Никита Культин

Основы программирования в Delphi XE

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.068+800.92Delphi XE
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Основы программирования в Delphi XE. — СПб.: БХВ-Петербург, 2011. — 416 с.: ил. + CD-ROM — (Самоучитель)

ISBN 978-5-9775-0683-0

Книга является пособием для начинающих по программированию в Delphi. В ней в доступной форме изложены принципы визуального проектирования и событийного программирования, на конкретных примерах показана методика создания программ различного назначения, приведено описание среды разработки Delphi XE и базовых компонентов. Рассмотрены вопросы программирования графики, мультимедиа, разработки программ работы с базами данных Microsoft Access. Многочисленные примеры демонстрируют назначение компонентов, раскрывают тонкости программирования в Delphi. В приложении приведено описание базовых компонентов и наиболее часто используемых функций. Книга отличается доступностью изложения, большим количеством примеров. Прилагаемый компакт-диск содержит проекты, приведенные в книге.

Для начинающих программистов

УДК 681.3.068+800.92Delphi XE
ББК 32.973.26-018.1

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.01.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 33,54.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09
от 26.05.2009 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 978-5-9775-0683-0

Оглавление

Предисловие

Среда разработки Delphi является одним из популярнейших инструментов разработки прикладных программ. Она поддерживает так называемую быструю разработку, основанную на технологии визуального проектирования и событийного программирования, суть которой состоит в том, что среда разработки берет на себя большую часть рутины, оставляя программисту работу по созданию диалоговых окон (визуальное проектирование) и процедур обработки событий (событийное программирование). Производительность программиста при этом просто фантастическая!

Delphi — это среда быстрой разработки приложений (RAD-среда, от *Rapid Application Development* — быстрая разработка приложений) на языке Delphi, в основе которого лежит хорошо знакомый многим программистам язык Pascal.

Изначально, вплоть до седьмой версии, Delphi была ориентирована на разработку Win32-приложений. После того как Microsoft стала продвигать технологию .NET, появилась Delphi 8 for The Microsoft .NET Framework — среда разработки .NET-приложений. Следующие версии Delphi выпускались в двух вариантах: для разработки Win32- и .NET-приложений. Теперь программистам стала доступна очередная версия Delphi — Embarcadero Delphi XE. Embarcadero — новое имя выделенного из Borland подразделения (изначально оно называлось CodeGear), отвечающего за инструменты разработки приложений.

Delphi XE существует в трех вариантах: Professional, Enterprise и Architect. Каждый комплект включает набор средств и компонентов, обеспечивающих разработку высокоэффективных приложений различного назначения, в том числе работы с базами данных InterBase, Blackfish SQL, Firebird, MySQL, Microsoft SQL Server, Oracle и др. Чем выше уровень пакета, тем больше возможностей он предоставляет программисту. Так, например, в Enterprise и Architect есть компоненты, позволяющие работать с удаленным сервером Blackfish SQL, а в Professional — только с локальным.

Среда Delphi XE доступна как отдельный инструмент разработки, а также как элемент Embarcadero RAD Studio XE.

Delphi XE может работать в среде операционных систем Microsoft Windows XP Home или Professional (SP2 или SP3), Microsoft Windows Vista SP2, Microsoft Windows Server 2003 (SP1) или 2008, а также в Microsoft Windows 7. Особых требований, по современным меркам, к ресурсам компьютера среда не предъявляет: процессор должен быть класса Intel Pentium (или совместимый) с частотой 1,4 ГГц (рекомендуется 2 ГГц и выше), 1 Гбайт оперативной памяти (рекомендуется 2 Гбайт и больше), 3,75 Гбайт свободного места на жестком диске (в том числе 750 Мбайт для Microsoft .NET Framework и Microsoft .NET SDK).

Книга, которую вы держите в руках, — это не описание среды разработки или языка программирования, это руководство по программированию в Delphi XE, разработке Win32-приложений. В ней представлена технология визуального проектирования и событийного программирования, подробно рассмотрен процесс создания программы, показано назначение базовых компонентов, описан процесс создания справочной системы и образа установочного диска, уделено внимание вопросам программирования графики, мультимедиа, разработки программ работы с базами данных.

