воспользоваться одной из утилит, предназначенных для создания инсталляционных программ.

В состав Visual Basic входит утилита Package and Deployment Wizard, предназначенная для создания программы установки, однако она не совсем соответствует современным требованиям. Поэтому для таких целей лучше использовать современные средства (например, утилиту из Microsoft Visual Studio или пакет InstallShield Express). Однако следует обратить внимание на то, что указанные средства являются профессиональными инстру-

ментами и их использование для создания программ установки простых приложений не оправдано (например, размер программы установки может превышать размер устанавливаемой программы в сотни раз). Поэтому для создания программ установки простых приложений лучше использовать более "легкий" инструмент — входящую в состав Windows утилиту IExpress (iexpress.exe), тем более что она всегда "под рукой".

Утилита IExpress

Утилита IExpress (iexpress.exe) входит в состав операционной системы Windows, она позволяет создать программу установки — самораспаковывающийся архив.

Процесс создания программы установки рассмотрим на примере. Пусть необходимо создать такую программу для игры **Сапер**.

Предварительно нужно выполнить подготовительную работу — составить список файлов, которые должны быть установлены на компьютер пользователя, а также, используя редактор текста, подготовить файл краткой справки (readme-файл). Список файлов программы Сапер, которые должны быть перенесены на компьютер пользователя, приведен в табл. 9.1.

Таблица 9.1. *Файлы программы Сапер, которые нужно установить на компьютер пользователя*

Файл	Назначение	Куда устанавливать
Saper.exe	Программа	Program Files \ Saper
Saper.chm	Файл справочной информации	Program Files \ Saper
Readme.txt	Краткая справка о программе	Program Files \ Saper

Поскольку утилита IExpress является составной частью операционной системы и файл iexpress.exe находится в каталоге System32, то для ее запуска достаточно набрать имя файла в окне **Запуск программы** (рис. 9.1). Вид окна после запуска утилиты приведен на рис. 9.2.

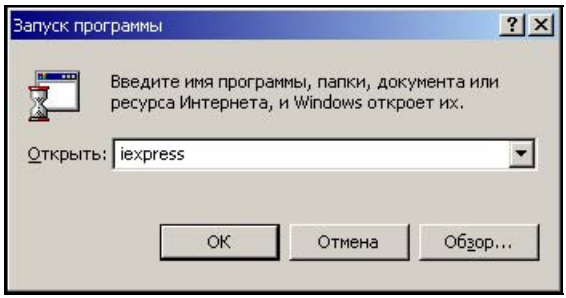


Рис. 9.1. Запуск утилиты IExpress

Рис. 9.2. Начало работы над новым проектом

Утилита IExpress представляет собой типичную *мастер-программу* (Wizard), которая предоставляет пользователю последовательность окон, в поля которых надо ввести информацию. Переход к следующему окну (шагу) осуществляется в результате щелчка на кнопке **Далее**. На каждом шаге имеется возможность вернуться к предыдущему окну (для этого надо щелкнуть на кнопке **Назад**). После того как будет введена вся необходимая

информация, мастер выполнит определенное действие — в данном случае сгенерирует программу установки.

Создание программы установки

Процесс создания программы установки при помощи IExpress состоит из 9 шагов. Переход к следующему шагу выполняется в результате щелчка на кнопке **Далее**.

Шаг 1

На первом шаге (рис. 9.3) надо задать действие, которое должно быть выполнено в результате запуска программы, созданной мастером. Так как необходимо создать программу установки, которая должна скопировать файлы на компьютер пользователя, то надо установить переключатель **Extract files only** (Извлечь файлы).

Рис. 9.3. Начало работы над программой установки

Шаг 2

На втором шаге (рис. 9.4) надо задать заголовок окна, которое увидит пользователь в результате запуска программы установки. Обычно в заголовке окна отображается слово **Setup**.

Рис. 9.4. Ввод заголовка окна программы установки

Шаг 3

На третьем шаге (рис. 9.5) надо задать текст, который увидит пользователь в окне, которое появится в результате запуска программы установки. Обычно это краткая информация об устанавливаемой программе и запрос о необходимости установки программы. В данном случае в окне **Confirmation prompt** надо установить переключатель **Prompt user with** (Запрос, подсказка пользователю) и ввести в поле редактирования текст: `Установка программы Сапер. Выполнить?` Если установить переключатель **No prompt** (Не запрашивать), то установка программы будет выполнена в тот каталог, который задаст разработчик программы, причем безусловно (у пользователя после

запуска программы setup не будет возможности отказаться от установки, а также он не сможет задать каталог, в который надо установить программу).

Рис. 9.5. Ввод текста запроса подтверждения установки программы

Шаг 4

На четвертом шаге (рис. 9.6) можно задать имя файла лицензионного соглашения. Для этого надо установить переключатель **Display a license** и задать имя RFTF-файла, в котором находится текст лицензионного соглашения. Если имя файла задано, то во время установки программы пользователь увидит стандартное окно с текстом лицензионного соглашения. Прочитав текст и выбрав переключатель **я принимаю соглашение**, он сможет продолжить установку программы, а в случае выбора переключателя **я не принимаю соглашение** — отказаться от установки. Если установить переключатель **Do not display license**, окно лицензионного соглашения во время установки, естественно, не появится.

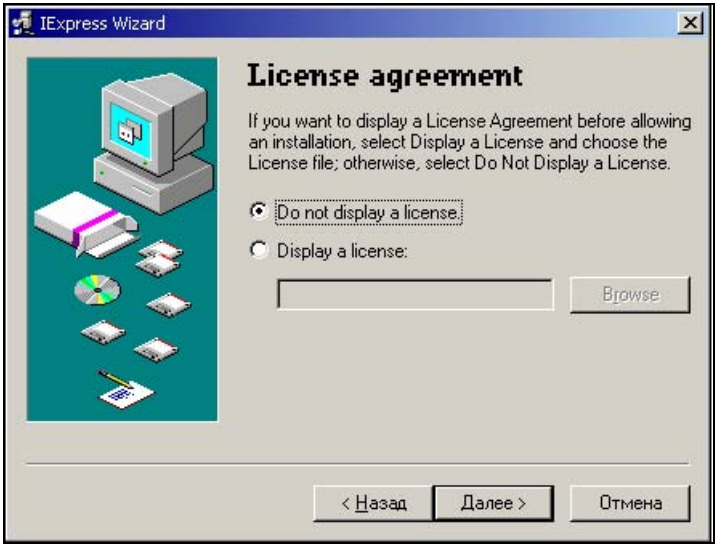


Рис. 9.6. Установка переключателя, определяющего необходимость отображения текста лицензионного соглашения

Шаг 5

На пятом шаге (рис. 9.7) надо задать файлы, входящие в устанавливаемую программу. Для этого нужно щелкнуть на кнопке **Add** и в стандартном окне **Открыть** выбрать необходимые файлы. Следует обратить внимание: если файлы находятся в одном каталоге, то за один раз можно добавить сразу несколько файлов. Для этого надо щелкнуть на кнопке **Add**, открыть каталог, в котором находятся нужные файлы, нажать клавишу <Ctrl> и, удерживая эту клавишу, щелчком мыши отметить нужные файлы.

Шаг 6

На шестом шаге можно задать вид окна программы установки во время копирования файлов. Окно может быть стандартного размера (Default), развернуто на весь экран (Maximized), свернуто (Minimized) или скрыто (Hidden). В зависимости от желания разработчика можно выбрать любой тип, но лучше выбрать Default.

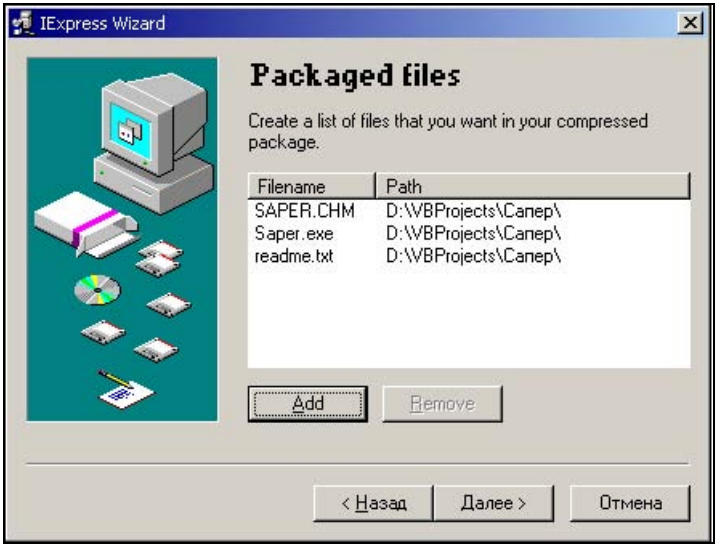


Рис. 9.7. Выбор файлов, которые необходимо перенести на компьютер пользователя

Шаг 7

На седьмом шаге (рис. 9.8) надо задать текст сообщения, которое будет выведено после того, как программа будет установлена.

Шаг 8

На восьмом шаге (рис. 9.9) надо задать имя файла программы установки. По умолчанию пакет называется setup.exe. Именно этот файл надо будет поместить на промежуточный носитель, чтобы передать программу пользователю. Следует обратить внимание: никакие другие файлы для установки программы, кроме созданного setup.exe, не нужны.

Шаг 9

На девятом шаге (рис. 9.10) мастер предлагает сохранить файл проекта программы, чтобы иметь возможность внести изменения в программу установки, а не выполнять все пройденные шаги заново.

Рис. 9.8. Ввод текста информационного сообщения, которое должно появиться после установки программы

Рис. 9.9. Ввод имени файла программы установки

Рис. 9.10. Установка переключателя, определяющего необходимость сохранения файла проекта программы установки в SED-файле

Рис. 9.11. Завершающее диалоговое окно мастера установки

На этом процесс подготовки к созданию программы заканчивается. Щелчок на кнопке **Далее** в этом и следующем (информационном) окне активизирует процесс создания программы установки. После того как файл setup.exe будет создан, на экране появится последнее окно мастера (рис. 9.11).

Установка программы

Чтобы проверить, как работает программа установки, надо запустить файл setup.exe, созданный мастером IExpress (каталог, в котором находится этот файл, был задан на восьмом шаге). На рис. 9.12, *а*, 9.12, *б*, 9.12, *в*, 9.12, *г* приведены окна, которые отображаются в процессе установки программы Сапер.

Здесь еще раз следует обратить внимание: чтобы программа, созданная в Visual Basic, могла работать на другом компьютере, там должны быть установлены используемые программой динамические библиотеки (рис. 9.13).

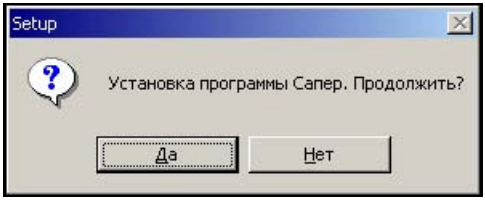


Рис. 9.12, *а*. Запрос на продолжение установки программы

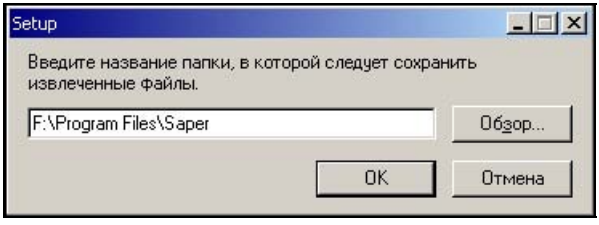


Рис. 9.12, *б*. Ввод названия папки, в которой следует сохранить извлеченные файлы

Рис. 9.12, *в*. Запрос на создание папки с нужным именем при ее отсутствии

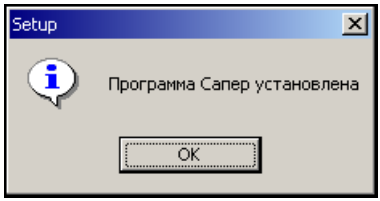


Рис. 9.12, *г*. Сообщение о завершении установки программы

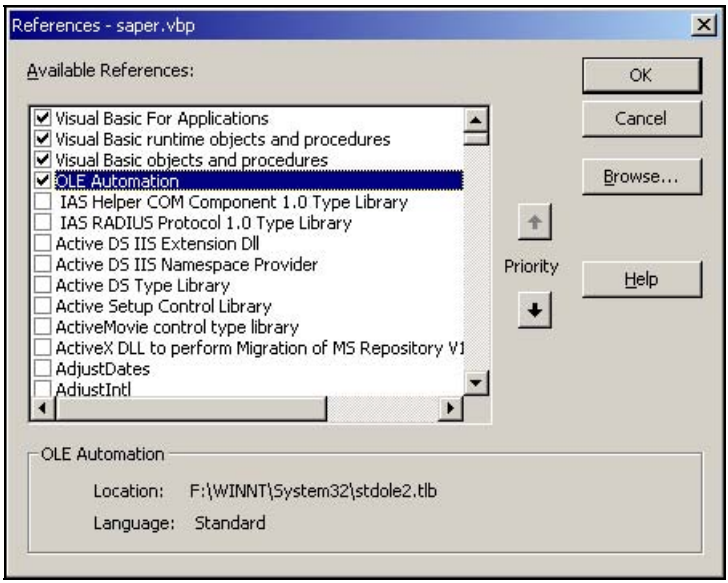
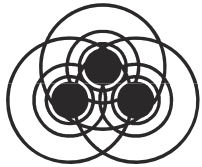


Рис. 9.13. Библиотеки, используемые программой Сапер, помечены флажками

В большинстве случаев эти библиотеки на компьютере имеются (некоторые являются составной частью операционной системы, а некоторые — других продуктов Microsoft). Однако возможна ситуация, когда нужной библиотеки на компьютере пользователя нет. В этом случае придется установить и зарегистрировать нужную библиотеку вручную. Узнать, какие библиотеки использует программа, можно, выбрав в меню **Project** команду **References**, а зарегистрировать их надо при помощи утилиты regsvr32.

Глава 10



Примеры программ

Система проверки знаний

Тестирование широко применяется для оценки уровня знаний в учебных заведениях, при приеме на работу, для оценки квалификации персонала учреждений, т. е. практически во всех сферах деятельности человека. Испытуемому предлагается ряд вопросов (тест), на которые он должен ответить. Обычно к каждому вопросу дается несколько вариантов ответа, из которых надо выбрать правильный. После того как испытуемый ответит на все вопросы, подсчитывается количество правильных ответов и на основе этой информации выставляется оценка.

Рассмотрим программу Экзаменатор, которая позволяет автоматизировать процесс тестирования.

Требования к программе

В результате анализа используемых на практике методик тестирования были сформулированы следующие требования к программе:

- программа должна обеспечивать работу с тестом произвольной длины, т. е. не должно быть ограничений на количество вопросов в тесте;
- для каждого вопроса может быть до четырех возможных вариантов ответа;
- вопрос может сопровождаться иллюстрацией;

- результат тестирования должен быть отнесен к одному из четырех уровней (например, "отлично", "хорошо", "удовлетворительно" или "плохо");
- вопросы теста должны находиться в текстовом файле;
- в программе должна быть заблокирована возможность возврата к предыдущему вопросу. Если вопрос предложен, то на него должен быть дан ответ.

На рис. 10.1 приведен пример окна программы тестирования во время ее работы.

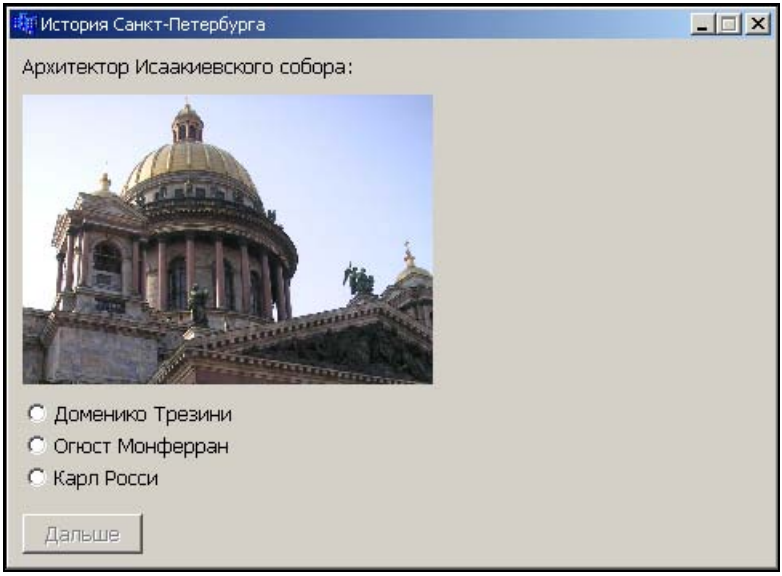


Рис. 10.1. Окно программы Экзаменатор: испытуемый должен выбрать правильный ответ

Файл теста

Тест представляет собой последовательность вопросов, на которые испытуемый должен ответить путем выбора правильного ответа из нескольких предложенных вариантов.

Файл теста состоит из:

- ❒ заголовка;
- ❒ раздела оценок;
- ❒ раздела вопросов.

Заголовок содержит название теста и общую информацию о тесте, например, о его назначении. Состоит заголовок из двух абзацев (строк): первый абзац — название теста, второй — вводная информация.

Вот пример заголовка:

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и
архитектурных сооружениях Санкт-Петербурга. Вы должны из
предложенных вариантов ответа выбрать правильный.
```

Здесь следует обратить внимание на то, что абзац — это последовательность символов, заканчивающаяся символом "конец абзаца", который добавляется в текст в результате нажатия клавиши <Enter>. В окне редактора абзац может представлять собой несколько строк текста. Тем не менее при чтении текста из файла инструкция `Line Input` считывает весь абзац.

За заголовком следует раздел оценок, в котором указывается количество баллов, необходимое для достижения уровня, и сообщение, информирующее испытуемого о достижении уровня. В простейшем случае сообщение — это оценка. Для каждого уровня надо указать балл (количество правильных ответов) и в следующей строке сообщение. Вот пример раздела оценок:

```
10
Отлично
8
Хорошо
6
Удовлетворительно
5
Плохо
```

За разделом оценок следует раздел вопросов теста.

Каждый вопрос начинается текстом вопроса, за которым (в следующей строке) находятся три целых числа:

- первое число — это количество альтернативных ответов;
- второе — номер правильного ответа;
- третье — признак наличия к вопросу иллюстрации. Если вопрос сопровождается иллюстрацией, то значение признака должно быть равно единице, если нет, то нулю. Если к вопросу есть иллюстрация, то в следующей строке должно быть имя файла иллюстрации.

Далее следуют альтернативные ответы, каждый из которых должен представлять собой один абзац текста.

Вот пример вопроса:

```
Архитектор Зимнего дворца
3 2 1
herm.bmp
Бартоломео Растрелли
Карл Росси
Огюст Монферран
```

В приведенном примере к вопросу даны три варианта ответа, правильным является второй ответ (Карл Росси). К вопросу есть иллюстрация (третье число во второй строке, равное единице), которая находится в файле herm.bmp.

Ниже в качестве примера приведен текст файла вопросов для контроля знания истории памятников и архитектурных сооружений Санкт-Петербурга.

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и
архитектурных сооружениях Санкт-Петербурга. Вы должны из
предложенных вариантов ответа выбрать правильный.
7
Вы прекрасно знаете историю Санкт-Петербурга!
6
Вы много знаете о Санкт-Петербурге, но на некоторые вопросы
ответили не верно.
5
Вы не достаточно хорошо знаете историю Санкт-Петербурга.
4
Вы, вероятно, только начали знакомиться с историей Санкт-
Петербурга?
```

```
Архитектор Исаакиевского собора:
3 2 1
isaak.bmp
Доменико Трезини
Огюст Монферран
Карл Росси
Александровская колонна воздвигнута в 1836 году по проекту
Огюста Монферрана как памятник, посвященный:
2 1 0
деяниям императора Александра I.
подвигу народа в Отечественной войне 1812 года.
Архитектор Зимнего дворца
3 2 1
herm.bmp
Бартоломео Растрелли
Карл Росси
Огюст Монферран
Михайловский (Инженерный) замок — жемчужина архитектуры Пе-
тербурга — построен по проекту
3 1 0
Воронихина Андрея Никифоровича
Старова Ивана Егоровича
Баженова Василия Ивановича
Остров, на котором находится Ботанический сад, основанный им-
ператором Петром I, называется:
3 3 1
bot.bmp
Заячий
Медицинский
Аптекарский
Невский проспект получил свое название
3 2 0
по имени реки, на которой стоит Санкт-Петербург.
по имени близко расположенного монастыря, Александро-Невской
лавры.
в память о знаменитом полководце - Александре Невском.
Скульптура знаменитого памятника Петру I выполнена
2 1 0
Фальконе
Клодтом
```

Файл теста может быть подготовлен в редакторе текста Блокнот или Microsoft Word. В случае использования Microsoft Word при сохранении текста следует указать, что надо сохранить только текст. Для этого в диалоговом окне **Сохранить**, в списке **Тип файла** следует выбрать **Только текст** (*.txt).