Цель книги — научить программировать, создавать программы различного назначения: от простых однооконных приложений до программ работы с базами данных. Следует обратить внимание, что хотя книга ориентирована на читателя, обладающего начальными знаниями и опытом в программировании, она вполне доступна и начинающим.

Научиться программировать можно, только программируя, решая конкретные задачи. Поэтому чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте листинги, старайтесь понять, как работают программы. Не бойтесь экспериментировать — совершенствуйте программы, вносите в них изменения. Чем больше вы сделаете самостоятельно, тем большему научитесь!

ЧАСТЬ I

DELPHI XE

В *данной части* книги приведено краткое описание среды разработки Delphi XE; на примере программы "Конвертер" показан процесс разработки приложения; приведены описание и примеры использования базовых компонентов.

Глава 1

Среда разработки Delphi XE

Установка

Основным вариантом поставки RAD Studio является "установка с сервера (Download)": при покупке программист получает серийный номер и ссылку на программу активизации установки, а все необходимые для установки файлы загружаются с сервера Embarcadero Technologies. При желании программист может за отдельную плату заказать DVD-диск и в дальнейшем использовать его, например, для установки отсутствующих компонентов.

Чтобы установить Delphi, надо с сайта компании Embarcadero Technologies загрузить пакет установки, представляющий собой ZIP-архив, распаковать его во временный каталог и запустить установщик (файл install_RADStudio.exe).

Delphi XE является .NET-приложением. Поэтому установка начинается с проверки наличия на компьютере разработчика Microsoft .NET Framework 3.5 SP1 Redistributable Package, Microsoft Visual J# version 2.0 Redistributable Package, Microsoft Data Access Components (MDAC) 2.8, Microsoft Core XML Services (MSXML) 6.0 и Language Pack for Microsoft .NET Framework 2.0. Если какой-либо из перечисленных компонентов отсутствует, то он устанавливается. После этого начинается установка Delphi.

Процесс установки обычный. Сначала на экране появляется окно лицензионного соглашения, затем — окно **Select Features**, в котором программист может выбрать необходимые для работы компоненты (точнее, отказаться от установки ненужных). По умолчанию на компьютер устанавливаются все доступные компоненты и, если на жестком диске достаточно свободного места, то в окне **Select Features** лучше ничего не трогать.

По окончании установки необходимо выполнить активацию продукта — ввести в окне активации серийный номер. Серийный номер передается на сервер регистрации, который в ответ пересылает на компьютер программиста файл активации. На этом процесс установки можно считать законченным.

Первое знакомство

Чтобы запустить Delphi XE, надо сделать щелчок на кнопке **Пуск** и в меню **Все программы** выбрать команду **Embarcadero RAD Studio XE ▸ Delphi XE**.

Затем, чтобы начать работу над новым *проектом*, надо в меню **File** выбрать команду **New ▸ VCL Forms Application — Delphi**.

Окно среды Delphi XE в начале работы над новым проектом приведено на рис. 1.1. В заголовке окна отображается имя проекта, над которым в данный момент работает программист. В верхней части окна находится строка меню и область отображения панелей инструментов.