Форма приложения

Форма программы Экзаменатор приведена на рис. 10.2.

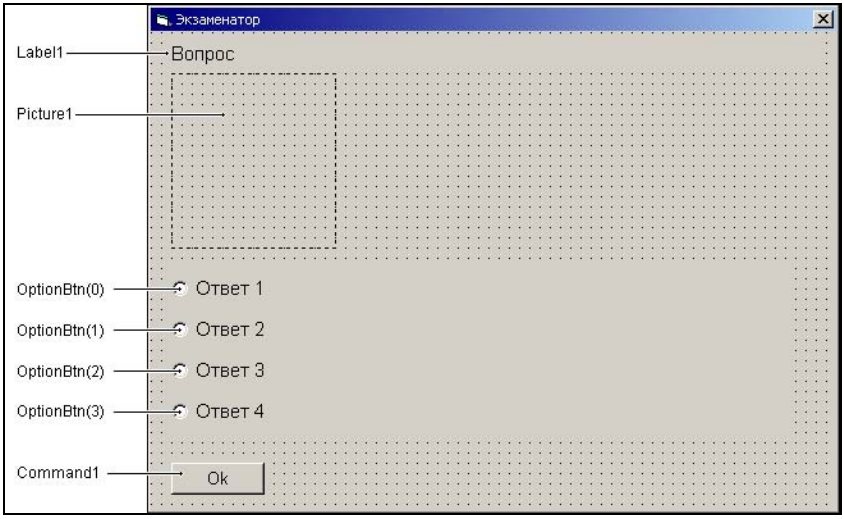


Рис. 10.2. Форма программы Экзаменатор

Поле `Label1` предназначено для вывода начальной информации, вывода вопроса и результатов тестирования. Компонент `Image1` предназначен для отображения иллюстрации, сопровождающей вопрос. Объединенные в массив `OptionButton` компоненты типа `Option` предназначены для отображения альтернативных ответов и приема ответа испытуемого.

В табл. 10.1 приведены значения свойств формы.

Таблица 10.1. *Значения свойств формы*

Свойство	Значение	Пояснение
`BorderStyle`	`1 - FixedSingle`	Тонкая граница. Изменить размер окна путем перетаскивания границы

Таблица 10.1 *(окончание)*

Свойство	Значение	Пояснение
`ScaleMode`	`1 - Pixel`	Единица измерения координат и размеров компонентов — пиксел
`StartUpPosition`	`2 - CenterScreen`	В начале работы программы окно разместить в центре экрана

Отображение иллюстраций

Для отображения иллюстраций используется компонент `Picture`, размер и положение которого проще всего задать во время разработки формы. Но в рассматриваемой программе используется другой подход: положение и размер компонента `Picture` задается (вычисляется) во время работы программы.

Очевидно, что размер области формы, которая может быть использована для вывода иллюстрации, зависит от размера поля, предназначенного для отображения вопроса и количества альтернативных ответов. Чем длиннее вопрос (больше размер поля отображения вопроса) и чем больше альтернативных ответов дано к вопросу (минимальное количество — два, максимальное — четыре), тем меньше места остается для иллюстрации. После того как очередной вопрос будет прочитан, станет известно, сколько места необходимо для отображения вопроса, сколько места займут поля отображения альтернативных ответов и, следовательно, сколько места можно выделить для отображения иллюстрации. Если размер иллюстрации превышает размер области, выделенной для ее отображения, то выполняется масштабирование.

Доступ к файлу теста

Имя файла теста можно задать в тексте программы. Очевидно, что такой подход неприемлем, если создается универсальная программа тестирования.

Другой способ задать файл теста — указать имя файла в качестве параметра команды запуска программы (в командной строке). Например, при запуске программы из операционной системы при помощи команды **Пуск ▸ Выполнить**, параметры командной строки указывают после имени выполняемого файла программы (рис. 10.3).

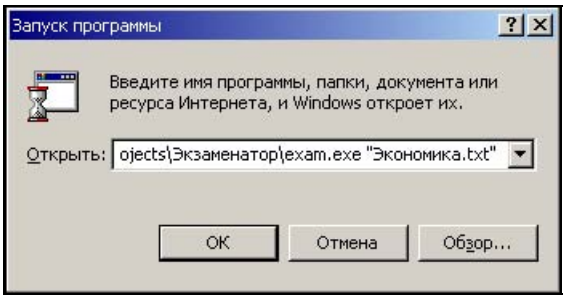


Рис. 10.3. Имя файла теста можно указать в качестве параметра команды запуска программы

Доступ к параметрам командной строки обеспечивает функция `Command`. Например, фрагмент кода, обеспечивающий прием параметра из командной строки программы Экзаменатор — выглядит так:

```
fTest = Command()
   If fTest = "" Then
      ' не задан параметр командной строки
      Label1.Caption = "В командной строке надо указать файл_
                       теста." & vbCr & "Например: exam.exe_
                       d:\exam\Экономика.txt"
      Command1.Tag = 2
      Exit Sub
   End If
```

При запуске программы Visual Basic параметры нужно ввести в поле **Command Line Arguments** вкладки **Make** окна **Projects Proper-**

ties (рис. 10.4), которое открывается в результате выбора в меню **Project** команды **Properties**.

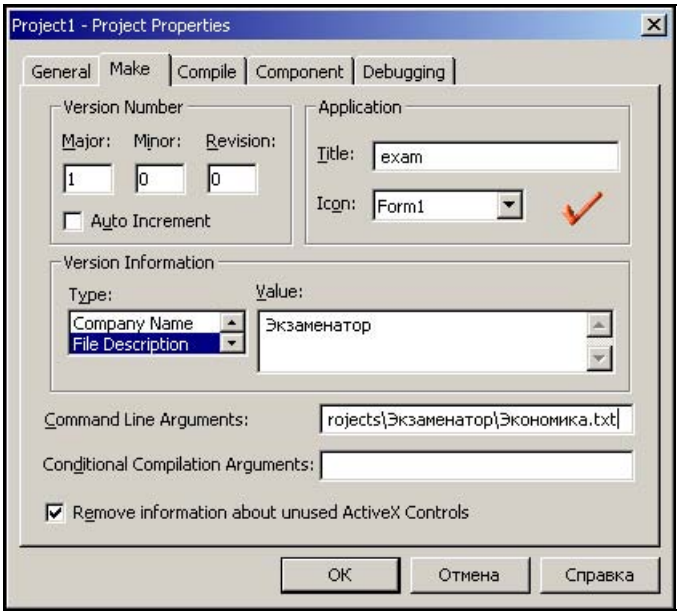


Рис. 10.4. Параметры командной строки надо ввести в поле **Command Line Arguments**

Текст программы

После того как будет создана форма программы, можно приступить к кодированию (набору теста). Текст программы Экзаменатор приведен в листинге 10.1.

Листинг 10.1. Программа Экзаменатор

```
Option Explicit


' количество уровней оценки - 4
' возможное количество вариантов ответа - 4
```

```
Dim fTest As String  ' файл теста
Dim fn As Integer    ' идентификатор файла
Dim title As String  ' название теста

Dim vopros As Integer   ' номер текущего вопроса
Dim otv As Integer      ' номер выбранного ответа
Dim right As Integer    ' номер правильного ответа
Dim nright As Integer    ' количество правильных ответов
Dim level(1 To 4) As Integer ' кол-во правильных ответов,
                         ' необходимое для достижения уровня
Dim mes(1 To 4) As String    ' уровень (оценка и сообщение)

' инициализация формы
Private Sub Form_Initialize()

   ' имя файла теста считывается из командной строки
   fTest = Command()
   If fTest = "" Then
      ' не задан параметр командной строки
      Label1.Caption = "В командной строке надо указать файл_
                       теста." & vbCr & "Например: exam.exe_
                       d:\exam\Экономика.txt"
      Command1.Tag = 2
      Exit Sub
   End If

   ' параметр командной строки указан
   On Error GoTo e1

   Call InitForm

   fn = FreeFile ' запросить у системы доступный
                 ' идентификатор файла
   Open fTest For Input As #fn' открыть файл теста для чтения
```

```
   Call info          ' вывод информации о тесте
   Call getLevel      ' чтение информации об оценках

   Form1.ScaleMode = vbPixels
   Command1.Tag = 0

   Label1.WordWrap = True
   Label1.AutoSize = True
   Exit Sub

e1:
    ' ошибка доступа к файлу теста
    Label1.WordWrap = True
    Label1.AutoSize = True
    Label1.Caption = "Ошибка доступа к файлу теста " & fTest
    Command1.Tag = 2

End Sub

' вывод информации о тесте
Sub info()
   Dim buf As String

   Line Input #fn, buf        ' чтение названия теста
   Form1.Caption = buf        ' вывод названия теста
   title = buf

   Line Input #fn, buf        ' чтение информации о тесте
   Label1.Caption = buf       ' вывод информации о тесте

End Sub

' чтение информации об уровнях оценки
Sub getLevel()
```

```
   Dim buf As String
   Dim i As Integer

   i = 1
   For i = 1 To 4
      Line Input #fn, buf ' количество баллов
      level(i) = buf
      Line Input #fn, buf ' оценка
      mes(i) = buf
   Next i

End Sub

' щелчок на кнопке OK/Дальше
Private Sub Command1_Click()
   Select Case Command1.Tag
      ' вывод первого вопроса
      Case 0:
         Command1.Enabled = False

         Call InitForm
         Call voprosToScr

         Command1.Tag = 1
         Command1.Caption = "Дальше"

      ' вывод остальных вопросов
      Case 1:
         If otv = right Then nright = nright + 1

         Command1.Enabled = False

         Call InitForm
```

```
         If Not EOF(fn) Then
            Call voprosToScr
         Else
            Close #fn
            Command1.Caption = "Ok"
            Form1.Caption = title
            Command1.Tag = 2
            Command1.Enabled = True
            Call itog   ' вывести результат
         End If

      ' завершение работы
      Case 2:
         Unload Me
   End Select
End Sub

' Очистка формы перед выводом очередного вопроса
Sub InitForm()
   Dim i As Integer

   ' "сбросить" кнопки выбора ответа
   For i = 0 To 3
        OptionBtn(i).Visible = False
        OptionBtn(i).Caption = ""
        OptionBtn(i).Value = False
   Next i

   Image1.Visible = False
End Sub

' вывод вопроса
Sub voprosToScr()
```

```
Dim n As Integer  ' кол-во вариантов ответа
Dim p As Integer  ' признак наличия иллюстрации

Dim fPicture As String     ' файл иллюстрации
Dim buf As String          ' буфер чтения

Dim i As Integer

vopros = vopros + 1
Form1.Caption = " Вопрос " + Str(vopros)

Line Input #fn, buf ' чтение вопроса
Label1.Caption = buf ' отображение вопроса

' следующая строка имеет вид: N R P
' где: N - количество альтернативных ответов;
'      R - номер правильного ответа;
'      P - признак наличия иллюстрации (1- есть, 0 - нет)

Line Input #fn, buf ' чтение информации об ответах

n = Val(Substr(buf, 1))
right = Val(Substr(buf, 2))   ' right - глобальная
                              ' переменная
p = Val(Substr(buf, 3))

If p = 1 Then
     ' прочитать имя файла иллюстрации
     Line Input #fn, fPicture
     Image1.Tag = 1
     On Error Resume Next
     Image1.Picture = LoadPicture(fPicture)
     If Err Then
```

```
            ' ошибка чтения файла иллюстрации
             ' (файл иллюстрации не найден)
            Image1.Tag = 0
        End If
    Else
        Image1.Tag = 0
    End If

    ' считывание вариантов ответа
     For i = 1 To n
        Line Input #fn, buf
        Select Case i
            Case 1: OptionBtn(0).Caption = buf
            Case 2: OptionBtn(1).Caption = buf
            Case 3: OptionBtn(2).Caption = buf
            Case 4: OptionBtn(3).Caption = buf
       End Select
    Next i

    ' здесь прочитана иллюстрация и альтернативные ответы

    ' вопрос выведен, иллюстрация и альтернативные ответы - нет
    If Image1.Tag = 1 Then    ' есть иллюстрация к вопросу
       Call showPicture
    End If

    ' вывод альтернативных ответов
    ' первый альтернативный ответ
    If OptionBtn(0).Caption <> "" Then
       If Image1.Tag = 1 Then
            OptionBtn(0).Top = Image1.Top + Image1.Height + 5
       Else
            OptionBtn(0).Top = Label1.Top + Label1.Height + 5
       End If
```

```
      OptionBtn(0).Visible = True
      OptionBtn(0).Value = False
   End If

   ' остальные альтернативные ответы
   For i = 1 To 3
    If OptionBtn(i).Caption <> "" Then
      OptionBtn(i).Top = OptionBtn(i - 1).Top + _
                         OptionBtn(i - 1).Height
      OptionBtn(i).Visible = True
      OptionBtn(i).Value = False
    End If
   Next i

   Command1.Enabled = False
End Sub

' щелчок на кнопке выбора ответа
Private Sub OptionBtn_Click(Index As Integer)
    otv = Index + 1 ' кнопки пронумерованы с нуля
    Command1.Enabled = True
End Sub

' определение достигнутого уровня
Sub itog()
   Dim i As Integer
   Dim buf As String

   buf = "Результат тестирования" + vbCr + vbCr + _
         "Всего вопросов: " + Str(vopros) + vbCr + _
         "Правильных ответов: " + Str(nright) + vbCr

   i = 0
   ' nright - кол-во правильных ответов
```

```
    ' сначала сравниваем с количеством баллов "на пять",
    ' затем - "на четыре" и т. д.
    While (nright <= level(i)) And (i < 3)
       i = i + 1
    Wend

    buf = buf + mes(i)
    Label1.Caption = buf
End Sub

' вывод иллюстрации
Sub showPicture()
    Dim w As Integer, h As Integer    ' размер области
                                      ' вывода иллюстрации

    Dim k As Single ' коэффициент масштабирования
    Dim i As Integer

    Image1.Stretch = False
    Image1.Top = Label1.Top + Label1.Height + 7

    ' определить размер области вывода иллюстрации
    w = Form1.ScaleWidth - Label1.Left * 2
    h = Command1.Top - (Label1.Top + Label1.Height) - 10 * 2

    ' размер области вывода иллюстрации зависит от количества
    ' альтернативных ответов - чем меньше
    ' вариантов ответа, тем больше область
    For i = 0 To 3
         If OptionBtn(i).Caption <> "" Then_
            h = h - OptionBtn(i).Height
    Next i

    ' если картинка меньше WxH, то она не масштабируется
```

```
    ' масштабирование по высоте
    If (Image1.Height > h) Then
       k = Image1.Width / Image1.Height
       Image1.Stretch = True
       Image1.Width = h * k
       Image1.Height = h
    End If

    ' масштабирование по ширине
    If (Image1.Width > w) Then
       Image1.Stretch = True
       Image1.Width = w
       Image1.Height = w / k
    End If

    Image1.Visible = True
End Sub

' Возвращает подстроку с указанным номером
' (строка разделена на подстроки пробелами)
Function Substr(st As String, n As Integer) As String
    Dim s As Integer ' указатель на первый символ подстроки
    Dim f As Integer ' указатель на последний символ подстроки
    Dim i As Integer

    s = 1
    For i = 1 To n - 1
        s = InStr(s, st, " ")
        s = s + 1
    Next i

    f = InStr(s + 1, st, " ")
    If f <> 0 Then
        f = f - 1
```

```
    Else
        f = Len(st)
    End If

    Substr = Mid(st, s, f - s + 1)

End Function
```

В начале работы программы процедура обработки события `Initialize` проверяет, указан ли в командной строке запуска программы параметр — имя файла теста. Если файл теста задан, то процедура открывает его, считывает из файла название теста и вводную информацию. Название теста отображается в заголовке окна, вводная информация — в поле компонента `Label`. Следует обратить внимание: предполагается, что программа запускается из того каталога, в котором находится файл теста и файлы иллюстраций. Такой подход позволяет сгруппировать все файлы одного теста в одном каталоге.

После того как прочитана общая информация о тесте, программа считывает из файла теста информацию об уровнях оценки и фиксирует ее в массивах `level` и `mes`.

После вывода информационного сообщения программа ждет, пока пользователь не нажмет кнопку **OK** (`Command1`).

Командная кнопка `Command1` используется для активизации процесса тестирования (после вывода информационного сообщения), перехода к следующему вопросу (после выбора варианта ответа) и завершения работы программы (после вывода результата тестирования или вывода сообщения об ошибке, если в командной строке запуска программы не указан файл теста).

Действие, выполняемое в результате нажатия кнопки `Command1`, зависит от значения ее свойства `Tag`. В начале работы программы значение свойства `Tag` равно нулю. Поэтому процедура обработки события `Click` выводит первый вопрос, заменяет текст на кнопке на слово **Дальше** и присваивает свойству `Tag` значение 1. В процессе тестирования значение свойства `Tag` кнопки `Command1` равно единице. Поэтому функция обработки события

`Click` сравнивает номер выбранного ответа (`otv`) с правильным (`right`), увеличивает на единицу счетчик правильных ответов (если выбран правильный ответ) и активизирует процесс чтения следующего вопроса. Если попытка чтения очередного вопроса завершилась неудачно (т. е. вопросы исчерпаны), то функция выводит результаты тестирования, заменяет текст на командной кнопке на **OK** и подготавливает операцию завершения работы программы (свойству `Tag` присваивает значение 2).

Чтение из файла очередного вопроса (вопрос, количество альтернативных ответов, номер правильного ответа, признак наличия иллюстрации, имя файла иллюстрации и альтернативные ответы) и его отображение выполняет функция `voprosToScr`. Сначала функция считывает строку из файла теста (вопрос) и выводит ее в поле компонента `Label1`. Затем считывает из файла строку вида `N R P`, где:

- `N` — количество альтернативных ответов;
- `R` — номер правильного ответа;
- `P` — признак наличия к вопросу иллюстрации (1 — иллюстрация есть, 0 — иллюстрации нет).

Номер правильного ответа фиксируется в глобальной переменной `right`. Если к вопросу есть иллюстрация, то на основании информации о количестве альтернативных ответов вычисляется размер области, которую можно использовать для отображения иллюстрации. После этого функция выводит альтернативные ответы. Положение компонента `OptionBtn(0)`, обеспечивающего вывод первого альтернативного ответа, отсчитывается от нижней границы компонента `Image1` (если к вопросу есть иллюстрация) или компонента `Label1` (если иллюстрации нет). Положение остальных компонентов `OptionBtn` отсчитывается от предыдущего компонента `OptionBtn` (`OptionBtn(1)` от `OptionBtn(0)`, `OptionBtn(2)` от `OptionBtn(1)` и т. д.).

Сразу после вывода вопроса кнопка **Дальше** (`Command1`) недоступна. Сделано это для того, чтобы блокировать возможность перехода к следующему вопросу, если не выбран ответ на вопрос текущий. Доступной кнопку **Дальше** делает процедура обработки события `Click` на одном из компонентов `OptionBtn`. Эта же процедура фиксирует в глобальной переменной `otv` номер вы-

бранного ответа (увеличенный на единицу номер компонента `OptionBtn`, в поле которого испытуемый сделал щелчок). Следует обратить внимание на то, что в рассматриваемой программе для обработки события `Click` на всех компонентах `Option` используется одна процедура (это возможно, т. к. все компоненты `Option` объединены в массив).

После того как пользователь ответит на все вопросы, процедура `itog` выводит результат тестирования. А саму процедуру `itog` вызывает процедура обработки события `Click` на кнопке `Command1` (если вопрос, на который ответил пользователь последний, т. е. значение функции `EOF` равно `True`).

Запуск программы тестирования

Пользователи Windows привыкли к тому, что в результате щелчка на имени файла автоматически запускается программа, обеспечивающая работу с этим файлом. Рассмотрим, что надо сделать, чтобы программа тестирования автоматически запускалась в результате щелчка на значке файла теста.

Когда пользователь производит щелчок на имени файла, операционная система определяет тип файла (по расширению) и запускает программу, которая *связана* (предназначена для работы) с файлами этого типа. Таким образом, чтобы обеспечить автоматический запуск программы тестирования, необходимо *связать* файл теста и программу тестирования. При этом важно, чтобы у файла теста было правильное расширение, т. к. именно по расширению операционная система определяет тип файла и, соответственно, программу, которую надо запустить. Очевидно, что расширение txt использовать нельзя. Поэтому сначала надо изменить расширение файла теста с txt, например, на enr (от examiner — экзаменатор). После этого можно связать файлы типа enr с программой Экзаменатор. Для этого надо раскрыть папку, в которой находится файл теста, щелкнуть правой кнопкой мыши на имени файла теста и из контекстного меню выбрать команду **Открыть с помощью**. На экране появится окно **Выбор программы**. В поле **Описание** этого окна надо ввести краткое описание типа файла (например, **Файл теста программы Экзаменатор**) и щелкнуть на кнопке **Другая**. Затем в появившемся окне **Открыть с помощью** надо раскрыть папку, в которой находится

программа тестирования, выбрать файл программы и щелкнуть на кнопке **Открыть**. В результате этих действий файл с расширением enr будет связан с программой тестирования (рис. 10.5). Теперь, чтобы запустить программу тестирования, достаточно сделать щелчок на имени файла теста. Естественно, у вновь создаваемых файлов теста расширение должно быть enr.

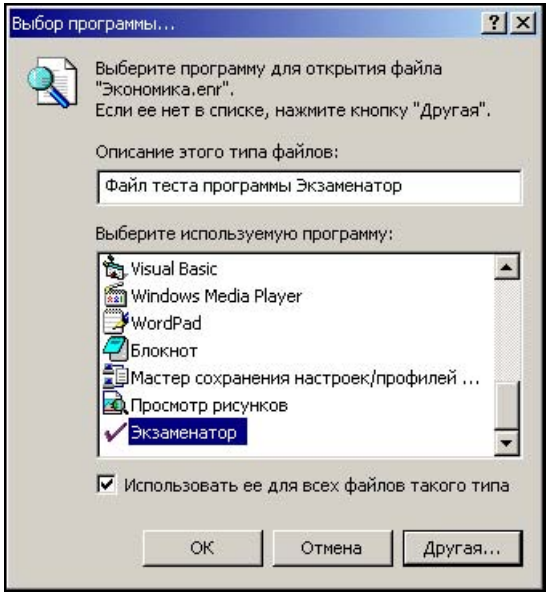


Рис. 10.5. Файлы с расширением enr связаны с программой Экзаменатор

Следует обратить внимание на то, что задачу связывания типа файла с конкретной программой можно возложить на программу установки.

Игра "Сапер"

Всем, кто работает в операционной системе Windows, хорошо знакома игра "Сапер". В этом разделе рассматривается аналогичная программа.

Пример окна программы в конце игры (после того как игрок открыл клетку, в которой находится мина) приведен на рис. 10.6.

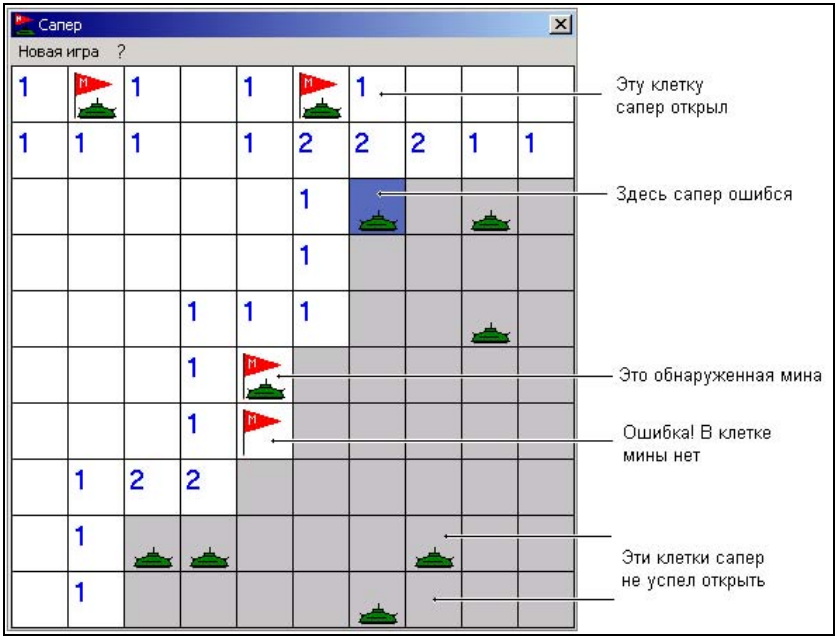


Рис. 10.6. Окно программы Сапер

Правила игры и представление данных

Игровое поле состоит из клеток, в каждой из которых может быть мина. Задача игрока — найти все мины и пометить их флажками.

Используя кнопки мыши, игрок может открыть клетку или поставить в нее флажок, указав тем самым, что в клетке находится мина. Клетка открывается щелчком левой кнопки мыши, флажок ставится щелчком правой. Если в клетке, которую открыл игрок, есть мина, то происходит взрыв (сапер ошибся, а он, как известно, ошибается только один раз) и игра заканчивается. Если в клетке мины нет, то в этой клетке появляется

число, соответствующее количеству мин, находящихся в соседних клетках. Анализируя информацию о количестве мин в клетках, соседних с уже открытыми, игрок может обнаружить и пометить флажками все мины. Ограничений на количество клеток, помеченных флажками, нет. Однако для завершения игры (выигрыша) флажки должны быть установлены только в тех клетках, в которых есть мины. Ошибочно установленный флажок можно убрать, щелкнув правой кнопкой мыши в клетке, в которой он находится.

В программе игровое поле представлено массивом $N + 2$ на $M + 2$, где $N \times M$ — размер игрового поля. Элементы массива с номерами строк от 1 до N и номерами столбцов от 1 до M соответствуют клеткам игрового поля (рис. 10.7), первые и последние столбцы и строки соответствуют границе игрового поля.

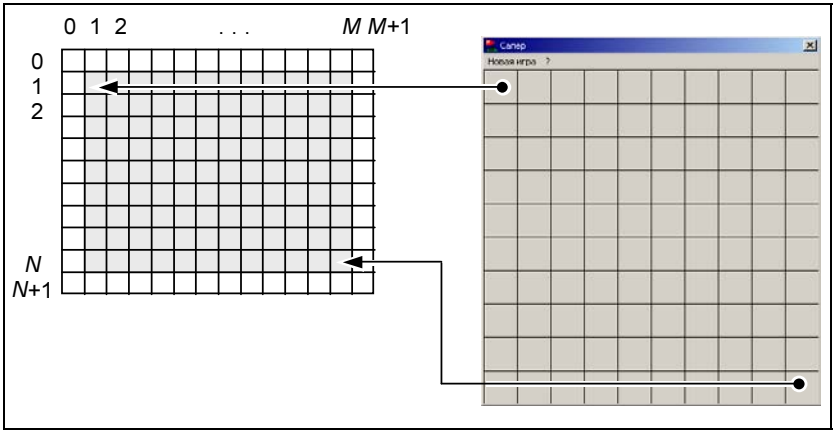


Рис. 10.7. Клетке игрового поля соответствует элемент массива

В начале игры каждый элемент массива, соответствующий клеткам игрового поля, может содержать число от 0 до 9. Ноль соответствует пустой клетке, рядом с которой нет мин. Клеткам, в которых нет мин, но рядом с которыми мины имеются, соответствуют числа от 1 до 8. Элементы массива, соответствующие клеткам, в которых находятся мины, имеют значение 9.

Элементы массива, соответствующие границе поля, имеют значение −3.

В качестве примера на рис. 10.8 изображен массив, соответствующий состоянию поля в начале игры.

–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3
–3	**9**	1	0	0	0	0	0	0	0	0	–3
–3	1	1	0	0	0	0	0	0	0	0	–3
–3	1	2	2	1	0	0	0	1	1	1	–3
–3	1	**9**	**9**	1	0	0	0	2	**9**	2	–3
–3	1	2	2	1	0	0	0	2	**9**	3	–3
–3	0	0	0	0	0	0	0	2	3	**9**	–3
–3	0	1	2	2	1	0	0	1	**9**	2	–3
–3	0	2	**9**	**9**	1	0	0	1	1	1	–3
–3	0	2	**9**	3	1	0	0	0	0	0	–3
–3	0	1	1	1	0	0	0	0	0	0	–3
–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3

Рис. 10.8. Массив в начале игры

В процессе игры состояние игрового поля меняется (игрок открывает клетки и ставит флажки) и, соответственно, меняются значения элементов массива. Если игрок поставил в клетку флажок, то значение соответствующего элемента массива увеличивается на 100. Например, если флажок поставлен правильно (т. е. в клетку, в которой есть мина), то значение соответствующего элемента массива станет 109. Если флажок поставлен ошибочно (например, в пустую клетку), то элемент массива будет содержать число 100. Если игрок просто открыл клетку, то значение элемента массива увеличивается на 200. Такой способ кодирования позволяет сохранить информацию об исходном состоянии клетки.

Форма приложения

У игры "Сапер" имеются две формы: главная (игровое поле) и **О программе**. Главная (стартовая) форма игры "Сапер" приведена на рис. 10.9.

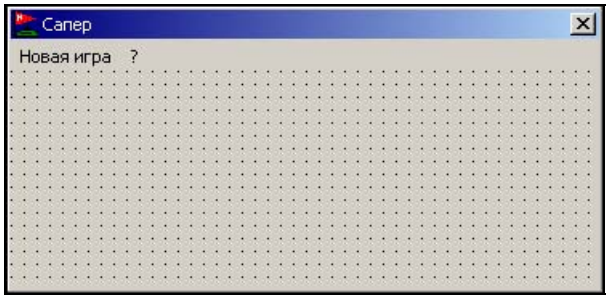


Рис. 10.9. Главная форма программы Сапер

Следует обратить внимание на то, что размер формы не соответствует размеру игрового поля. Нужный размер формы будет установлен во время работы программы. Делает это функция обработки события `Load`, которая на основе информации о размере игрового поля (количестве клеток по вертикали и горизонтали) и размере клеток устанавливает значения свойств `Width` и `Height`.

Управление работой программы осуществляется при помощи команд меню. Меню содержит команды: **Новая игра**, **Справка** **О программе**, назначение которых очевидно. Доступ к командам **Справка** и **О программе** осуществляется через раздел **?**. Меню создается при помощи утилиты Menu Editor, команда запуска которой находится в меню **Tools**. Вид окна **Menu Editor** в конце работы над меню программы Сапер приведен на рис. 10.10, характеристики меню — в табл. 10.2.

Таблица 10.2. *Меню программы Сапер*

Команда (Caption)	Идентификатор (Name)
Новая игра	`new_game`
?	`question`

Таблица 10.2 *(окончание)*

Команда (Caption)	Идентификатор (Name)
Справка	`help`
О программе	`about`

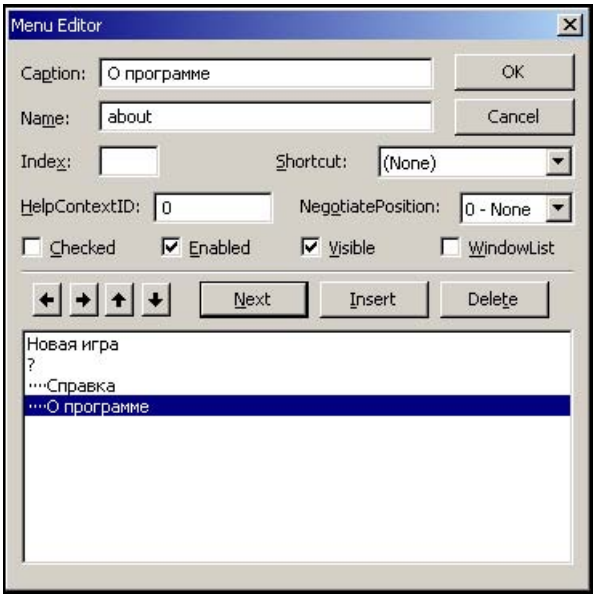


Рис. 10.10. Структура меню программы Сапер

Начало работы программы

В начале работы программы процедура обработки события `Load` (листинг 10.2) сначала загружает из файлов битовые образы (рис. 10.11), т. е. изображения флажка, мины, помеченной флажком мины, ошибочно открытой мины. Затем на основе информации о размере игрового поля (количестве клеток по горизонтали и вертикали) и размере битового образа (размер всех битовых образов одинаковый) устанавливает размер формы.

Рис. 10.11. Изображения флажка, мины, помеченной флажком мины, ошибочно открытой мины

Листинг 10.2. Процедура обработки события `Load`

```
Private Sub Form_Load()
   Dim row As Integer, col As Integer

   ' В неотображаемые элементы массива (клетки по границе
   ' игрового поля) записывается число -3. Это значение
   ' используется процедурой n_open для завершения
   ' рекурсивного процесса открытия соседних пустых клеток
   For row = 0 To MR + 1
      For col = 0 To MC + 1
         Pole(row, col) = -3
      Next col
   Next row

   ' загрузка изображений для клеток
   Set bm1 = LoadPicture("bm1.bmp") ' флажок
   Set bm2 = LoadPicture("bm2.bmp") ' мина
   Set bm3 = LoadPicture("bm3.bmp") ' мина, отмеченная флажком
   Set bm4 = LoadPicture("bm4.bmp") ' мина, на которой сапер
                                    ' подорвался

   W = ScaleX(bm1.Width, vbHimetric, vbPixels)
   H = ScaleX(bm1.Height, vbHimetric, vbPixels)

   ' установка размеров формы
   frmSaper.Width = (frmSaper.Width - frmSaper.ScaleWidth) + _
                    (MC * W) * Screen.TwipsPerPixelX
```

```
    frmSaper.Height = (frmSaper.Height - frmSaper.ScaleHeight)_
                      + (MR * H) * Screen.TwipsPerPixelY

    ' размеры объектов на форме задаются в пикселах
    frmSaper.ScaleMode = vbPixels

    Call NewGame          ' новая игра
End Sub
```

Новая игра

В начале игры нужно расставить мины и для каждой клетки поля подсчитать, сколько мин находится в соседних клетках. Процедура `NewGame` (ее текст приведен в листинге 10.3) решает эту задачу.

Листинг 10.3. Процедура NewGame

```
' процедура генерирует новое игровое поле
Sub NewGame()
    Dim row As Integer    ' координаты клетки
    Dim col As Integer

    Dim n As Integer      ' количество поставленных мин
    Dim k As Integer      ' количество мин в соседних клетках

    ' очистка игрового поля
    For row = 1 To MR
       For col = 1 To MC
          Pole(row, col) = 0
       Next col
    Next row
```

```
' расстановка мин
Randomize     ' инициализация ГСЧ
n = 0         ' количество мин

Do
   row = Int((MR * Rnd) + 1)
   col = Int((MC * Rnd) + 1)
   If (Pole(row, col) <> 9) Then
      Pole(row, col) = 9
      n = n + 1
   End If
Loop Until (n = NM)

' вычисление количества мин в соседних клетках
' для каждой клетки
For row = 1 To MR
   For col = 1 To MC
      If (Pole(row, col) <> 9) Then
         k = 0
         If Pole(row - 1, col - 1) = 9 Then k = k + 1
         If Pole(row - 1, col) = 9 Then k = k + 1
         If Pole(row - 1, col + 1) = 9 Then k = k + 1
         If Pole(row, col - 1) = 9 Then k = k + 1
         If Pole(row, col + 1) = 9 Then k = k + 1
         If Pole(row + 1, col - 1) = 9 Then k = k + 1
         If Pole(row + 1, col) = 9 Then k = k + 1
         If Pole(row + 1, col + 1) = 9 Then k = k + 1
         Pole(row, col) = k
      End If
   Next col
Next row
```

```
   status = 0        ' начало игры
   nMin = 0          ' нет обнаруженных мин
   nFlag = 0         ' нет поставленных флагов
End Sub
```

После того как процедура NewGame расставит мины, процедура ShowPole (ее текст приведен в листинге 10.4) выведет изображение игрового поля.

Листинг 10.4. Процедура ShowPole

```
' процедура выводит поле
Sub ShowPole(status As Integer)
   Dim row As Integer, col As Integer

   For row = 1 To MR
      For col = 1 To MC
         Call Kletka(row, col, status)
      Next col
   Next row
End Sub
```

Процедура ShowPole выводит изображение поля последовательно, клетка за клеткой. Вывод изображения отдельной клетки (в начале игры, во время игры и в ее конце) выполняет процедура Kletka (текст приведен в листинге 10.5).

В начале игры (значение параметра status = 0) процедура выводит изображение закрытой клетки, во время игры — количество мин в соседних клетках или флажок, а в конце — отображает состояние клетки. Информацию о фазе игры процедура Kletka получает через параметр status. Непосредственный вывод изображения клетки выполняет метод PaintPicture, которому в качестве параметра передается битовый образ, содержащий изображение, соответствующее состоянию клетки.

Листинг 10.5. Процедура `Kletka`

```
Sub Kletka(row As Integer, col As Integer, status As Integer)
   Dim X As Integer, Y As Integer    ' координаты верхнего
                                     ' левого угла клетки

   ' вычислить координаты клетки на поверхности формы
   X = (col - 1) * W
   Y = (row - 1) * H

   If status = 0 Then ' начало игры
      ' клетка закрыта
      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      Exit Sub
   End If

   ' *** неоткрытая клетка ***
   If Pole(row, col) < 100 Then

      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      ' если игра завершена (status = 2) и клетка с миной
      ' не была открыта, открываем ее
      ' (чтобы игрок увидел мину)
      If (status = 2) And (Pole(row, col) = 9) Then _
         frmSaper.PaintPicture bm2, X, Y ' мина
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
                                  ' контур клетки
      Exit Sub
   End If

   ' *** открытая клетка ***
   Line (X, Y)-Step(W, H), RGB(255, 255, 255), BF
   Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
```

```
    ' клетка открыта, в соседних клетках нет мин
    If (Pole(row, col) = 100) Then Exit Sub

   ' клетка открыта, в соседних клетках есть мины
    If (Pole(row, col) >= 101) And (Pole(row, col) <= 108)
Then
       frmSaper.CurrentX = X + 3
       frmSaper.CurrentY = Y + 3

       ' вывод количества мин в соседних клетках
       ' шрифт определяют свойства формы Font и ForeColor
       Print Str(Int(Pole(row, col) - 100))
       Exit Sub
    End If

    ' в клетку поставлен флаг
    If (Pole(row, col) >= 200) Then
         frmSaper.PaintPicture bm1, X, Y          ' флаг
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
                                        ' контур клетки
    End If

    ' на этой мине подорвались
    If (Pole(row, col) = 109) Then
         frmSaper.PaintPicture bm4, X, Y ' мина
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
                                      ' контур клетки
  End If

    ' правильно поставленный флаг
    If (Pole(row, col) = 209) And (status = 2) Then
       frmSaper.PaintPicture bm3, X, Y
```

```
                                ' мина, помеченная флагом
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
   End If
End Sub
```

Игра

Во время игры программа воспринимает нажатие кнопок мыши и в соответствии с правилами игры открывает клетки или устанавливает там флажки.

Основную работу выполняет процедура обработки события `MouseDown` (текст приведен в листинге 10.6). Процедура получает координаты точки формы, в которой игрок щелкнул кнопкой мыши, а также информацию о том, какая кнопка была нажата. Сначала процедура преобразует координаты точки формы, в которой игрок нажал кнопку мыши, в координаты клетки игрового поля. Затем делает необходимые изменения в массиве `Pole`, и если нажата правая кнопка, то вызывает функцию `Kletka`, которая отображает содержимое клетки. Если левая кнопка мыши нажата в той клетке, где мины нет, то вызывается процедура `n_open`, которая открывает клетку (отображает ее содержимое). Если левая кнопка мыши нажата в той клетке, где есть мина, то фиксируется факт окончания игры и процедура `ShowPole` показывает все мины, в т. ч. и те, которые игрок не успел найти.

Листинг 10.6. Обработка события `MouseDown` на поверхности игрового поля

```
' нажатие кнопки мыши на игровом поле
Private Sub Form_MouseDown(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   Dim row As Integer, col As Integer

   If status = 2 Then Exit Sub       ' игра завершена
```

```
If status = 0 Then status = 1     ' первый щелчок

' преобразование координат мыши в индексы клетки поля
row = Int(Y / H) + 1
col = Int(X / W) + 1

' нажатие левой кнопки мыши
If Button = vbLeftButton Then
   If Pole(row, col) = 9 Then
      ' открыта клетка, в которой есть мина
      Pole(row, col) = Pole(row, col) + 100
      status = 2              ' игра закончена
      Call ShowPole(status)   ' вывод поля
   Else
      ' открытие клетки
      If Pole(row, col) < 9 Then Call n_open(row, col)
   End If
End If

' нажатие правой кнопки мыши
If Button = vbRightButton Then
   ' в клетке стоит флаг, пользователь хочет убрать его
   If Pole(row, col) >= 200 Then
      nFlag = nFlag - 1
      ' уберем флаг из клетки
      Pole(row, col) = Pole(row, col) - 200
      ' закрытие клетки
      Call Kletka(row, col, status)

   ' в клетке нет флага, пользователь хочет его поставить
   Else
      ' если клетка открыта, то флаг нельзя поставить,
      ' если клетка закрыта, то можно
      If Pole(row, col) >= 100 Then Exit Sub
```

```
            nFlag = nFlag + 1
            Pole(row, col) = Pole(row, col) + 200  ' установка
                                                   ' флага
            Call Kletka(row, col, status)          ' вывод флага

            If Pole(row, col) = 209 Then
               nMin = nMin + 1

               ' если все мины помечены флажками, игра закончена
               If (nMin = NM) And (nFlag = NM) Then
                  status = 2                ' игра закончена
                  Call ShowPole(status)     ' вывод поля
               End If
            End If

         End If
      End If

End Sub
```

Справочная информация

В результате выбора в меню **?** команды **Справка** или нажатия клавиши <F1> должна появиться справочная информация — правила игры (рис. 10.12).