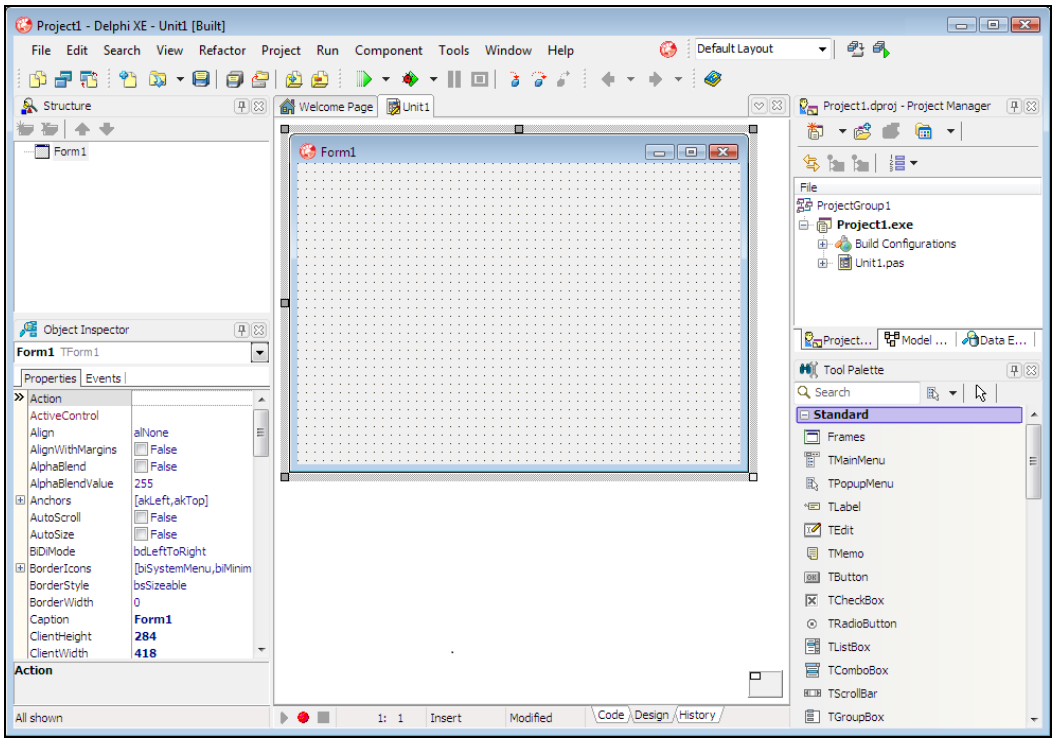


Рис. 1.1. Окно среды Delphi XE
в начале работы над новым проектом

Центральную часть окна среды Delphi XE занимает окно конструктора (дизайнера) формы (рис. 1.2). В нем находится *форма* — заготовка окна приложения (окно программы во время его разработки принято называть формой).

За окном конструктора формы находится окно редактора кода (рис. 1.3). Доступ к окну редактора кода можно получить, сделав щелчок кнопкой мыши на ярлыке **Code** или нажав клавишу <F12> (повторное нажатие <F12> или щелчок кнопкой мыши на ярлыке **Design** активизирует конструктор формы).