Чтобы во время работы программы пользователь, нажав клавишу <F1>, мог получить справочную информацию, надо ввести имя файла справочной информации в поле **Help File Name** окна **Project Properties**, которое становится доступным в результате выбора в меню **Project** команды **Project Properties**.

Для того чтобы справочная информация появилась на экране в результате выбора в меню **?** команды **Справка**, надо создать функцию обработки события `Click` для соответствующей коман-

ды меню. Чтобы это сделать, надо в окне формы раскрыть меню и сделать щелчок мышью в строке команды.

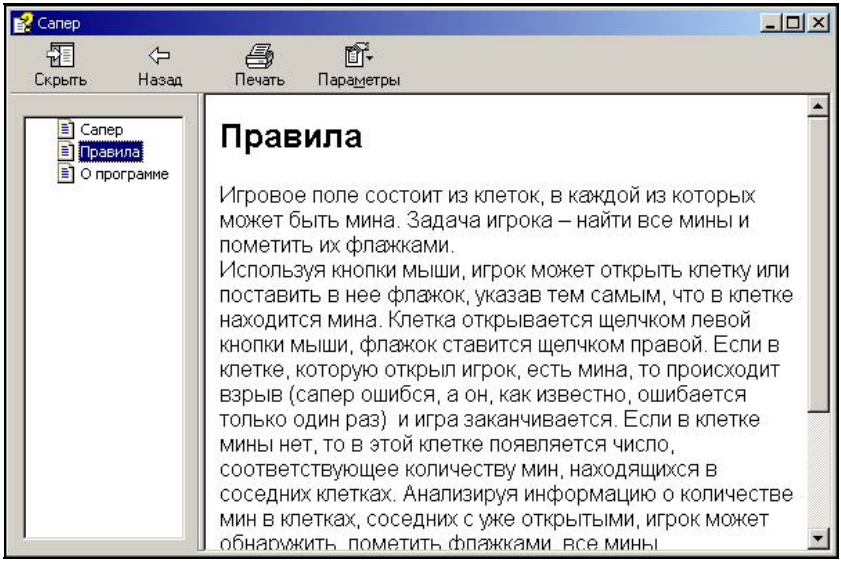


Рис. 10.12. Окно справочной системы программы Сапер

В качестве примера приведена функция обработки события `Click` для команды **Справка**.

```
' выбор пункта меню "Справка"
Private Sub help_Click()
    Shell "hh saper.chm", vbNormalFocus
End Sub
```

Информация о программе

При выборе из меню **?** команды **О программе** на экране должно появиться одноименное окно (рис. 10.13).

Чтобы программа во время своей работы могла вывести на экран окно, отличное от главного (стартового), нужно добавить в про-

ект форму. Делается это выбором из меню **Project** команды **Add Form**. В результате выполнения команды в проект добавляется новая форма и соответствующий ей модуль.

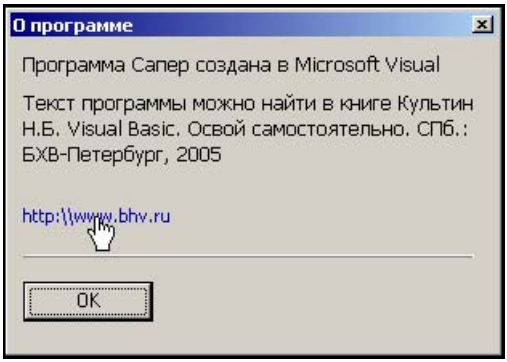


Рис. 10.13. Окно, в котором, выбрав ссылку, можно активизировать браузер и перейти на нужную страницу

Вид формы **О программе** (`frmAbout`) после добавления необходимых компонентов приведен на рис. 10.14, значения ее свойств — в табл. 10.3.

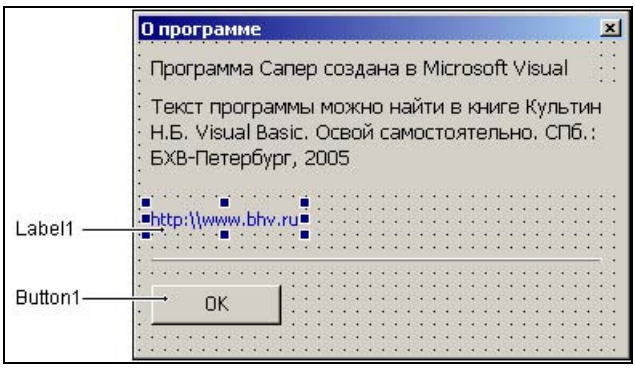


Рис. 10.14. Форма **О программе**

Таблица 10.3. *Значения свойств формы* ***О программе***

Свойство	Значение
Name	frmAbout
BorderStyle	4 — Fixed ToolWindow
StartUpPosition	1 — CenterOwner

Вывод окна **О программе** выполняет функция обработки события Click, которое происходит в результате выбора из меню **?** команды **О программе** (листинг 10.7). Непосредственно вывод окна выполняет метод ShowM с параметром vbModal который выводит окно как *модальный* диалог. Модальный диалог перехватывает все события, адресованные другим окнам приложения (в т. ч. и главному). Таким образом, пока модальный диалог находится на экране, продолжить работу с приложением, которое вывело этот модальный диалог, нельзя.

Листинг 10.7. Вывод окна О программе

```
' выбор команды "О программе" в меню "?"
Private Sub about_Click()
    frmAbout.Show vbModal, Me
End Sub
```

Следует обратить внимание на то, что окно **О программе** появляется в центре главного окна программы, потому что свойству StartUpPosition присвоено значение CenterOwner (что означает: отображать в центре того окна, которое активизировало процесс отображения). В данном случае процесс отображения активизировало главное окно программы.

На поверхности формы **О программе** есть ссылка на сайт издательства "БХВ-Петербург". Предполагается, что в результате щелчка на ссылке в окне браузера будет открыта указанная страница. Чтобы это произошло, надо создать функцию обработки события Click для компонента Label1. Кроме того, чтобы указатель принимал форму руки (при позиционировании

указателя мыши на ссылке), в проект надо добавить соответствующий *курсор* и установить значения свойств `MousePointer` и `MouseIcon`.

Курсор — это битовый образ размера 32×32. Можно использовать один из готовых курсоров (курсоры находятся в файлах с расширением cur) или при помощи утилиты Image Editor создать свой собственный (рис. 10.15). Процесс создания курсора практически ничем не отличается от процесса создания значка приложения (был подробно описан в *главе 2*), за исключением того, что в окне **Resource Type**, которое появляется в результате выбора в меню **File** команды **New**, надо выбрать **Cursor**, а не **Icon**.

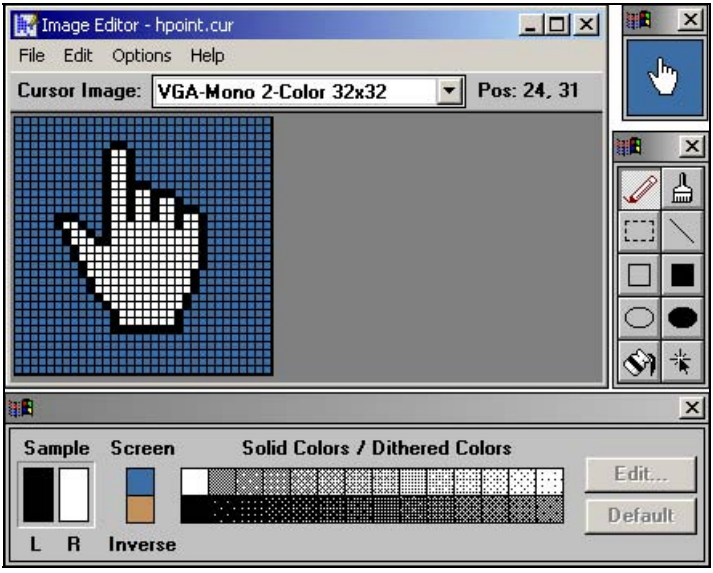


Рис. 10.15. Курсор можно создать при помощи утилиты Image Editor

Значения свойств компонента `Label1` приведены в табл. 10.4, текст процедуры обработки события `Click` в поле компонента приведен в листинге 10.8.

Таблица 10.4. *Значения свойств компонента* `Label1`

Свойство	Значение
`Caption`	`http:\\www.bhv.ru`
`MousePointer`	99 — `Custom`
`MouseIcon`	`hpoint.cur`

Листинг 10.8. Процедура обработки события `Click` в поле компонента `Label1` (щелчок в поле URL)

```
Private Declare Function ShellExecute Lib "shell32.dll"_
   Alias "ShellExecuteA" (ByVal hwnd As Long,_
   ByVal lpOperation As String, ByVal lpFile As String,_
   ByVal lpParameters As String, ByVal lpDirectory As String,_
   ByVal nShowCmd As Long) As Long


Private Sub Label1_Click()
    Dim r As Long
    r = ShellExecute(Me.hwnd, "Open", "http:\\www.bhv.ru",_
        "", "", 0)
End Sub
```

Для запуска браузера использована API-функция `ShellExecute`, которая сообщает операционной системе, что необходимо открыть указанный файл. Так как в данном случае документ имеет тип URL, то операционная система запускает программу, обеспечивающую работу с документами указанного типа, т. е. браузер.

Окно **О программе** закрывается в результате щелчка на кнопке **OK**. Текст процедуры обработки этого события приведен в листинге 10.9.

Листинг 10.9. Щелчок на кнопке OK

```
Private Sub Command1_Click()
    Hide ' закрыть (скрыть) окно "О программе"
End Sub
```

Текст программы

Полный текст программы Сапер приведен в двух листингах: в листинге 10.10 приведен модуль главной формы (frmSaper), в листинге 10.11 — формы **О программе** (frmAbout).

Листинг 10.10. Модуль главной формы

```
Const MR = 10    ' количество клеток по вертикали
Const MC = 10    ' количество клеток по горизонтали
Const NM = 10    ' количество мин

' минное поле
Dim Pole(0 To MR + 1, 0 To MC + 1) As Integer
' значение элемента массива:
' 0..8 - количество мин в соседних клетках,
' 9 - в клетке мина,
' 100..109 - клетка открыта,
' 200..209 - в клетку поставлен флаг

Dim nMin As Integer      ' количество найденных мин
Dim nFlag As Integer     ' количество поставленных флагов
Dim status As Integer    ' статус игры: 0 - начало игры,
                         ' 1- идет игра, 2 - результат игры

' изображения клеток
Dim bm1 As StdPicture ' флажок
Dim bm2 As StdPicture ' мина
```

```
Dim bm3 As StdPicture ' мина, отмеченная флажком
Dim bm4 As StdPicture ' мина, на которой сапер подорвался

Dim W, H As Integer ' размер клетки

' начало работы программы
Private Sub Form_Load()
   Dim row As Integer, col As Integer

   ' В неотображаемые элементы массива (клетки по границе
   ' игрового поля) записывается число -3. Это значение
   ' используется процедурой n_open для завершения
   ' рекурсивного процесса открытия соседних пустых клеток
   For row = 0 To MR + 1
      For col = 0 To MC + 1
         Pole(row, col) = -3
      Next col
   Next row

   ' загрузка изображений для клеток
   Set bm1 = LoadPicture("bm1.bmp") ' флажок
   Set bm2 = LoadPicture("bm2.bmp") ' мина
   Set bm3 = LoadPicture("bm3.bmp") ' мина, отмеченная флажком
   Set bm4 = LoadPicture("bm4.bmp") ' мина, на которой сапер
                                    ' подорвался

   W = ScaleX(bm1.Width, vbHimetric, vbPixels)
   H = ScaleX(bm1.Height, vbHimetric, vbPixels)

   ' установка размеров формы
   frmSaper.Width = (frmSaper.Width - frmSaper.ScaleWidth)_
                    + (MC * W) * Screen.TwipsPerPixelX
```

```
    frmSaper.Height = (frmSaper.Height - frmSaper.ScaleHeight)_
                    + (MR * H) * Screen.TwipsPerPixelY

    ' размеры объектов на форме задаются в пикселах
    frmSaper.ScaleMode = vbPixels
    Call NewGame          ' новая игра
End Sub

' процедура генерирует новое игровое поле
Sub NewGame()
    Dim row As Integer    ' координаты клетки
    Dim col As Integer

    Dim n As Integer      ' количество поставленных мин
    Dim k As Integer      ' количество мин в соседних клетках

    ' очистка игрового поля
    For row = 1 To MR
       For col = 1 To MC
          Pole(row, col) = 0
       Next col
    Next row

    ' расстановка мин
    Randomize     ' инициализация ГСЧ
    n = 0         ' количество мин
    Do
       row = Int((MR * Rnd) + 1)
       col = Int((MC * Rnd) + 1)
       If (Pole(row, col) <> 9) Then
          Pole(row, col) = 9
          n = n + 1
       End If
    Loop Until (n = NM)
```

```
    ' вычисление количества мин в соседних клетках
    ' для каждой клетки
    For row = 1 To MR
       For col = 1 To MC
          If (Pole(row, col) <> 9) Then
             k = 0
             If Pole(row - 1, col - 1) = 9 Then k = k + 1
             If Pole(row - 1, col) = 9 Then k = k + 1
             If Pole(row - 1, col + 1) = 9 Then k = k + 1
             If Pole(row, col - 1) = 9 Then k = k + 1
             If Pole(row, col + 1) = 9 Then k = k + 1
             If Pole(row + 1, col - 1) = 9 Then k = k + 1
             If Pole(row + 1, col) = 9 Then k = k + 1
             If Pole(row + 1, col + 1) = 9 Then k = k + 1
             Pole(row, col) = k
          End If
       Next col
    Next row

    status = 0        ' начало игры
    nMin = 0          ' нет обнаруженных мин
    nFlag = 0         ' нет поставленных флагов
End Sub

' процедура выводит поле
Sub ShowPole(status As Integer)
    Dim row As Integer, col As Integer

    For row = 1 To MR
       For col = 1 To MC
          Call Kletka(row, col, status)
       Next col
    Next row
End Sub
```

```
' выводит изображение клетки
Sub Kletka(row As Integer, col As Integer, status As Integer)
   Dim X As Integer, Y As Integer    ' координаты верхнего
                                     ' левого угла клетки
   ' вычислить координаты клетки на поверхности формы
   X = (col - 1) * W
   Y = (row - 1) * H
   If status = 0 Then ' начало игры
      ' клетка закрыта
      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      Exit Sub
   End If

   ' *** неоткрытая клетка ***
   If Pole(row, col) < 100 Then
      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      ' если игра завершена (status = 2) и клетка с миной
      ' не была открыта, открываем ее
      ' (чтобы игрок увидел мину)
      If (status = 2) And (Pole(row, col) = 9) Then _
         frmSaper.PaintPicture bm2, X, Y ' мина
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
                                     ' контур клетки
      Exit Sub
   End If

   ' *** открытая клетка ***
   Line (X, Y)-Step(W, H), RGB(255, 255, 255), BF
   Line (X, Y)-Step(W, H), RGB(0, 0, 0), B

   ' клетка открыта, в соседних клетках нет мин
   If (Pole(row, col) = 100) Then Exit Sub
```

```
    ' клетка открыта, в соседних клетках есть мины
    If (Pole(row, col) >= 101) And (Pole(row, col) <= 108) Then
       frmSaper.CurrentX = X + 3
       frmSaper.CurrentY = Y + 3

       ' вывод количества мин в соседних клетках
       ' шрифт определяют свойства формы Font и ForeColor
       Print Str(Int(Pole(row, col) - 100))
       Exit Sub
    End If

    ' правильно поставленный флаг
    If (Pole(row, col) = 209) And (status = 2) Then
       frmSaper.PaintPicture bm3, X, Y
                          ' мина, помеченная флагом
       Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
    End If

  ' неправильно поставленный флаг
    If (Pole(row, col) >= 200) Then
         frmSaper.PaintPicture bm1, X, Y          ' флаг
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
                                      ' контур клетки
    End If

    ' на этой мине подорвались
    If (Pole(row, col) = 109) Then
         frmSaper.PaintPicture bm4, X, Y ' мина
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
                                    ' контур клетки
   End If
End Sub
```

```
' нажатие кнопки мыши на игровом поле
Private Sub Form_MouseDown(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   Dim row As Integer, col As Integer

   If status = 2 Then Exit Sub       ' игра завершена

   If status = 0 Then status = 1     ' первый щелчок

   ' преобразование координат мыши в индексы клетки поля
   row = Int(Y / H) + 1
   col = Int(X / W) + 1

   ' нажатие левой кнопки мыши
   If Button = vbLeftButton Then
      If Pole(row, col) = 9 Then
         ' открыта клетка, в которой есть мина
         Pole(row, col) = Pole(row, col) + 100
         status = 2              ' игра закончена
         Call ShowPole(status)   ' вывод поля
      Else
         ' открытие клетки
         If Pole(row, col) < 9 Then Call n_open(row, col)
      End If
   End If

   ' нажатие правой кнопки мыши
   If Button = vbRightButton Then
      ' в клетке стоит флаг, пользователь хочет убрать его
      If Pole(row, col) >= 200 Then
         nFlag = nFlag - 1
         ' уберем флаг из клетки
```

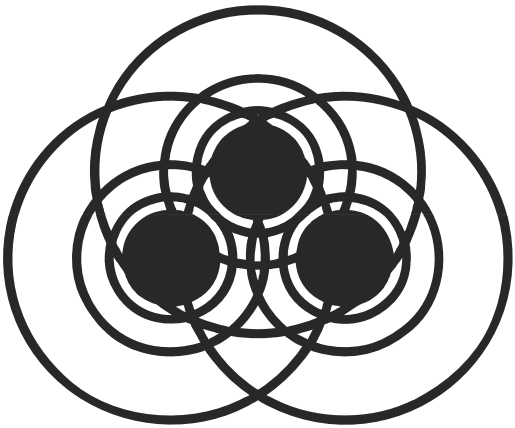
```
            Pole(row, col) = Pole(row, col) - 200
            Call Kletka(row, col, status)

         Else ' в клетке нет флага, пользователь
              ' хочет его поставить
              ' если клетка открыта, то флаг нельзя поставить,
              ' если клетка закрыта, то можно
            If Pole(row, col) >= 100 Then Exit Sub

            nFlag = nFlag + 1
            Pole(row, col) = Pole(row, col) + 200  ' установка
                                                   ' флага
            Call Kletka(row, col, status)          ' вывод флага

            If Pole(row, col) = 209 Then
               nMin = nMin + 1
               ' если все флаги расставлены на правильных местах
               If (nMin = NM) And (nFlag = NM) Then
                  status = 2                ' игра закончена
                  Call ShowPole(status)     ' вывод поля
               End If
            End If

         End If
      End If

End Sub

' рекурсивная процедура открывает текущую и все соседние
' клетки, в которых нет мин
Sub n_open(row As Integer, col As Integer)
   If Pole(row, col) = 0 Then
      Pole(row, col) = 100
      Call Kletka(row, col, 1)
```

```
      ' примыкающие клетки по вертикали и горизонтали
      Call n_open(row, col - 1)
      Call n_open(row - 1, col)
      Call n_open(row, col + 1)
      Call n_open(row + 1, col)
      ' примыкающие диагонально
      Call n_open(row - 1, col - 1)
      Call n_open(row - 1, col + 1)
      Call n_open(row + 1, col - 1)
      Call n_open(row + 1, col + 1)
   Else
      If (Pole(row, col) < 100) And (Pole(row, col) <> -3) Then
         Pole(row, col) = Pole(row, col) + 100
         Call Kletka(row, col, 1)
      End If
   End If
End Sub

' обработка события Paint
Private Sub Form_Paint()
   ' вывод игрового поля
   Call ShowPole(status)
End Sub

' выбор команды "О программе" в меню "?"
Private Sub about_Click()
    frmAbout.Show vbModal, Me
End Sub

' выбор пункта меню "Справка"
Private Sub help_Click()
    Shell "hh saper.chm", vbNormalFocus
End Sub
```

```
' выбор пункта меню "Новая игра"
Private Sub new_game_Click()
   Call NewGame              ' новая игра
   Call ShowPole(status)     ' вывод игрового поля
End Sub
```

Листинг 10.11. Модуль формы О программе

```
Private Declare Function ShellExecute Lib "shell32.dll"_
   Alias "ShellExecuteA" (ByVal hwnd As Long,_
   ByVal lpOperation As String, ByVal lpFile As String,_
   ByVal lpParameters As String, ByVal lpDirectory As String,_
   ByVal nShowCmd As Long) As Long

' щелчок на Web-ссылке (в поле компонента Label1)
Private Sub Label1_Click()
    Dim r As Long
    r = ShellExecute(Me.hwnd, "Open", "http:\\www.bhv.ru",_
                     "", "", 0)
End Sub

' щелчок на кнопке ОК
Private Sub Command1_Click()
    Hide ' закрыть (скрыть) окно "О программе"
End Sub
```

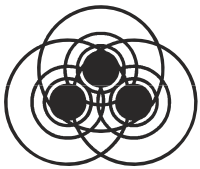


ЧАСТЬ III

Справочник

Третья часть книги представляет собой справочник по компонентам, методам и функциям Visual Basic.

Глава 11



Краткий справочник

Компоненты

Форма

Форма (объект `Form`) является основой программы. Свойства формы (табл. 11.1) определяют вид окна программы.

Таблица 11.1. *Свойства объекта* `Form` *(формы)*

Свойство	Описание
`Name`	Имя формы. Используется для управления формой и доступа к ее компонентам или свойствам
`Caption`	Текст заголовка
`Top`	Расстояние от верхней границы формы до верхней границы экрана
`Left`	Расстояние от левой границы формы до левой границы экрана
`Width`	Ширина формы. Задается в твипах
`Height`	Высота формы. Задается в твипах
`ScaleWidth`	Ширина рабочей области формы (без учета ширины левой и правой границ). Может задаваться как в твипах, так и в других единицах

Таблица 11.1 *(продолжение)*

Свойство	Описание
`ScaleHeight`	Высота рабочей (клиентской) области формы (без учета высоты заголовка и ширины нижней и верхней границ формы). Может задаваться как в твипах, так и в других единицах
`ScaleMode`	Определяет единицы измерения размеров формы и объектов на ней. Значение этого свойства не влияет на единицы измерения свойств `Width` и `Height` (независимо от него их значения измеряются в твипах)
`BorderStyle`	Стиль (вид) границы формы (окна программы). Граница может быть: • обычной (`Sizable`); • тонкой (`FixedSingle`); • вообще отсутствовать (`None`). Если значение свойства равно `FixedSingle`, то изменить размер окна программы путем перемещения границы нельзя, но окно можно развернуть на весь экран (кнопка **Maximize**) или свернуть (кнопка **Minimaze**). Если значение свойства равно `None`, то граница окна отсутствует. Изменить размер и положение такого окна нельзя. Если значение свойства равно `FixedDialog`, то окно называется "модальный диалог" (нельзя изменить размер окна, нельзя свернуть окно, доступ к другим окнам программы блокируется)
`Icon`	Значок (иконка) в заголовке окна
`BackColor`	Цвет фона формы. Цвет фона можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой и выбранным компонентом привязки, и тогда он меняется при изменении цветовой схемы операционной системы

Таблица 11.1 *(продолжение)*

Свойство	Описание
ForeColor	Цвет, используемый при выводе текста и для контуров графических объектов
FillColor	Цвет заполнения внутренней части графических объектов
FillStyle	Стиль (способ) заливки графических объектов: • Transparent(1) — прозрачный цвет (заливки нет); • Solid(0) — сплошная заливка; • HorizontalLine(2) — горизонтальная штриховка; • VerticalLine(3) — вертикальная штриховка. Цвет линий штриховки определяет значение свойства FillColor
DrawMode	Способ вывода графических объектов. Например, если значение свойства равно Blackness (1), то цвет всех контуров объектов и цвет заливки будет черным (значение этого свойства не влияет на цвет текста, выводимого при помощи метода Print)
DrawWidth	Толщина линии для графических объектов
DrawStyle	Стиль контура графических объектов, тип линии: • Solid(0) — сплошная линия; • Dash(1) — пунктирная линия; • Dot(2) — линия из точек; • Dash-Dot(3) — линия "точка-тире"; • Dash-Dot-Dot(4) — линия "тире-точка-точка"; • Transparent(5) — "прозрачная" линия
Font	Шрифт. Шрифт, заданный в этом свойстве, используется при выводе текста непосредственно на поверхность формы (например, при помощи команды Print)

Таблица 11.1 *(окончание)*

Свойство	Описание
`MaxButton`	Признак наличия (значение — `True`) или отсутствия (значение — `False`) в заголовке формы кнопки **Развернуть окно на весь экран**
`MinButton`	Признак наличия (значение — `True`) или отсутствия (значение — `False`) в заголовке формы кнопки **Свернуть окно**

CheckBox

Компонент `CheckBox` (рис. 11.1) представляет собой независимый переключатель. Свойства компонента приведены в табл. 11.2.

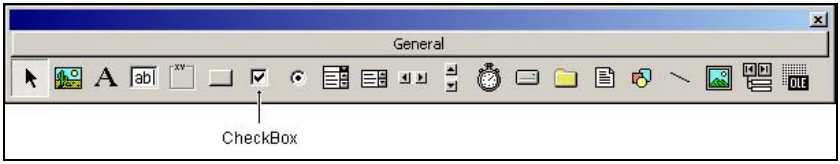


Рис. 11.1. Компонент `CheckBox`

Таблица 11.2. Свойства компонента `CheckBox`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Текст, который находится справа от флажка
`Value`	Состояние переключателя: • `Checked` — флажок установлен; • `Unchecked` — флажок сброшен
`Left`	Расстояние от левой границы флажка до левой границы формы

Таблица 11.2 *(окончание)*

Свойство	Описание
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента (флажок и область для пояснительного текста)
`Font`	Шрифт, используемый для отображения пояснительного текста
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

ComboBox

Компонент `ComboBox` (рис. 11.2) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или путем выбора из списка. Свойства компонента приведены в табл. 11.3.

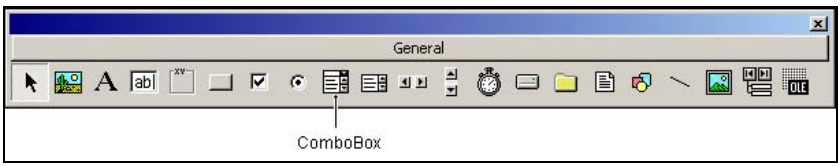


Рис. 11.2. Компонент `ComboBox`

Таблица 11.3. Свойства компонента `ComboBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения элементов списка

Таблица 11.3 *(продолжение)*

Свойство	Описание
`BackColor`	Цвет фона области вывода списка. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет, используемый для отображения элементов списка
`List`	Элементы списка — массив строк
`ListCount`	Количество элементов списка
`ListIndex`	Номер элемента, выбранного в списке. Если ни один из элементов списка не был выбран, то значение свойства равно `-1`. Нумерация элементов начинается с нуля
`Style`	Стиль (вид) списка. Если значение свойства равно `DropdownCombo(0)`, то данные в поле редактирования можно ввести с клавиатуры или выбрать из списка (чтобы получить доступ к списку, его надо раскрыть). Если значение свойства равно `SimpleCombo(1)`, то данные можно ввести в поле редактирования с клавиатуры или выбрать из списка, причем список доступен всегда Если значение свойства равно `DropdownList(2)`, то данные в поле редактирования можно ввести только путем выбора из списка
`Text`	Содержимое поля редактирования (данные, введенные пользователем с клавиатуры или выбранные из списка)
`Sorted`	Признак необходимости автоматической сортировки списка после добавления очередного элемента (значение свойства — `True`)
`Locked`	Блокировка списка. Определяет запрет выбора элемента из списка (строка ввода также блокируется)

Таблица 11.3 *(окончание)*

Свойство	Описание
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

CommandButton

Компонент `CommandButton` (рис. 11.3) представляет собой командную кнопку. Свойства компонента приведены в табл. 11.4.

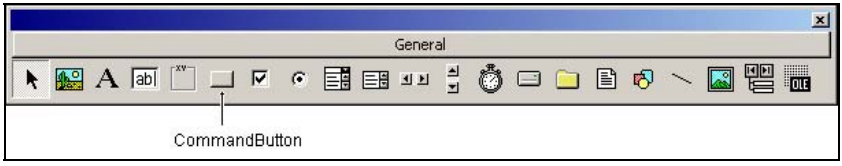


Рис. 11.3. Компонент `CommandButton`

Таблица 11.4. Свойства компонента `CommandButton`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы

Таблица 11.4 *(окончание)*

Свойство	Описание
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Height`	Высота кнопки
`Width`	Ширина кнопки
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна. Если значение свойства равно `False`, то кнопка не доступна, т. е. при щелчке на кнопке никаких событий не возникает
`Visible`	Позволяет скрыть кнопку (значение — `False`) или сделать ее видимой (значение — `True`)
`Style`	Вид кнопки: • обычная (`Standard`); • графическая (`Graphical`) (кнопка, на поверхности которой есть картинка)
`Picture`	Свойство задает картинку для "графической" кнопки. Картинка отображается на поверхности формы, если значение свойства `Style` равно `Graphical`
`DisabledPicture`	Задает картинку для недоступной графической кнопки. Картинка отображается, если значение свойства `Style` равно `Graphical`, а свойства `Enabled` — `False`
`DownPicture`	Задает картинку для нажатой графической кнопки. Картинка отображается в момент нажатия кнопки, если значение свойства `Style` равно `Graphical`
`ToolTipText`	Задает текст подсказки, которая появляется при позиционировании указателя мыши на кнопке

CommonDialog

Компонент ComonDialog (рис. 11.4) представляет собой одно из стандартных диалоговых окон: **Открыть**, **Сохранить**, **Цвет**, **Шрифт**, **Печать** или **Справка**. Свойства компонента приведены в табл. 11.5. Тип окна определяет метод, обеспечивающий отображение диалога (табл. 11.6).

Чтобы компонент был доступен, надо подключить библиотеку Microsoft Common Dialog Control 6.0 (COMDLG32.OCX).

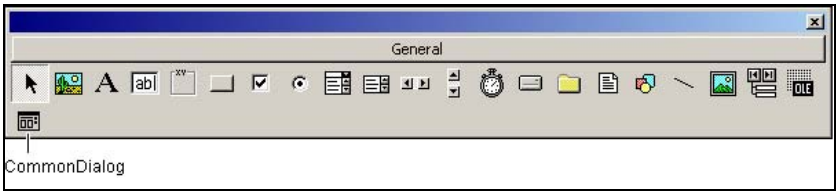


Рис. 11.4. Компонент CommonDialog

Таблица 11.5. *Свойства компонента* CommonDialog

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
DialogTitle	Заголовок окна
FileName	Полное имя файла, выбранного в диалоге **Открыть**. Если файл не был выбран (т. е. диалог завершен нажатием кнопки **Cancel**), то значением свойства является пустая строка
Filter	Фильтр. Задает файлы, отображаемые в окнах (диалогах) **Открыть** и **Сохранить**. Если фильтр не задан, то отображаются все файлы
FilterIndex	Номер выбранного фильтра. Фильтры номеруются с 1

***Таблица 11.5** (окончание)*

Свойство	Описание
`Flags`	Флаги спецификации для диалоговых окон
`HelpFile`	Путь к HELP-файлу, который нужно отобразить

***Таблица. 11.6.** Методы отображения объекта `CommoDialog`*

Метод	Диалог
`ShowOpen`	**Открыть**
`ShowSave`	**Сохранить**
`ShowColor`	**Цвет**
`ShowFont`	**Выбор шрифта**
`ShowPrinter`	**Печать**
`ShowHelp`	**Справка**

DirListBox

Компонент `DirListBox` (рис. 11.5) отображает структуру указанного каталога. Свойства компонента приведены в табл. 11.7.

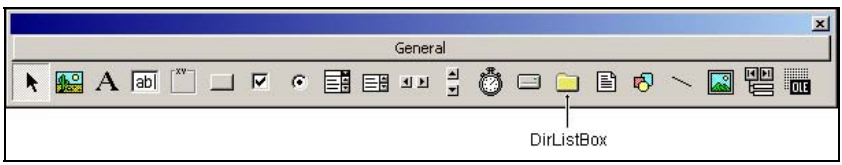


Рис. 11.5. Компонент `DirListBox`

***Таблица 11.7.** Свойства компонента `DirListBox`*

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам

Таблица 11.7 *(продолжение)*

Свойство	Описание
`Font`	Шрифт, используемый для отображения названий каталогов и подкаталогов
`BackColor`	Цвет фона области вывода списка каталогов. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет шрифта, используемого для отображения названий каталогов и подкаталогов
`List`	Элементы указанного каталога (подкаталога) — массив строк. В элемент массива записывается полный путь доступа к подкаталогу
`ListCount`	Количество подкаталогов указанного каталога
`ListIndex`	Номер элемента, выбранного в списке каталогов и подкаталогов. Нумерация подкаталогов начинается с нуля. Если в списке выбран каталог, структура которого отображается в компоненте `Dir1`, то значение свойства равно –1. Если в структуре выбрать каталог на один уровень выше (при этом список должен отображаться), то значение будет равно –2. Значение свойства продолжает уменьшаться на 1 при каждом переходе вверх по дереву каталогов
`Path`	Путь к каталогу
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента

Таблица 11.7 *(окончание)*

Свойство	Описание
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

DriveListBox

Компонент `DriveListBox` (рис. 11.6) является раскрывающимся списком, отображающим диски компьютера. Свойства компонента приведены в табл. 11.8.

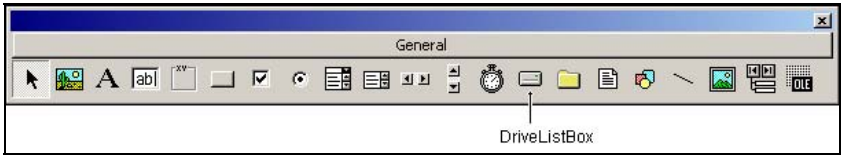


Рис. 11.6. Компонент `DriveListBox`

Таблица 11.8. *Свойства компонента* `DriveListBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения названий дисков
`BackColor`	Цвет фона области вывода списка дисков. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы

Таблица 11.8 *(окончание)*

Свойство	Описание
`ForeColor`	Цвет шрифта, используемого для отображения имен дисков
`List`	Элементы списка — массив строк, в котором находятся названия дисков (например, C:)
`ListCount`	Количество дисков компьютера
`ListIndex`	Номер элемента, выбранного в списке дисков. Если ни один из элементов списка не был выбран, то значение свойства равно –1. Нумерация дисков начинается с нуля
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

FileListBox

Компонент `FileListBox` (рис. 11.7) отображает список файлов указанного каталога. Свойства компонента приведены в табл. 11.9.

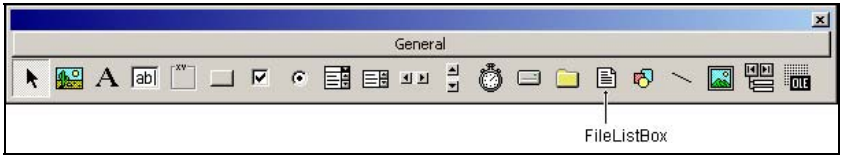


Рис. 11.7. Компонент `FileListBox`

Таблица 11.9. *Свойства компонента* `FileListBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения названий каталогов и подкаталогов
`BackColor`	Цвет фона области вывода списка каталогов. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет шрифта, используемого для отображения списка файлов указанного каталога
`List`	Элементы указанного каталога (список файлов) — массив строк. В элемент массива записывается название файла (имя.расширение)
`ListCount`	Количество файлов в указанном каталоге
`ListIndex`	Номер элемента, выбранного в списке файлов. Нумерация файлов начинается с нуля
`Path`	Путь к каталогу
`Archive`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "архивный"
`Hidden`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "скрытый"
`ReadOnly`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "только чтение"
`System`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "системный"
`Normal`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом: "архивный", "только чтение", без атрибутов или со всевозможными комбинациями этих атрибутов

Таблица 11.9 *(окончание)*

Свойство	Описание
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Image

Компонент `Image` (рис. 11.8) обеспечивает отображение иллюстраций. Отличается от компонента `PictureBox` тем, что на его поверхности рисовать нельзя. Свойства компонента приведены в табл. 11.10.

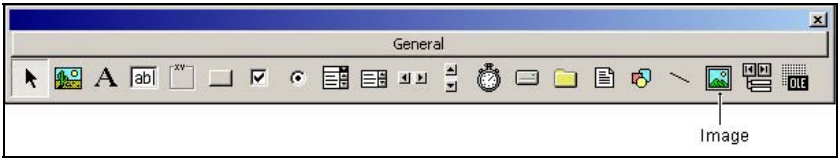


Рис. 11.8. Компонент `Image`

Таблица 11.10. *Свойства компонента* `Image`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам

Таблица 11.10 *(окончание)*

Свойство	Описание
`Picture`	Картинка (объект `BitMap`), отображаемая в поле компонента. Задать картинку можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`BorderStyle`	Стиль границы компонента: • `FixedSingle(1)` — стандартная граница (тонкая линия); • `None(0)` — границы нет
`Stretch`	Признак масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. При значении `True` иллюстрация масштабируется в соответствии с размером компонента (если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена). При значении `False` масштабирование не выполняется
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Label

Компонент `Label` (рис. 11.9) предназначен для вывода текста на поверхность формы. Свойства компонента (табл. 11.11) определяют вид и расположение текста на поверхности формы.

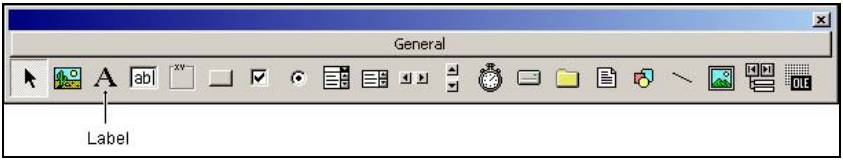


Рис. 11.9. Компонент `Label` — поле вывода текста

Таблица 11.11. *Свойства компонента* `Label`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Отображаемый текст
`Left`	Расстояние от левой границы поля вывода до левой границы формы
`Top`	Расстояние от верхней границы поля вывода до верхней границы формы
`Height`	Высота поля вывода
`Width`	Ширина поля вывода
`AutoSize`	Признак того, что размер поля определяется его содержимым
`WordWrap`	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку
`Alignment`	Задает способ выравнивания текста внутри поля. Текст может быть выровнен: • по левому краю — `LeftJustify(0)`; • по центру — `Center(2)`; • по правому краю — `RightJustify(1)`
`Font`	Шрифт, используемый для отображения текста. Уточняющие свойства `Name`, `Size` и `FColor` задают способ начертания, размер и цвет символов

Таблица 11.11 *(окончание)*

Свойство	Описание
`BackColor`	Цвет фона области вывода текста. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`BackStyle`	Управляет отображением фона области вывода текста. Область вывода текста может быть закрашена (`Opaque`) или быть "прозрачной" (`Transparent`). Если значение свойства равно `Opaque`, то цвет закраски области определяет значение свойства `BackColor`
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

Line

Компонент `Line` (рис. 11.10) — это графический объект "линия". Компонент может использоваться только в качестве декоративного элемента, т. к. он не может воспринимать события. Свойства компонента приведены в табл. 11.12.

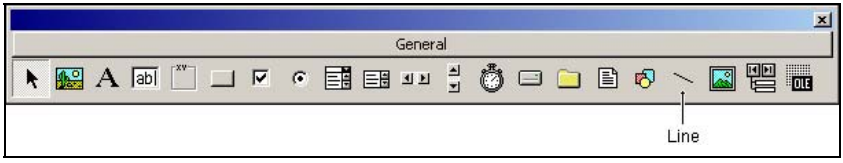


Рис. 11.10. Компонент `Line`

Таблица 11.12. *Свойства компонента* `Line`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам

Таблица 11.12 *(окончание)*

Свойство	Описание
`BorderColor`	Цвет линии
`BorderStyle`	Вид линии: • `Solid(0)` — сплошная; • `Dash(1)` — пунктирная; • `Dot(2)` — линия из точек; • `Dash-Dot(3)` — линия "точка-тире"; • `Dash-Dot-Dot(4)` — линия "тире-точка-точка"; • `Transparent(5)` — прозрачная линия
`DrawWidth`	Толщина линии
`X1`	Горизонтальная координата точки начала линии
`Y1`	Вертикальная координата точки начала линии
`X2`	Горизонтальная координата точки конца линии
`Y2`	Вертикальная координата точки конца линии
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

ListBox

Компонент `ListBox` (рис. 11.11) представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 11.13.