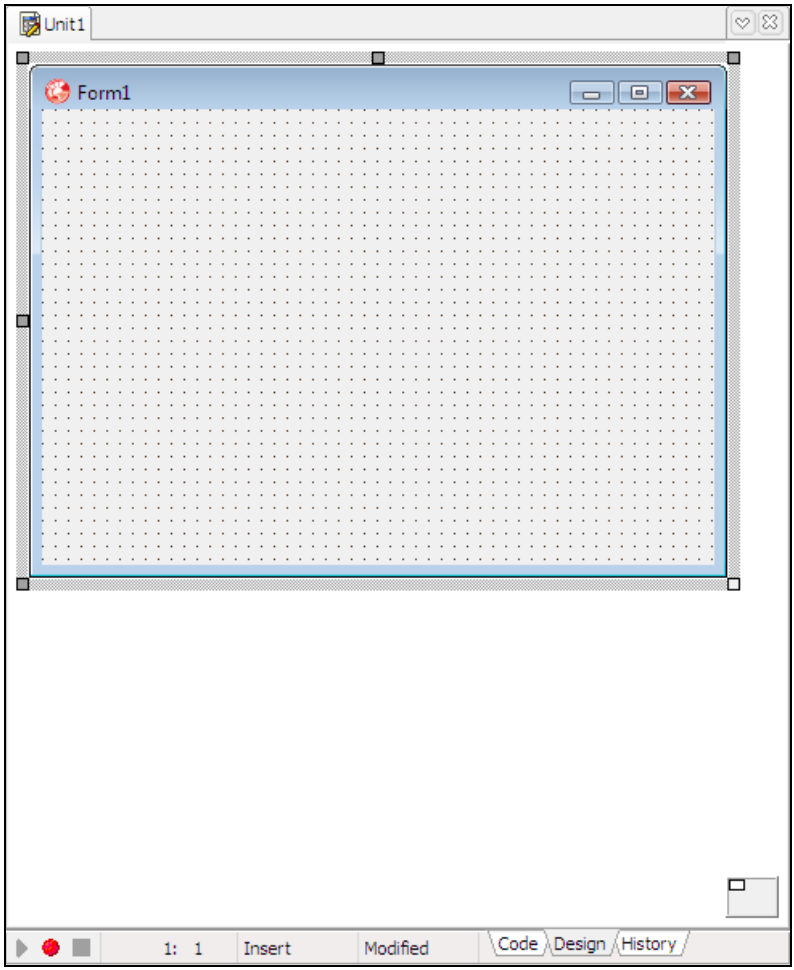


Рис. 1.2. Окно конструктора формы

Слева от окна дизайнера формы находится окно **Object Inspector** (рис. 1.4). Вкладка **Properties** этого окна используется для редактирования значений *свойств объектов*. Свойство (property) — это характеристика *объекта* (формы, командной кнопки, поля редактирования и т. д.). Свойства определяют вид объекта, его положение и поведение. Например, свойство `Caption` формы определяет текст, который отображается в ее заголовке, а свойства `Width` и `Height` — ее размеры (ширину и высоту). Справа от названия свойств указаны их значения. Свойства по функциональному признаку объединены в группы (названия групп выделены цветом). Так, например, свойства, определяющие внешний вид объекта, объединены в группу **Visual**. Программист может изменить способ отображения свойств, выбрав в контекстном меню вкладки **Properties** команду **Arrange ▸ by Name** (В алфавитном порядке) или **Arrange ▸ by Category** (По категориям). На вкладке **Events** окна **Object Inspector** перечислены *события*, которые может воспринимать объект.

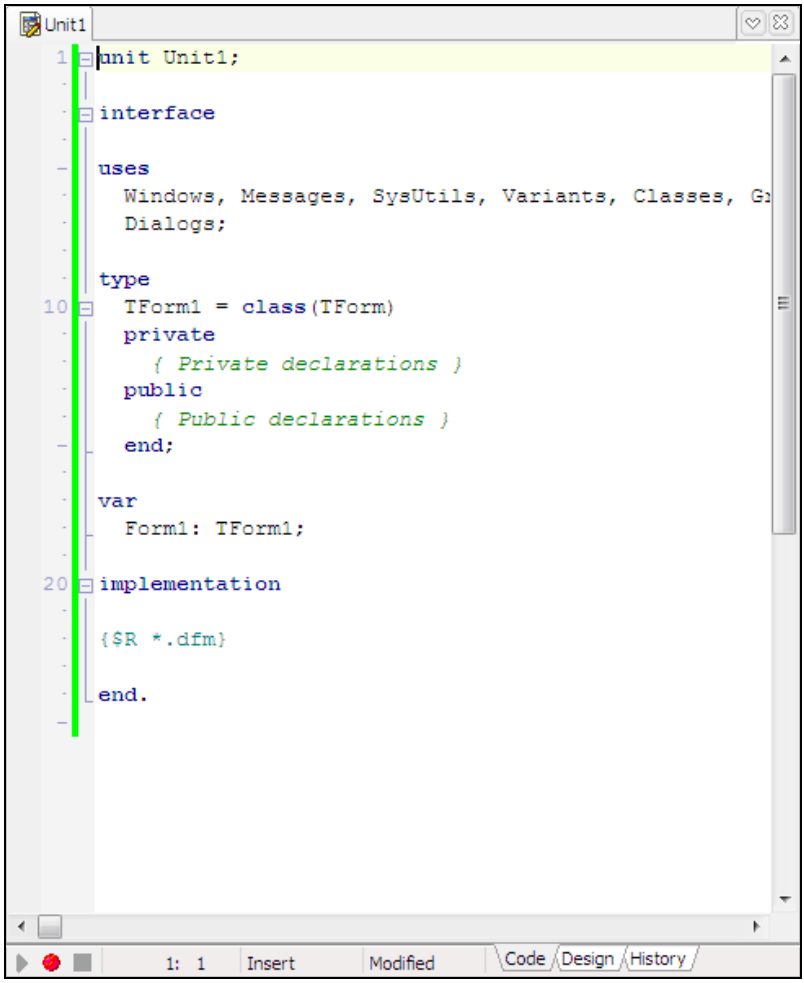


Рис. 1.3. Окно редактора кода

На вкладках палитры компонентов (окно **Tool Palette**) находятся *компоненты* (рис. 1.5). Компонент — это элемент пользовательского интерфейса или объект, реализующий некоторую функциональность. Например, на вкладке **Standard** находятся компоненты, обеспечивающие взаимодействие с пользователем (`Label` — поле отображения текста; `Edit` — поле редактирования; `Button` — командная кнопка и др.), а на вкладке **dbGo** — компоненты доступа к базам данных.

Компоненты, обеспечивающие взаимодействие с пользователем, объединены в так называемую VCL-библиотеку (Visual Components Library). Поэтому приложения, использующие эти компоненты, называются VCL-приложениями (VCL Forms Application). Вспомните, чтобы начать работу над новым проектом (новой программой), надо в меню **File** выбрать команду **New ▸ VCL Forms Application**.

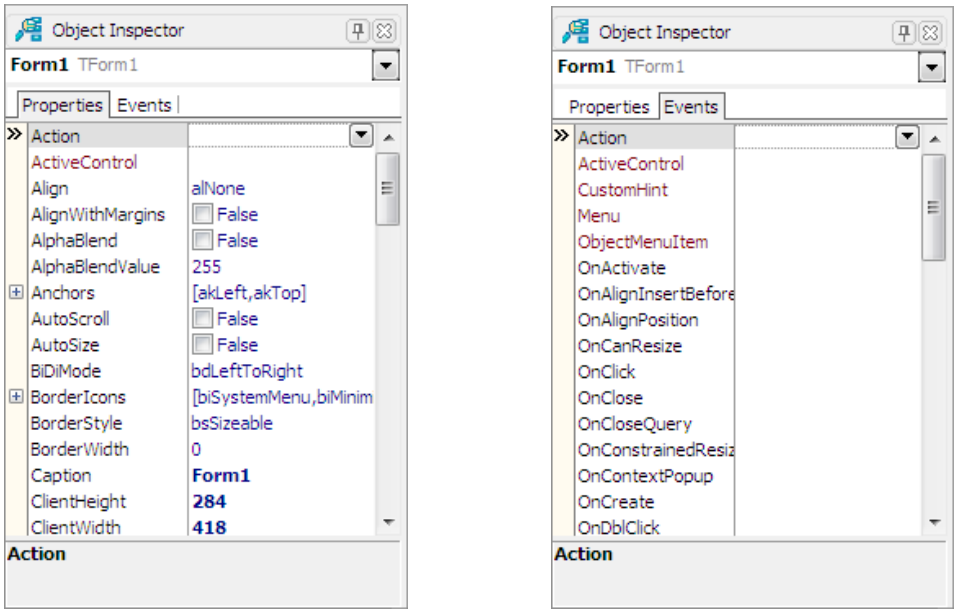


Рис. 1.4. В окне **Object Inspector** на вкладке **Properties** перечислены свойства объекта, а на вкладке **Events —** события, на которые объект может реагировать

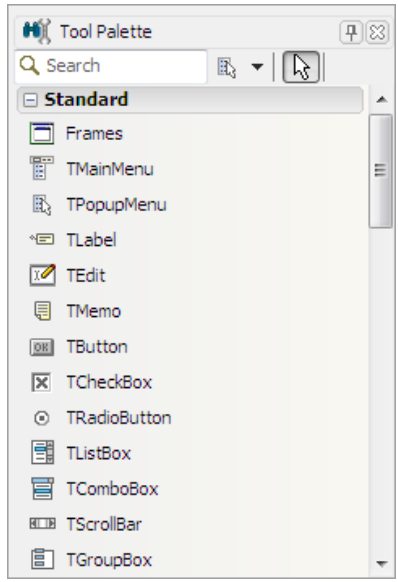


Рис. 1.5. Вкладка **Standard** содержит компоненты, обеспечивающие взаимодействие пользователя с программой

В окне **Project Manager** (рис. 1.6) отображается структура проекта, над которым в данный момент идет работа.