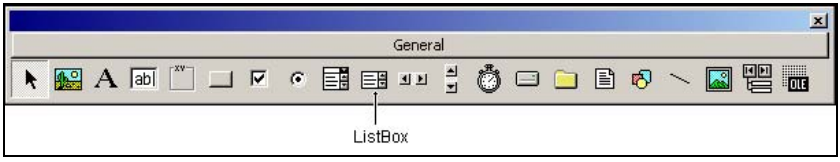


Рис. 11.11. Компонент `ListBox`

Таблица 11.13. *Свойства компонента* `ListBox`

Свойство	Описание
`Name`	Имя компонента. В программе используется для доступа к компоненту и его свойствам
`BackColor`	Цвет фона области вывода списка. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет, используемый для отображения элементов списка
`List`	Элементы списка — массив строк
`ListCount`	Количество элементов списка
`Sorted`	Признак необходимости автоматической сортировки (значение свойства — `True`) списка после добавления очередного элемента
`ListIndex`	Номер выбранного элемента списка (нумерация элементов списка начинается с нуля)
`MultiSelect`	Позволяет сделать возможным множественный выбор из списка. Если значение этого свойства — `None(0)`, то выбрать несколько элементов из списка нельзя. При значении, равном `Simple(1)`, каждый элемент, на котором сделан щелчок, становится выбранным. Отмена выбора производится при помощи повторного щелчка мыши или при помощи клавиши <пробел>. Если значение свойства равно `Extended(2)`, то множественный выбор осуществляется при помощи мыши и клавиши <Shift> или <Ctrl> (щелчок кнопкой мыши на элементе списка при нажатой клавише <Shift> помечает элемент как выбранный, повторный щелчок отменяет выбор)

Таблица 11.13 *(окончание)*

Свойство	Описание
`Style`	Стиль (вид) списка. Если значение свойства равно `CheckBox(1)`, то перед каждым элементом списка отображается компонент `CheckBox`, при этом для выбора элемента из списка нужно установить соответствующий флажок. При значении свойства, равном `Standard(0)`, список имеет стандартный вид
`Left`	Расстояние от левой границы списка до левой границы формы
`Top`	Расстояние от верхней границы списка до верхней границы формы
`Height`	Высота поля списка
`Width`	Ширина поля списка
`Font`	Шрифт, используемый для отображения элементов списка
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

MMControl

Компонент `MMControl` (рис. 11.12, 11.13) обеспечивает воспроизведение звуковых и видеофайлов. Свойства компонента приведены в табл. 11.14.

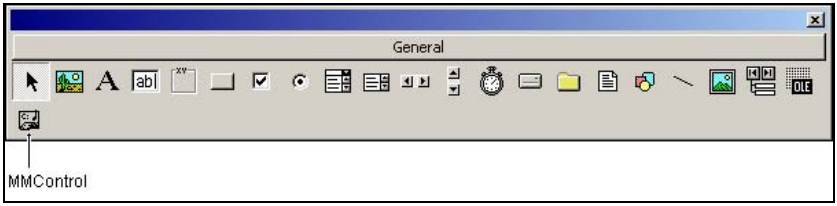


Рис. 11.12. Значок компонента `MMControl`

Рис. 11.13. Кнопки компонента `MMControl`

Чтобы компонент был доступен, надо подключить библиотеку Microsoft Multimedia Control 6.0 (MCI32.OCX).

Таблица 11.14. *Свойства компонента* `MMControl`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`DeviceType`	Тип устройства: • `CDAudio` — проигрыватель звуковых CD; • `WaveAudio` — проигрыватель WAV-файлов; • `AVIVideo` — проигрыватель AVI-файлов и др.
`Command`	Команда
`TimeFormat`	Формат измерения времени
`FileName`	Имя файла, который нужно воспроизвести
`PlayEnabled`	Делает кнопку **Play** данного компонента доступной (значение — `True`) или недоступной (значение — `False`). Аналогичное свойство есть для всех остальных кнопок компонента
`PlayVisible`	Позволяет скрыть кнопку **Play** (значение — `False`) компонента или сделать ее видимой (значение — `True`). Аналогичное свойство имеется для всех остальных кнопок компонента
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента

Таблица 11.14 *(окончание)*

Свойство	Описание
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

OptionButton

Компонент `OptionButton` (рис. 11.14) представляет зависимую кнопку, состояние которой определяется состоянием других кнопок группы. Свойства компонента приведены в табл. 11.15.

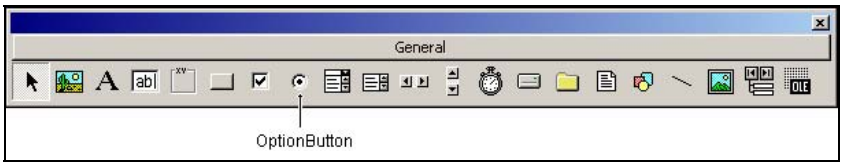


Рис. 11.14. Компонент `OptionButton`

Таблица 11.15. *Свойства компонента* `OptionButton`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст, который находится справа от кнопки
`Value`	Состояние кнопки: • если кнопка выбрана, то значение свойства — `True`; • если кнопка не выбрана, значение свойства — `False`
`Tag`	Пользователь может использовать это свойство по своему усмотрению
`Left`	Расстояние от левой границы флажка до левой границы формы

Таблица 11.15 *(окончание)*

Свойство	Описание
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента (флажок и область для пояснительного текста)
`Font`	Шрифт, используемый для отображения поясняющего текста
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

PictureBox

Компонент `PictureBox` (рис. 11.15) обеспечивает отображение графики. Изображение можно загрузить из файла или сформировать из графических примитивов (нарисовать) во время работы программы. Свойства компонента приведены в табл. 11.16.

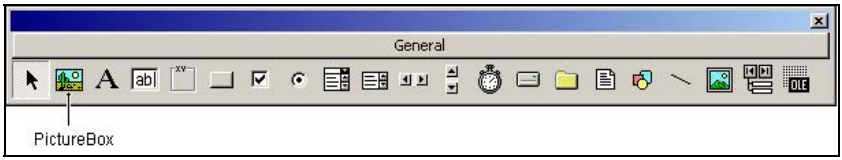


Рис. 11.15. Компонент `PictureBox`

Таблица 11.16. *Свойства компонента* `PictureBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам

Таблица 11.16 *(продолжение)*

Свойство	Описание
`Picture`	Картинка (объект `BitMap`), отображаемая в поле компонента. Задать картинку можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`AutoSize`	Свойство, разрешающее (значение — `True`) или запрещающее (значение — `False`) автоматическое изменение размера компонента (области вывода иллюстрации) в соответствии с размером картинки, загруженной в этот компонент
`BorderStyle`	Стиль границы компонента: • если значение свойства — `FixedSingle(1)`, то граница стандартная (тонкая линия); • если значение свойства — `None(0)`, то граница не отображается
`BackColor`	Цвет фона компонента. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`Font`	Шрифт, которым метод `Print` выводит текст
`ForeColor`	Свойство, задающее цвет: • для метода `Print` задает цвет символов; • для методов вычерчивания графических примитивов (объектов) задает цвет линий
`FillColor`	Свойство, задающее цвет закраски внутренних областей графических примитивов (объектов), вычерчиваемых в поле (на поверхности) компонента
`FillStyle`	Стиль закраски графических объектов, вычерчиваемых в поле компонентов соответствующими методами: • `Solid(0)` — сплошная заливка; • `HorizontalLine(2)` — горизонтальная штриховка;

Таблица 11.16 *(продолжение)*

Свойство	Описание
	• `VerticalLine(3)` — вертикальная штриховка; • `Transparent(1)` — закраска "прозрачным" цветом. Цвет линий штриховки определяет свойство `FillColor`
`DrawStyle`	Вид контура графических объектов, вычерчиваемых в поле компонента соответствующими методами: • `Solid(0)` — сплошная линия; • `Dash(1)` — пунктирная линия; • `Dot(2)` — линия, состоящая из точек; • `Dash-Dot(3)` — линия вида "точка-тире"; • `Dash-Dot-Dot(4)` — линия вида "тире-точка-точка"; • `Transparent(5)` — "прозрачная" линия
`DrawWidth`	Толщина линий для графических объектов
`ScaleWidth`	Ширина рабочей области компонента (без учета ширины левой и правой границ). Единицу измерения задает свойство `ScaleMode`
`ScaleHeight`	Высота рабочей области компонента (без учета ширины нижней и верхней границ компонента). Единицу измерения задает свойство `ScaleMode`
`ScaleMode`	Свойство, задающее единицу измерения размеров компонента и объектов на его поверхности. Значение этого свойства не влияет на единицы измерения свойств `Width` и `Height` (их значения измеряются в твипах)
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента

Таблица 11.16 *(окончание)*

Свойство	Описание
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

ProgressBar

Компонент `ProgressBar` (рис. 11.16) представляет собой индикатор, который обычно используется для наглядного представления протекания процесса, например, обработки (копирования) файлов, загрузки информации из сети и т. п. Свойства этого компонента приведены в табл. 11.17.

Чтобы компонент был доступен, надо подключить библиотеку Microsoft Windows Common Controls 6.0 SP4 (MSCOMCTL.OCX).

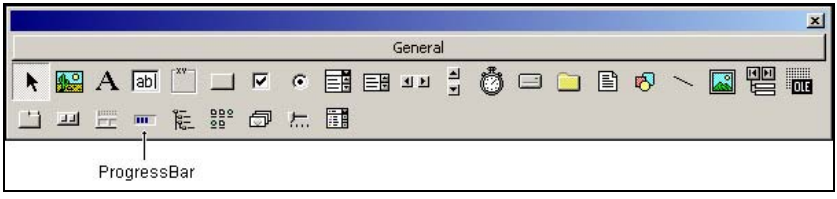


Рис. 11.16. Компонент `ProgressBar`

Таблица 11.17. Свойства компонента `ProgressBar`

Свойство	Описание
`Value`	Значение, которое отображается в поле компонента (на индикаторе) в виде полосы или ряда прямоугольников. Длина полосы (количество прямоугольников) пропорционально значению свойства `Value`. Свойство доступно только во время работы программы

Таблица 11.17 *(окончание)*

Свойство	Описание
`Min`	Минимально допустимое значение свойства `Value`
`Max`	Максимально допустимое значение свойства `Value`
`Scrolling`	Вид индикатора. Индикатор может представлять собой последовательность прямоугольников (0 – `ccScrollingStandard`) или полосу (1 — `ccScrollingSmoth`)
`Appearance`	Вид компонента: • вровень с поверхностью формы — (0 — `ccFlat`); • в стиле 3D — (1 – `cc3D`)
`Border`	Граница компонента: • есть — (1 — `ccFixedSingle`); • нет — (0 — `ccNone`)

Shape

Компонент `Shape` (рис. 11.17) — это графический объект (прямоугольник, овал (круг), прямоугольник со скругленными углами), который можно поместить на поверхность формы. Компонент используется только в качестве декоративного элемента, т. к. он не может воспринимать события. Свойства компонента приведены в табл. 11.18.

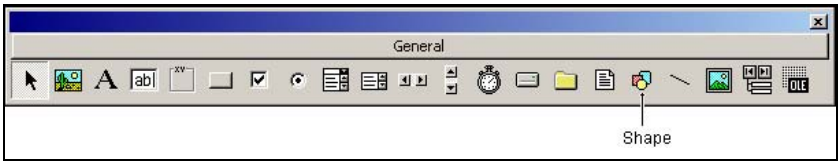


Рис. 11.17. Компонент `Shape`

Таблица 11.18. *Свойства компонента* `Shape`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Shape`	Вид геометрической фигуры: • `Rectangle(0)` — прямоугольник; • `Square(1)` — квадрат; • `Oval(2)` — овал; • `Circle(3)` — круг; • `RoundedRectangle(4)` — прямоугольник со скругленными углами; • `RoundedSquare(5)` — квадрат со скругленными углами
`BackColor`	Цвет фона компонента. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`BackStyle`	Стиль фона компонента: • `Transparent(0)` — прозрачный; • `Opaque(1)` — непрозрачный
`BorderColor`	Цвет границы объекта (контура геометрической фигуры)
`BorderStyle`	Вид контура объекта (геометрической фигуры): • `Solid(0)` — сплошная линия; • `Dash(1)` — пунктирная линия; • `Dot(2)` — линия из точек; • `Dash-Dot(3)` — линия "точка-тире"; • `Dash-Dot-Dot(4)` — линия "тире-точка-точка"; • `Transparent(5)` — "прозрачная" линия

Таблица 11.18 *(окончание)*

Свойство	Описание
`DrawWidth`	Толщина линии контура объекта (геометрической фигуры)
`FillColor`	Цвет закраски внутренней области объекта (геометрической фигуры)
`FillStyle`	Стиль закраски внутренней области объекта (геометрической фигуры): • `Transparent(1)` — "прозрачный" цвет (нет закраски); • `Solid(0)` — сплошная заливка; • `HorizontalLine(2)` — горизонтальная штриховка; • `VerticalLine(3)` — вертикальная штриховка. Цвет линий штриховки определяется значением свойства `FillColor`
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

StatusBar

Компонент `StatusBar` (рис. 11.18) представляет собой полосу вывода служебной информации.

Чтобы компонент был доступен, надо подключить библиотеку Microsoft Windows Common Controls 6.0 SP4 (MSCOMCTL.OCX).

Чтобы полоса состояния была разделена на области, в нее надо добавить соответствующее количество *панелей*. Для этого надо

раскрыть окно свойств компонента StatusBar (из контекстного меню компонента выбрать команду **Properties**), открыть вкладку **Panels** и щелкнуть на кнопке **InsertPanel** столько раз, на сколько областей должна быть разделена строка состояния. Свойства компонента StatusBar приведены в табл. 11.19, панели компонента StatusBar — в табл. 11.20.

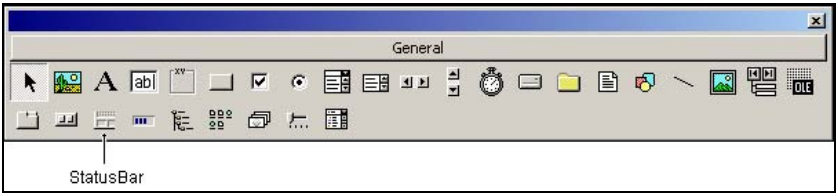


Рис. 11.18. Компонент StatusBar

Таблица 11.19. *Свойства компонента* StatusBar

Свойство	Описание
Style	Стиль (вид) области состояния. Панель может быть: • обычной (0 – sbrNormal), она разделена на области; • простой (1 — sbrSimple), представляющей собой одну-единственную область
SimpleText	Текст, который отображается на панели состояния, если панель простая (значение свойства Style равно sbrSimple)

Таблица 11.20. *Свойства панели (объекта* StatusBarPanel*)*

Свойство	Описание
Style	Тип информации, отображаемый на панели: • текст — (0 — sbrText); • время — (5 — sbrTime); • дата — (6 — sbrDate).

Таблица 11.20 *(окончание)*

Свойство	Описание
	Состояние клавиатуры: • 1 — `sbrCaps`; • 2 — `sbrNum`; • 3 — `sbrIns`
`Text`	Текст, отображаемый на панели (если значение свойства `Style` равно `Style`)
`Alignment`	Размещение текста на панели: • прижат к левой границе — (0 — `sbrLeft`); • прижат к правой границе — (2 — `sbrRight`); • расположен по центру — (1 — `sbrCenter`)
`Picture`	Картинка, отображаемая на панели
`AutoSize`	Признак автоматического изменения размера панели. Если значение свойства равно 2 (`sbrContents`), то ширина панели зависит от ее содержания (длины текста). Если значение свойства равно 1 (`sbrSpring`), то ширина панели устанавливается такой, чтобы находящаяся справа другая панель была прижата к правой границе окна. Если справа панели нет, то ширина устанавливается такой, чтобы правая граница панели была прижата к правой границе формы

TextBox

Компонент `TextBox` (рис. 11.19) представляет собой поле ввода-редактирования текста. Свойства компонента приведены в табл. 11.21.

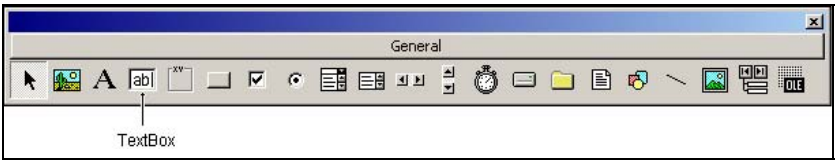


Рис. 11.19. Компонент `TextBox`

Таблица 11.21. *Свойства компонента* `TextBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам (в частности — для доступа к тексту, введенному в поле редактирования)
`Text`	Текст, находящийся в поле редактирования
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота поля
`Width`	Ширина поля
`Font`	Шрифт, используемый для отображения содержимого поля
`Locked`	Свойство, используемое для ограничения возможности изменения текста в поле редактирования. Если значение свойства равно `False`, то текст в поле редактирования изменить нельзя
`MultiLine`	Свойство, делающее возможным многострочное отображение текста
`ScrollBars`	Свойство, управляющее отображением полос прокрутки: • `Vertical` — только полоса вертикальной прокрутки;

Таблица 11.21 *(окончание)*

Свойство	Описание
	• `Horizontal` — только полоса горизонтальной прокрутки; • `Both` — вертикальная и горизонтальная полосы прокрутки; • `None` — без полос прокрутки
`Visible`	Позволяет скрыть компонент (значение — False) или сделать его видимым (значение — `True`)

Timer

Компонент `Timer` (рис. 11.20) обеспечивает генерацию последовательности событий. Свойства компонента приведены в табл. 11.22.

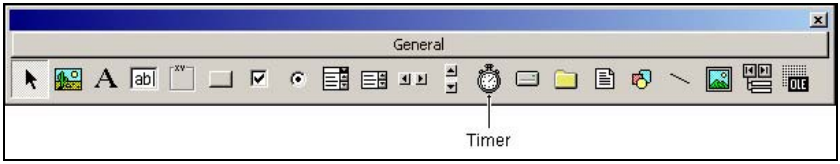


Рис. 11.20. Компонент `Timer`

Таблица 11.22. *Свойства компонента* `Timer`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Interval`	Период генерации события `Timer`. Задается в миллисекундах
`Enabled`	Разрешение работы (запуск/остановка). Разрешает (значение свойства — `True`) или запрещает (значение свойства — `False`) генерацию события `Timer`

UpDown

Компонент UpDown (рис. 11.21) представляет собой две кнопки, используя которые можно изменить (увеличить или уменьшить) значение внутренней переменной-счетчика. Компонент UpDown можно связать с другим компонентом (например, с Label) и использовать его в качестве индикатора значения переменной-счетчика. Свойства компонента UpDown приведены в табл. 11.23.

Чтобы компонент был доступен, надо подключить библиотеку Microsoft Windows Common Controls-2 6.0 (MSCOMCT2.OCX).

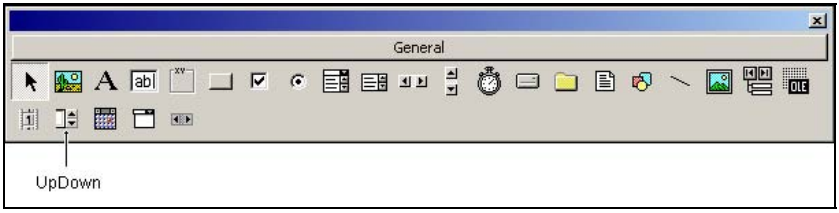


Рис. 11.21. Компонент UpDown

Таблица 11.23. *Свойства компонента* UpDown

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Value	Счетчик. Значение счетчика изменяется в результате щелчка на кнопке **Up** (увеличение) или **Down** (уменьшение)
Increment	Величина изменения значения переменной-счетчика
Max	Верхняя граница изменения значений переменной-счетчика Value
Min	Нижняя граница изменения значений переменной-счетчика Value

Таблица 11.23 *(продолжение)*

Свойство	Описание
`BuddyControl`	Компонент, который используется в качестве индикатора значения переменной-счетчика (например, `Label` или `TextBox`)
`BuddyProperty`	Свойство компонента, указанного в `BuddyControl`, обеспечивающее индикацию значения переменной-счетчика (`Caption`, если индикатором является компонент `Label`)
`AutoBuddy`	Автоматическое определение свойства компонента-индикатора, предназначенного для индикации состояния переменной-счетчика. Если в качестве индикатора используется компонент `Label`, то автоматически в качестве значения свойства `BuddyProperty` устанавливается `Caption`
`SyncBuddy`	Свойство, синхронизирующее (при значении `True`) изменение значения `Value` и значения свойства компонента-индикатора
`Orientation`	Ориентация кнопок (стрелок на кнопках) компонента: • `OrientationVertical(0)` — по вертикали (вверх, вниз); • `OrientationHorizontal(1)` — по горизонтали (влево, вправо)
`Wrap`	Если значение свойства равно `False`, то значение переменной `Value`, достигнув максимума, не изменяется при последующих нажатиях кнопки **Up**. Аналогично для кнопки **Down**. Если значение свойства равно `True`, то при аналогичных действиях максимальное значение переменной `Value` изменяется на минимальное и наоборот
`Enabled`	Доступность компонента (значение свойства — `True`) или его недоступность (значение свойства — `False`)

Таблица 11.23 *(окончание)*

Свойство	Описание
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Графика

В этом разделе приведено описание методов, обеспечивающих вывод графики на поверхность формы или компонента `PictureBox`.

Инструкции вызова метода или обращения к свойству в общем виде выглядят так:

```
Объект.Метод
Объект.Свойство
```

Следует обратить внимание на то, что объект в инструкции вызова метода или обращения к свойству объекта можно не указывать. Если объект не указан, то используется объект "по умолчанию", т. е. форма.

В описании параметры методов выделены курсивом. Необязательные параметры заключены в квадратные скобки.

Circle

```
[объект.]Circle [Step] (x,y), Radius, [Color,Start,End,Aspect]
```

Метод `Circle` позволяет нарисовать окружность, эллипс (если задан параметр *`Aspect`*) или дугу. Цвет контура определяет свойство `ForeColor` графической поверхности, на которой метод рисует (если не задан параметр *`Color`*).

Вид контура (стиль линии) определяют соответственно свойства `DrawWidth` и `DrawStyle` графической поверхности, на которой метод рисует.

Параметр *`Step`* показывает, что реальные координаты отсчитываются от указателя текущей точки.

Параметр *`Radius`* задает радиус окружности или, если задан параметр *`Aspect`*, больший радиус эллипса.

Параметр *`Color`* задает цвет контура (по умолчанию цвет контура определяет свойство `ForeColor` графической поверхности).

При вычерчивании дуги параметр *`Start`* задает угловую координату точки начала дуги, а параметр *`End`* — угловую координату точки конца дуги. Дуга вычерчивается от точки *`Start`* против часовой стрелки. Угловые координаты измеряются в радианах. Для пересчета величины угла из градусов в радианы можно воспользоваться формулой:

$$r = 2\ p\ /\ (a\ /\ 360),$$

где:

- r — величина угла в радианах;
- a — величина угла в градусах;
- p — число "Пи" (3,1415926).

Параметр *`Aspect`* задает вид эллипса.

Если значение параметра *`Aspect`* меньше 1, то эллипс получается путем сжатия окружности по вертикали (если значение параметра 0, то эллипс вырождается в вертикальную линию).

Если значение параметра *`Aspect`* больше 1, то эллипс получается путем сжатия окружности по горизонтали.

Пример:

```
pi = 3.1415926 ' число "Пи"
ScaleMode = 3 ' единица измерения координат - пикселы

Circle (100, 50), 30 ' окружность
Circle (50, 50), 30, RGB(255, 0, 0)
Circle (150, 50), 30, , 0, pi ' дуга
```

```
Circle (150, 50), 30, , 0, pi / 2 ' дуга
Circle (150, 50), 30, RGB(255, 0, 0), pi / 2, 0 ' дуга

Circle (100, 120), 30, , , , 0.5 ' эллипс
Circle (100, 120), 30, , , , 2   ' эллипс
```

Line

[*Объект*.]Line [*Step*] (*x1*, *y1*) [*Step*]-(*x2*, *y2*), [*Color*], [*B*][*F*]

Метод Line рисует линию или прямоугольник (если указан параметр *B*).

Параметры *x1*, *y1* и *x2*, *y2* задают координаты точки начала и конца линии или находящихся на одной диагонали углов прямоугольника (если указан параметр *B*).

Цвет линии (контура прямоугольника) определяет свойство ForeColor графической поверхности (если не указан параметр *Color*).

Толщину и вид определяют соответственно свойства DrawWidth и DrawStyle объекта, на поверхности которого метод рисует.

Параметр *Step* показывает, что реальные координаты отсчитываются от указателя текущей точки.

Параметр *Color* задает цвет линии или (если задан параметр *B*) границы прямоугольника (по умолчанию цвет контура определяет свойство ForeColor графической поверхности).

Параметр *B* указывает на то, что надо нарисовать прямоугольник.

Параметр *F* говорит о том, что прямоугольник должен быть закрашен тем же цветом, что и граница.

Пример:

```
ScaleMode = 3 ' единица измерения координат - пикселы
Line (10, 20)-(100, 20) ' линия
Line (10,30)-(100,30), RGB(255,0,0) ' линия красного цвета
```

```
Line (10, 40)-(100, 50), , B ' прямоугольник
Line (10, 60)-(100, 70), RGB(0, 127, 0), B ' прямоугольник
Line (10, 80)-(100, 90), RGB(0, 127, 0), BF
```

LoadPicture

Объект.Picture = LoadPicture(*ФайлИллюстрации*)

Функция LoadPicture позволяет загрузить из файла (BMP, GIF, JPG) и отобразить в поле компонента PictureBox или Image иллюстрацию.

Пример:

```
Picture1.Picture = LoadPicture("D:\Images\image02.gif")
```

LoadResPicture

Объект.Picture = LoadResPicture(*ИдентификаторРесурса*, vbResBitmap)

Функция LoadResPicture позволяет загрузить из ресурса и отобразить в поле компонента PictureBox или Image иллюстрацию. Параметр *ИдентификаторРесурса* задает битовый образ, загружаемый из ресурса.

Пример:

```
Picture1.Picture = LoadResPicture(102, vbResBitmap)
```

PaintPicture

Pic1.PaintPicture
Pic2,*x1*,*y1*,*[w1]*,*[h1]*,*[x2]*,*[y2]*,*[w2]*,*[h2]*,*[OpCode]*

Метод PaintPicture копирует иллюстрацию (или ее часть) с поверхности объекта *Pic2* (типа Picture, Image или StdPicture) на поверхность объекта. *Pic1* (типа Picture) Параметры *x1*, *y1*, *w1*, *h1* задают ту область, куда копируется иллюстрация. Параметры *x2*, *y2*, *w2*, *h2* задают ту область, откуда копируется иллюстрация.

Пример:

```
Picture1.PaintPicture 0,0 Picture2

' копировать фрагмент рисунка Image1 на поверхность Picture1
Picture1.PaintPicture 0,0 Image1, 15,0,16,16

Dim aPicture As StdPicture
Set aPicture = LoadPicture("D:\Temp\image06.gif")
Picture1.PaintPicture aPicture, 0, 0
```

Print

```
[объект.]Print String
```

Метод `Print` выводит на поверхность объекта строку `String` от текущей точки (узнать координаты текущей точки можно, обратившись к свойствам `CurrentX` и `CurrentY`).

Шрифт определяется свойством `Font` графической поверхности (например, формы), на которую выводится текст, цвет символов — свойством `ForeColor` той же поверхности.

Пример:

```
Font.Name = "Arial"
Font.Size = 12
ForeColor = RGB(0, 0, 255)
ScaleMode = 3 ' координаты отсчитывать в пикселах
CurrentX = 10
CurrentY = 20
Print "Microsoft Visual Basic"
```

PSet

```
Объект.PSet (x, y)
```

Метод `PSet` рисует точку на поверхности графического объекта. Цвет и размер точки определяют соответственно свойства `Fore-`

Color и DrawWidth той графической поверхности, на которой рисует метод.

Пример:

```
Form1.Forecolor = RGB(56,176,222) ' цвет "Осеннее небо"
Form1.DrawWidth = 2
Form1.Pset(10,20)
```

RGB

RGB(*r*, *g*, *b*)

Функция RGB возвращает код цвета.

Параметры (*r*, *g*, *b*) задают долю красной (*r* — read), зеленой (*g* — green) и синей (*b* — blue) составляющих.

Диапазон изменения этих параметров — от 0 до 255.

В табл. 11.24 приведены значения параметров (*r*, *g*, *b*) и указан цвет, соответствующий сочетанию значений параметров.

Таблица 11.24. *Кодирование цвета*

Цвет	*r*	*g*	*b*
Аквамарин	112	219	147
Багряный	140	23	23
Васильковый	66	66	111
Весенне-зеленый, средний	127	255	0
Бирюзовый	173	234	234
Бирюзовый, средний	112	219	219
Бледно-зеленый	143	188	143
Бронзовый	140	120	83
Бронзовый 2	166	125	61
Весенне-зеленый	0	255	127
Голубой	35	107	142
Голубой кадет	95	159	159

Таблица 11.24 *(продолжение)*

Цвет	`r`	`g`	`b`
Древесный, темный	133	94	66
Древесный, светлый	233	194	166
Древесный, средний	166	128	100
Дымчато-серый	84	84	84
Желтовато-зеленый	153	204	50
Зеленовато-желтый	147	219	112
Зеленовато-медный	82	127	118
Зеленовато-медный, темный	74	118	110
Зеленый морской, средний	66	111	66
Золотарниковый	219	219	112
Золотарниковый, средний	234	234	174
Золотой	205	127	50
Индейский красный	78	47	47
Кварц	217	217	243
Кирпич	142	35	35
Коралловый	255	127	0
Коричневато-зеленый	50	205	50
Коричневый	166	42	42
Латунный	181	166	66
Лесной зеленый	35	142	35
Лиловый	153	50	205
Мандариновый (оранжевый)	228	120	51
Медный	184	115	51
Морской волны	35	142	104
Насыщенный синий	89	89	171
Небесно-голубой	50	153	204
Неоновый розовый	255	110	199

Таблица 11.24 *(продолжение)*

Цвет	`r`	`g`	`b`
Новый полуночно-синий	0	0	156
Оливково-зеленый, темный	79	79	47
Оранжево-красный	255	36	0
Оранжевый	255	127	0
Осеннее небо	56	176	222
Острый розовый	255	28	174
Охотничий зеленый	33	94	33
Охра	142	107	35
Очень светло-серый	205	205	205
Очень темно-коричневый	92	64	51
Перванш	0	127	255
Перванш, средний	127	0	255
Перванш, темный	107	35	142
Полевой шпат	209	146	117
Полуночно-голубой	47	47	79
Прохладный медный	217	135	25
Пшеничный	216	216	191
Розовый	188	143	143
Розовый, с серовато-коричневым оттенком	133	99	99
Рыжевато-коричневый	219	147	112
Рыжевато-коричневый, новый	235	199	158
Рыжевато-коричневый, темный	151	105	79
Светло-лиловый, средний	147	112	219
Светло-голубой	143	143	189
Светло-лиловый	219	112	219
Светло-серый	168	168	168

Таблица 11.24 *(окончание)*

Цвет	r	g	b
Светло-синий	192	217	217
Серебряный	230	232	250
Серый	192	192	192
Сине-фиолетовый	159	95	159
Сливовый	234	173	234
Сомон	111	66	66
Средне-синий	50	50	205
Средний лесной зеленый	107	142	35
Средний аквамарин	50	205	153
Старое золото	207	181	59
Темно-бирюзовый	112	147	219
Темно-бордовый	142	35	107
Темно-зеленый	47	79	47
Темно-коричневый	92	64	51
Темно-пурпурный	135	31	120
Темно-серый, аспидный	47	79	79
Темно-синий	35	35	142
Фиолетово-красный	204	50	153
Фиолетово-красный, средний	219	112	147
Фиолетовый	79	47	79
Хаки	159	159	95
Чертополох	216	191	216
Шоколадный, баркера	80	51	23
Шоколадный, полусладкий	107	66	38
Яркий золотой	217	217	25

Функции

В описании функций параметры выделены курсивом. Необязательные параметры заключены в квадратные скобки.

Ввод и вывод

Ввод исходных данных может быть реализован при помощи функции `InputBox`, а вывод результата — при помощи функции `MsgBox`.

InputBox

`InputBox(`*`Prompt`*`[,`*`Title`*`][,`*`Default`*`][,`*`X`*`][,`*`Y`*`][,`*`HlpFile`*`,`*`Cnt`*`])`

Функция `InputBox` выводит на экран диалоговое окно, в поле редактирования которого пользователь может ввести исходные данные — строку символов. Значением функции является введенная строка.

Параметр *`Prompt`* задает строку-подсказку, т. е. сообщение, которое отображается в диалоговом окне.

Параметр *`Title`* задает текст заголовка окна. Если этот параметр не указан, то в заголовке будет имя приложения, т. е. имя программы, которая запрашивает данные.

Параметр *`Default`* (выражение строкового типа) задает текст, отображаемый в поле ввода (пользователь может ввести исходные данные путем редактирования этого текста). Если параметр не указан, то при появлении окна на экране поле ввода будет пустым.

Параметры *`X`* и *`Y`* задают положение окна ввода. Параметры задаются в твипах. Если параметры не указаны, то окно будет выведено в центре экрана.

MsgBox

`MsgBox(`*`Prompt`*`[,`*`MsgType`* `[,`*`Title`*`][,`*`HlpFile`*`,`*`Cnt`*`])`

Функция `MsgBox` выводит на экран окно с сообщением. Значение функции — код кнопки, щелчком на которой пользователь закрыл окно.

Параметр *Prompt* (выражение строкового типа) задает текст сообщения.

Параметр *MsgType* (выражение целого типа) задает тип сообщения и командные кнопки, которые отображаются в окне сообщения (если параметр не указан, в окне сообщения отображается только кнопка **OK**). Необходимое значение этого параметра можно вычислить по формуле:

```
Msg + Btn,
```

где:

- `Msg` — тип сообщения:
  - `vbInformation(64)` — информационное;
  - `vbCritical(16)` — сообщение об ошибке;
- `Btn` — код кнопки (кнопок), которую надо отобразить в окне сообщения:
  - `vbOKOnly(0)` — отображается кнопка **OK**;
  - `vbOKCancel(1)` — отображаются кнопки **OK** и **Cancel**;
  - `vbYesNo(4)` — отображаются кнопки **Yes** и **No**.

Параметр *Title* задает заголовок окна сообщения. Если этот параметр не указан, то в заголовке отображается имя программы, которая вывела сообщение.

Параметр *HlpFile* задает файл справочной информации.

Параметр *Cnt* задает номер раздела справочной информации.

Чтобы получить доступ к справочной информации, пользователь должен нажать клавишу <F1>.

Математические функции

В табл. 11.25 приведены наиболее часто используемые математические функции.

Таблица 11.25. *Математические функции*

Функция	Значение
`Abs(N)`	Абсолютное значение (модуль) *N*
`Sqr(N)`	Квадратный корень *N*

Таблица 11.25 *(продолжение)*

Функция	Значение
`Exp(N)`	Экспонента *N*
`Sgn(N)`	Знак *N*. Если значение выражения *N* меньше нуля, то значение функции равно единице. Если значение выражения *N* больше или равно нулю, то значение функции равно нулю
`Rnd[(N)]`	Случайное число в диапазоне от 0 до *N* – 1. Если параметр *N* не указан, то значение функции — случайное число в диапазоне [0, 1]. Перед первым обращением к функции `Rnd` необходимо инициализировать генератор случайных чисел, т. е. вызвать функцию `Randomize`
`Int(N)`	Целая часть *N*. Значение получается путем отбрасывания дробной части. Если *N* отрицательное, то значение функции — ближайшее отрицательное целое число (меньшее либо равное *N*). Например: `Int(5.85)=5, Int(-5.85)=-6`
`Fix(N)`	Целая часть *N*. Значение получается путем отбрасывания дробной части числа. Для отрицательных чисел функция возвращает ближайшее отрицательное целое число (большее либо равное *N*). Например: `Fix(5.85)=5, Fix(-5.85)=-5`
`IsNumeric(S)`	Логическое значение. Если строка *S* (или подстрока от первого символа) является изображением числа, то значение функции `True`. Если строка не является изображением числа, то значение функции `False`.

Таблица 11.25 *(окончание)*

Функция	Значение
	Например: • `IsNumeric("5,85")` возвращает `True`; • `IsNumeric("5,8  5")` возвращает `True`; • `IsNumeric("hello")` возвращает `False`
`Log(N)`	Логарифм *N*. Функция вычисляет натуральный логарифм (логарифм по основанию *e*). Десятичный логарифм можно вычислить по формуле: `Log(N) / Log(10)`
`Sin(α)`	Синус угла α
`Cos(α)`	Косинус угла α
`Tan(α)`	Тангенс угла α
`Atn(α)`	Арктангенс угла α

Величина угла тригонометрических функций (`Sin`, `Cos`, `Tan`, `Atan`) должна быть указана в радианах. Для преобразования величины угла из градусов в радианы можно воспользоваться формулой:

```
(g * 3.1415926)/180
```

где: `g` — величина угла в градусах; `3.1415926` — число π.

Преобразование данных

В табл. 11.26 приведены наиболее часто используемые функции преобразования.

Таблица 11.26. *Функции преобразования данных*

Функция	Описание
`CBool(Expression)`	Преобразует выражение в тип `Boolean`. Например: • `CBool(5 > 4)` возвращает значение `True`; • `CBool(5 = 4)` возвращает значение `False`
`CDbl(Expression)`	Преобразует выражение в тип `Double`
`CInt(Expression)`	Преобразует выражение в тип `Integer`
`CLng(Expression)`	Преобразует выражение в тип `Long`
`CSng(Expression)`	Преобразует выражение в тип `Single`
`CVar(Expression)`	Преобразует выражение в тип `Variant`
`CDate(Expression)`	Преобразует выражение в тип `Date`
`CStr(Expression)`	Преобразует числовое выражение в строку

Работа со строками

В табл. 11.27 приведены наиболее часто используемые функции, обеспечивающие операции со строками.

Таблица 11.27. *Функции работы со строками*

Функция	Описание
`Chr(`*`Code`*`)`	Возвращает ANSI-символ, код которого равен *`Code`*. Значение *`Code`* должно лежать в промежутке от 0 до 255
`Asc(`*`Ch`*`)`	Возвращает код символа *`Ch`*. Если аргумент является строкой символов, то функция возвращает код первого символа строки

Таблица 11.27 *(продолжение)*

Функция	Описание
`InStr([Start,] String1,String2 [,Compare])`	Выполняет поиск подстроки в строке. Просмотр осуществляется слева направо от первого символа или (если присутствует параметр *Start*) от заданного параметром *Start*. Значением функции `InStr` — является номер позиции подстроки (символа) в строке. Если искомой подстроки в строке нет, то значение функции равно 0. Параметр *Sring1* — строка, в которой ведется поиск. Параметр *Sring2* — подстрока (символ), которую надо найти в строке *Sring1*. Параметр *Compare* задает режим сравнения строк: • `TextCompare(1)` — текстовое сравнение; • `BinaryCompare(0)` — побитовое сравнение. В режиме сравнения строк прописные и строчные символы считаются одинаковыми, а в режиме побитового сравнения — нет. Например: • `InStr(1, "Hh", "h", vbBinaryCompare)` = 2 • `InStr(1, "Hh", "h", vbTextCompare)` = 1
`InStrRev([,Start] String1, String2 [, Compare])`	Выполняет поиск подстроки в строке. Просмотр осуществляется справа налево от последнего или от заданного параметром *Start* символа. Значение функции — позиция подстроки (символа) в строке (положение найденной подстроки отсчитывается от первого (левого) символа). Если искомой подстроки в строке нет, то значение функции равно 0. Параметр *Sring1* — строка, в которой ведется поиск.

Таблица 11.27 *(продолжение)*

Функция	Описание
	Параметр *Sring2* — подстрока (символ), которую надо найти в строке *Sring1*. Параметр *Compare* задает режим сравнения строк: • `TextCompare(1)` — текстовое сравнение; • `BinaryCompare(0)` — побитовое сравнение. В режиме сравнения строк прописные и строчные символы считаются одинаковыми, а в режиме побитового сравнения — нет. Например: • `InStrRev(1, "HHt", "ht", vbBinary-Compare)` = 0; • `InStrRev(1, "Hht", "ht", vbText-Compare)` = 2
`Len(S)`	Возвращает длину строки *S* (количество символов в строке)
`LCase(S)`	Преобразует прописные символы строки *S* в строчные. Цифры и строчные буквы остаются без изменения
`UCase(S)`	Преобразует строчные символы строки *S* в прописные. Цифры и прописные буквы остаются без изменения
`Left(S, L)`	Возвращает первые (отсчет от начала строки, т. е. слева) *L* символов строки *S*. Если значение *L* больше, чем количество символов в строке *S*, то значением функции является сама строка *S*
`Right(S, L)`	Возвращает последние (отсчет от конца строки, т. е. справа) *L* символов строки *S*. Если значение *L* больше, чем количество символов в строке *S*, то значением функции является строка *S*
`LTrim(S)`	Удаляет пробелы в начале строки
`RTrim(S)`	Удаляет пробелы в конце строки

Таблица 11.27 *(продолжение)*

Функция	Описание
`Trim(S)`	Удаляет пробелы в начале и конце строки
`Mid(Str,Start [,Len])`	Возвращает подстроку, выделенную из строки *Str*. Параметр *Start* задает позицию подстроки, а *Len* — ее длину (число символов). Например: `Mid("Ms Visual Basic",4,6)` = `Visual`
`Space(N)`	Возвращает строку, состоящую из *N* пробелов
`String(N, Ch)`	Возвращает строку, состоящую из *N* символов *Ch*.
`StrReverse(S)`	Возвращает значение строки *S*, прочитанное справа налево. Например: `StrReverse("Hello")` = `"olleH"`

`Val(S)`	Возвращает числовое значение, изображением которого является строка *S*.
	Если в строке есть недопустимые символы, то будет обработана только та часть строки, которую можно преобразовать в число. Пробелы игнорируются.
	При обработке строки, являющейся изображением дробного числа, правильным символом-разделителем является точка.
	Если строку преобразовать в число нельзя, то значение функции `Val` равно нулю.
	Примеры:
	`Val(123 45) = 12345;`
	`Val(123,45) = 123;`
	`Val(123.45) = 123.45;`
	`Val("Text") = 0`

Таблица 11.27 *(окончание)*

Функция	Описание

`CDbl(S)`	Возвращает числовое значение, изображением которого является строка *S*.
	
	Если в строке есть недопустимые символы, то будет обработана только та часть строки, которую можно преобразовать в число. Пробелы игнорируются.
	
	При обработке строки, являющейся изображением дробного числа, правильным символом-разделителем является символ, заданный в настройке операционной системы (для России — это запятая).
	
	Если строку преобразовать в число нельзя, то возникает ошибка (исключение).
	
	Примеры:
	
	`CDbl(123 45) = 12345;`
	
	`CDbl(123,45) = 123,45;`
	
	`CDbl(123.45) = 123`

Работа с датами и временем

В табл. 11.28 приведены наиболее часто используемые функции манипулирования датами и временем.

Таблица 11.28. *Функции работы с датами и временем*

Функция	Значение
`Date`	Текущая дата по системному календарю компьютера
`Time`	Текущее время по системным часам компьютера
`Now`	Текущее время и дата по системным часам и календарю компьютера

Таблица 11.28 *(продолжение)*

Функция	Значение
`Year(Date)`	Год для заданной даты
`Month(Date)`	Номер месяца для заданной даты
`MonthName(Month [, Abbreviate])`	Функция возвращает полное или сокращенное (3 символа) название месяца по его номеру (аргумент `Month` должен быть числом в диапазоне от 1 до 12). Если значение аргумента `Abbreviate` равно `True`, то возвращается сокращенное название месяца. Если значение аргумента `Abbreviate` равно `False`, то возвращается полное название месяца
`Day(Date)`	День месяца (число от 1 до 31) для заданной даты
`Weekday(Date, [FirstDayOfWeek])`	Номер дня недели по дате `Date`. Аргумент `FirstDayOfWeek` определяет первый день недели: • `VbUseSystem(0)` — используются значения системных установок; • `Sunday (1)` — воскресенье; • `Monday(2)` — понедельник и т. д.
`WeekdayName (Weekday, Abbreviate, FirstDayOfWeek)`	Возвращает полное или сокращенное (2 символа) название дня недели по его номеру. Аргумент `Weekday` должен быть числом в диапазоне от 1 до 7. Если значение аргумента `Abbreviate` равно `True`, то возвращается сокращенное название месяца, если `False` — возвращается полное название.

Таблица 11.28 *(окончание)*

Функция	Значение
	Аргумент *FirstDayOfWeek* определяет первый день недели: • vbUseSystem(0) — используются значения системных установок; • Sunday(1) — воскресенье; • Monday(2) — понедельник и т. д.
Hour(*Time*)	Количество часов из выражения *Time*
Minute (*Time*)	Количество минут из выражения *Time*
Second (*Time*)	Количество секунд из выражения *Time*
Timer	Количество секунд с точностью до одной сотой, прошедших от полуночи до текущего момента времени

Работа с файлами

В табл. 11.29 приведены наиболее часто используемые функции, обеспечивающие операции с файлами.

Таблица 11.29. *Функции работы с файлами*

Функция	Описание
Open *PathName* For *Mode* [Access *Am*] [Lock] As #*FileNumber* [Len = *reclen*]	Открывает файл для выполнения операций чтения/записи. Параметр *PathName* задает имя файла, к которому надо получить доступ. Параметр *Mode* задает режим доступа к файлу: • Input — ввод даных (чтение); • Output — вывод данных (запись);

Таблица 11.29 *(продолжение)*

Функция	Описание
	• `Binary` — чтение/запись файла прямого доступа; • `Random` — чтение/запись текстового двоичного файла. Параметр *`Am`* задает операции, разрешенные для открытого файла: • `Read` (чтение); • `Write` (запись); • `Read Write` (чтение/запись). Параметр *`FileNumber`* задает номер файла (число в диапазоне от 1 до 511), используется в файловых операциях в качестве идентификатора файла. Параметр *`reclen`* задает длину записи файла (размер буфера), если файл открывается в режиме прямого доступа (*`Binary`*)
`Seek #`*`FileNumber`*`,` *`Position`*	Устанавливает указатель текущей позиции для выполнения операции чтения/записи файла, открытого в режиме прямого доступа (`Binary`). Параметр *`FileNumber`* — идентификатор файла. Параметр *`Position`* задает позицию (номер байта или записи), которую надо прочитать или перезаписать
`Seek( #`*`FileNumber`* `)`	Возвращает текущую позицию указателя чтения/записи для файла
`FreeFile[(`*`Range`*`)]`	Возвращает число, которое можно использовать в качестве идентификатора файла (параметра *`FileNumber`*) в функции `Open`

Таблица 11.29 *(продолжение)*

Функция	Описание
`Get #FileNumber, [RecNumber], VarName`	Считывает данные из файла: • `FileNumber` — номер (идентификатор) файла; • `RecNumber` — позиция (номер байта или номер записи, если файл открыт в режиме `Binary`), в которую надо установить указатель чтения перед выполнением операции; • `VarName` — переменная, в которую надо поместить данные
`Put #FileNumber, [RecNumber], VarName`	Записывает данные в файл: • `FileNumber` — номер (идентификатор) файла; • `RecNumber` — позиция (номер байта или номер записи, если файл открыт в режиме `Binary`), в которую надо установить указатель чтения перед выполнением операции; • `VarName` — переменная, в которой находятся данные
`Line Input #FileNumber, VarName`	Считывает строку из файла `FileNumber` и записывает ее в переменную `VarName`. Чтение происходит до тех пор, пока не будет обнаружен символ "новая строка" (код 13)
`Input #FileNumber, VarList`	Считывает данные из файла: • `FileNumber` — номер файла; • `VarList` — список переменных, значение которых надо прочитать из файла. Пример: `Input #1, a, b, c`
`Input(Number, #FileNumber)`	Считывает символьные или байтовые данные из файла, открытого в режиме `Input` или `Binary`: • `Number` — число считываемых символов или байтов;

Таблица 11.29 *(продолжение)*

Функция	Описание
	• *FileNumber* — номер файла. Пример: `IDChar = Input(1, #1)`
`Print #`*FileNumber*`,` `[`*OutputList*`]`	Записывает в заданный параметром *FileNumber* текст. Параметр *OutputList* (список вывода) — список выражений символьного типа. Пример: `Print #1, "a="+Str(a),"b="+Str(b)`
`Write #`*FileNumber*`,` `[`*OutputList*`]`	Записывает данные в файл. *OutputList* — записываемые данные (список переменных). Символьные данные в файле будут заключены в кавычки. Пример: `Write #1, a, b`
`FileLen(`*PathName*`)`	Возвращает длину файла в байтах
`LOF(`*FileNumber*`)`	Возвращает длину файла в байтах
`EOF(`*FileNumber*`)`	Проверяет положение указателя чтения/записи. Значение функции равно `True`, если достигнут конец файла (прочитан последний элемент данных)
`Dir[(`*Path* `[,`*Attributes*`])]`	Возвращает имя файла или папки, соответствующее критерию, заданному параметрами *Path* и *Attributes*. Если файлов (каталогов), удовлетворяющих указанным параметрам, нет, то значением функции является пустая строка (""). Если в качестве параметра *Path* задан шаблон имени файла (например, c:\temp*.bmp), то значение функции — имя файла, соответствующее шаблону. Чтобы получить имена остальных файлов, соответствующих данному шаблону, надо вызвать функцию `Dir` еще раз, но без параметров.

Таблица 11.29 *(продолжение)*

Функция	Описание

	Например:
	`fn =Dir("c:\temp\*.bmp")`
	`fn = fn + Chr(13) + Dir`
	Параметр *Attributes* задает (уточняет) тип файла:
	• `Normal(0)` — обычный;
	• `ReadOnly(1)` — "только для чтения";
	• `Hidden(2)` — скрытый;
	• `System(4)` — системный;
	• `Directory(16)` — каталог
	Примеры:
	`Dir("e:\test.txt")` возвращает `"test.txt"`, если файл test.txt имеется на диске E:;
	`Dir("e:\t\*.txt")` возвращает имя первого найденного в каталоге `e:\t` файла с расширением txt;
	`Dir("e:\",vbDirectory)` возвращает имя первой (по порядку) папки корневого каталога диска `e:`

`CurDir`	Возвращает полное имя текущей (рабочей) папки. Сразу после запуска программы текущая папка — это папка, из которой запущена программа
`ChDir` *Path*	Задает текущий (рабочий) каталог
`MkDir` *Path*	Создает новый каталог. Параметр *Path* задает путь к новому каталогу и имя каталога. При попытке создать каталог в несуществующей папке возникнет ошибка

Таблица 11.29 *(окончание)*

Функция	Описание
`RmDir` *`Path`*	Удаляет каталог. Параметр `Path` (полное имя каталога) задает каталог, который надо удалить. При попытке удалить каталог, в котором есть файлы, возникает ошибка. В этом случае нужно сначала из папки удалить файлы (функция `Kill`), а затем – каталог.
`Kill` *`PathName`*	Удаляет файл. Параметр *`PathName`* (полное имя файла) задает файл, который надо удалить. Если в качестве имени файла задать шаблон, то будут удалены все файлы, имена которых соответствуют указанному шаблону. Например: `Kill "c:\temp\*.tmp"`

Приложение

Описание прилагаемого компакт-диска

Прилагаемый к книге CD-ROM содержит проекты Visual Basic, приведенные в книге в качестве примеров. Каждый проект (табл. П1) находится в отдельном каталоге. Помимо файлов проекта в каждом каталоге есть выполняемый файл, что позволяет запустить программу из под Windows.

Большинство программ могут быть запущены непосредственно с CD-ROM без каких-либо предварительных действий (настроек системы). Некоторые программы (например, программы работы с базами данных) требуют дополнительной настройки системы (регистрации источника данных ODBC).

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера.

Таблица П1. *Содержимое CD-ROM*

Папка	Содержание	Глава
CD-плеер	Проигрыватель CD-дисков на базе компонента `MMControl` (Microsoft Multimedia Control). *Примечание.* Компонент `MMControl` (MCI32.OCX) должен быть зарегистрирован в системе	6
CheckBox	Программа позволяет вычислить стоимость заказа в кафе. Демонстрирует использование компонента `CheckBox`	3
ComboBox	Программа Любимый напиток демонстрирует использование компонента `ComboBox`	3

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
CommandButton	Программа позволяет пересчитать вес из фунтов в килограммы. Демонстрирует работу с "графической" кнопкой (компонент `CommandButton`), а также обработку события `KeyPress` для компонента `TextBox`	3
CommonDialog	Программа позволяет просмотреть иллюстрации. Выбор каталога (иллюстрации) выполняется в стандартном окне **Открыть файл**. Демонстрирует использование компонентов `CommonDialog`, `FileList`	4
Image	Программа позволяет просмотреть иллюстрации, которые находятся в том каталоге, из которого запущена программа. Демонстрирует использование компонента `Image`, а также функции `Dir`	3
Label	Программа демонстрирует влияние значений свойств компонента `Label` на вид текста, отображаемого в поле компонента	3
MEdit	Проект `MEdit` (простой редактор текста) демонстрирует использование компонентов `TextBox`, `CommonDialog`, работу с меню, выполнение операций чтения и записи текста в файл	3
MIDI	Игра "Угадай число". Демонстрирует использование компонента `MMControl` для воспроизведения MIDI-мелодии. *Примечание*. Компонент `MMControl` (MCI32.OCX) должен быть зарегистрирован в системе	6
MP3 плеер	MP3-плеер на базе компонента `MMControl` (Microsoft Multimedia Control) с регулятором громкости. *Примечание*. Компонент `MMControl` (MCI32.OCX) должен быть зарегистрирован в системе	6

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
Menu	Программа Medit — простой редактор текста. Демонстрирует использование компонентов `Menu`, `TexBox`, `CommonDialog`, чтение текста из файла, запись текста в файл	4
OptionButton	Программа Кафе демонстрирует использование компонентов `OptionButton` и `Frame`	3
PictureBox	Программа строит график (изменения курса доллара). Демонстрирует использование компонента `PictureBox` в качестве фонового рисунка (рисунок задается во время создания формы), а также вывод графики на поверхность компонента `PictureBox`	3
PlaySound	Демонстрирует использование API-функции `PlaySound` для воспроизведения звука	6
ProgressBar	В качестве индикатора времени используется компонент `ProgressBar`	4
StatusBar	Программа демонстрирует назначение и возможности компонента `StatusBar`	4
TextBox	Проект демонстрирует использование компонента `TextBox` для ввода данных разного типа. Показывает, как можно выполнить фильтрацию символов	3
Timer	Будильник. Программа выводит сообщение в заданный пользователем момент времени. Появление окна сообщения сопровождается звуковым сигналом. Воспроизведение звука обеспечивает функция `PlaySound`. Программа демонстрирует использование компонентов `Timer`, `UpDown` и др.	3
UpDown	Таймер. Программа выводит сообщение по прошествии заданного пользователем промежутка времени. Появление окна сообщения сопровождается звуковым сигналом. Воспроизведение звука обеспечивает функция `PlaySound`. Программа демонстрирует использование компонентов `UpDown`	3

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
Video плеер	Программа позволяет просмотреть видеоролик. Основой программы является компонент `MMControl`. *Примечание*. Компонент `MMControl` (`MCI32.OCX`) должен быть зарегистрирован в системе	6
usd2rub	Программа пересчитывает цену из долларов в рубли. Демонстрирует обработку события `Click`, преобразование строки в число и числа в строку. Программа спроектирована так, что в поля редактирования можно ввести только правильные данные (дробные числа)	2
Адресная книга	Программа для работы с базой данных **Адресная книга**. Позволяет вводить, редактировать и удалять записи. База данных **Адресная книга** (формат Microsoft Access) состоит из одной-единственной таблицы `Contacts`, в которой пять столбцов: • `Title` — `Char(50)`; • `Phone` — `Char(50)`; • `Manager` — `Char(50)`; • `Address` — `Char(50)`; • `EMail` — `Char(50)`. Программа демонстрирует использование компонентов: `Adodc` — для доступа к данным; `TextBox` — для отображения данных. *Внимание!* База данных должна быть зарегистрирована в системе как источник данных adrbk. Зарегистрировать базу данных можно при помощи Администратора источников данных (**Пуск ▸ Настройка ▸ Панель управления ▸ Администрирование ▸ Источники данных (ODBC)**). В процессе регистрации необходимо выбрать драйвер Microsoft Access Driver (*.mdb)	7

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
Адресная книга-2	Программа для работы с базой данных **Адресная книга**. Позволяет вводить, редактировать и удалять записи. База данных **Адресная книга** (adrbk.mdb, формат Microsoft Access) состоит из одной-единственной таблицы `Contacts`. в которой пять столбцов: • `Title` — `Char(50)`; • `Phone` — `Char(50)`; • `Manager` — `Char(50)`; • `Address` — `Char(50)`; • `EMail` — `Char(50)`. Программа демонстрирует использование компонентов: • `Adodc` — для доступа к данным; • `DataGrid` — для отображения данных в режиме таблицы; • `TextBox` — для отображения данных в режиме формы. *Внимание!* База данных adrbk.mdb должна быть зарегистрирована в системе как источник данных ODBC adrbk. Зарегистрировать базу данных можно при помощи Администратора источников данных (**Пуск ▸ Настройка ▸ Панель управления ▸ Администрирование ▸ Источники данных (ODBC)**). В процессе регистрации необходимо выбрать драйвер Microsoft Access Driver (*.mdb)	7
Битовый образ	Программа показывает, как вывести рисунок с "прозрачным" фоном	5

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
Записная книжка	Программа для работы с базой данных **Записная книжка**. Позволяет вводить, редактировать и удалять записи. Демонстрирует использование компонентов `Adodc` и `DataGrid`. База данных **Записная книжка** (phbk.mdb) состоит из одной-единственной таблицы `Phones`, в которой два столбца: `Title` и `Phone`. *Внимание!* База данных должна быть зарегистрирована в системе как источник данных phbk. Зарегистрировать базу данных можно при помощи Администратора источников данных (**Пуск ▸ Настройка ▸ Панель управления ▸ Администрирование ▸ Источники данных (ODBC)**). В процессе регистрации необходимо выбрать драйвер Microsoft Access Driver (*.mdb)	7
Записная книжка-2	Программа для работы с базой данных **Записная книжка-2**. Позволяет вводить, редактировать и удалять записи, а также выбирать нужную информацию по содержимому поля. Демонстрирует использование компонентов `Adodc` и `DataGrid`. База данных **Записная книжка-2** (phbk.mdb, формат Microsoft Access) состоит из одной-единственной таблицы `Phones`, в которой два столбца: `Title` и `Phone`. *Внимание!* База данных должна быть зарегистрирована в системе как источник данных phbk. Зарегистрировать базу данных можно при помощи Администратора источников данных (**Пуск ▸ Настройка ▸ Панель управления ▸ Администрирование ▸ Источники данных (ODBC)**). В процессе регистрации необходимо выбрать драйвер Microsoft Access Driver (*.mdb)	7

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
Линия	Программа демонстрирует влияние значений свойства `DrawStyle` на вид линии, которую рисует метод `Line`	5
Мультипликация	Программа демонстрирует принципы создания анимации. Изображение формируется непосредственно на поверхности формы. Битовые образы загружаются из файлов	5
Мультипликация-2	Программа демонстрирует, как можно сформировать картинку на невидимой графической поверхности (в памяти) и затем вывести ее на поверхность формы. Битовые образы загружаются из файлов. Формирование картинки в памяти осуществляют API-функции	5
Мультипликация-3	Программа демонстрирует, как можно сформировать картинку на невидимой графической поверхности (в памяти) и затем вывести ее на поверхность формы. Битовые образы загружаются из ресурса. Формирование картинки в памяти осуществляют API-функции	5
Окружность	Программа демонстрирует использование метода `Circle`, а также показывает, как разместить рисунок и текст по центру окна (при изменении ширины окна рисунок и текст всегда находятся в центре)	5
Пинг-понг	Программа Пинг-понг показывает, как заставить двигаться объект. Эта программа — пример простой интерактивной игры, она показывает, как обеспечить перемещение объекта в зависимости от действий пользователя	5
Просмотр иллюстраций	Программа демонстрирует использование компонента `Image` для отображения иллюстрации, размер которой больше размера компонента. Для выбора папки, в которой находятся иллюстрации, используется стандартное окно **Обзор папок**	5

Таблица П1 *(продолжение)*

Папка	Содержание	Глава
Прямоугольник	Программа демонстрирует влияние значений свойств `DrawStyle`, `DrawWidth`, `FillStyle` и `FillColor` на вид прямоугольника, который рисует метод `Line`	5
Регулятор громкости	Каталог содержит два проекта. Проект vave_vc показывает, как можно регулировать громкость воспроизведения WAV-(MP3-) файла. В качестве регулятора громкости используется компонент `Slider`. Проект midi_vc показывает, как можно регулировать громкость воспроизведения MIDI-файла. В качестве регулятора громкости используется компонент `UpDown`	6
Сапер	Игра Сапер демонстрирует работу с графикой, вывод дочернего окна (модального диалога), отображение справочной информации	10
Сектор	Программа показывает, как при помощи метода `Circe` нарисовать сектор	5
Создать БД	Программа показывает, как можно создать базу данных (файл базы данных) не прибегая к помощи СУБД. Таблица в БД создается в результате выполнения соответствующей SQL-команды. Следует обратить внимание, что программа позволяет зарегистрировать созданную базу данных в системе как источник данных ODBC	7
Справка	Программа демонстрирует различные методы отображения справочной информации. В каталоге Chm находятся файлы, необходимые для создания файла справочной системы формата MS HTML Help: файлы справочной информации (*.htm), файлы проекта (hhp) и контекста (hhc) справочной системы, а также файл определения идентификаторов разделов (h)	8

Таблица П1 *(окончание)*

Папка	Содержание	Глава
Текст	В зависимости от времени суток программа приветствует пользователя фразой "Доброе утро", "Добрый день", "Добрый вечер" или "Доброй ночи". Демонстрирует вывод текста на поверхность формы при помощи метода `Print`, а также использование функций `Format`, `TextWidth` и `TextHeight`	5

Экзаменатор	Универсальная программа тестирования. Имя файла теста надо указать в команде запуска программы. Например, если программа тестирования (exam.exe) и файл теста (экономика.txt) находятся в каталоге d:\exam, то команда запуска программы выглядит так:
	`d:\exam\exam.exe`
	`d:\exam\экономика.txt`
	Команду надо ввести в окне **Запуск программы**, которое становится доступным в результате выбора в меню **Пуск** команды **Выполнить**
		10

Эллипс	Программа демонстрирует использование метода `Circle` для вычерчивания эллипсов	5

Предметный указатель

T

U

V

W

Y

Б

В

Г

Д

З

И

К

Л

М

О

П

Р

С

Т

У

Ф

Ц

Э