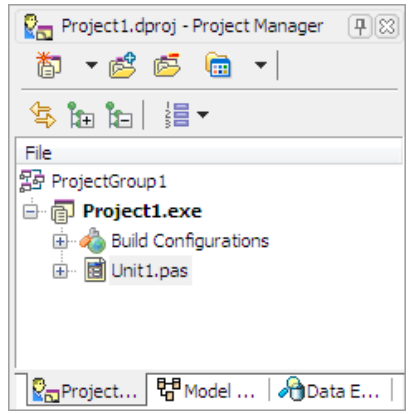


Рис. 1.6. В окне **Project Manager** отображается структура проекта

Если какое-либо из перечисленных окон на экране отсутствует, то для того чтобы его увидеть, надо в меню **View** выбрать соответствующую команду.

Глава 2

Первый проект

Процесс разработки программы в Delphi рассмотрим на примере — создадим *приложение* (так принято называть прикладную программу), с помощью которого можно пересчитать цену из долларов в рубли (рис. 2.1).

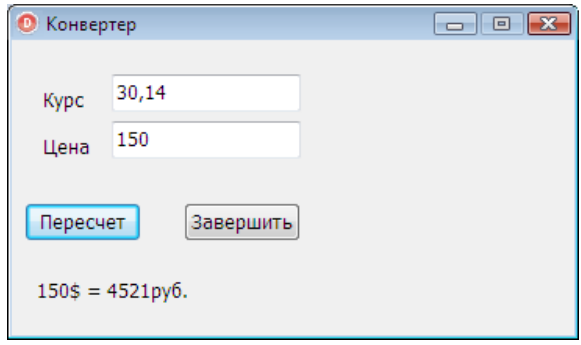


Рис. 2.1. Окно программы "Конвертер"

Начало работы

Чтобы начать работу над новым приложением, нужно в меню **File** выбрать команду **New ▸ VCL Forms Application - Delphi**.

Форма

Работа над приложением начинается с создания стартовой *формы* — главного окна программы.

Сначала нужно установить требуемые значения формы, затем — поместить на форму необходимые *компоненты* (поля ввода информации, командные кнопки, поля отображения текста и др.).

Настройка формы (а также компонентов) осуществляется путем изменения значений *свойств*. Свойства *объекта* (формы, компонента) определяют его вид и поведение. Например, свойство `Caption` определяет текст заголовка окна, а свойство `Position` — положение окна в момент появления на экране.

Основные свойства формы (объекта `TForm`) приведены в табл. 2.1.

Таблица 2.1. *Свойства формы (объекта `TForm`)*

Свойство	Описание
`Name`	Имя (идентификатор) формы. Используется для доступа к форме, ее свойствам и методам, а также для доступа к компонентам формы
`Caption`	Текст заголовка
`Width`	Ширина формы
`Height`	Высота формы
`Position`	Положение окна в момент первого его появления на экране (`poCenterScreen` — в центре экрана; `poOwnerFormCenter` — в центре родительского окна; `poDesigned` — положение окна определяют значения свойств `Top` и `Left`)
`Top`	Расстояние от верхней границы формы до верхней границы экрана
`Left`	Расстояние от левой границы формы до левой границы экрана
`BorderStyle`	Вид границы. Граница может быть обычной (`bsSizeable`), тонкой (`bsSingle`) или вообще отсутствовать (`bsNone`). Если у окна обычная граница, то во время работы программы пользователь может с помощью мыши изменить размер окна. Изменить размер окна с тонкой границей нельзя. Если граница отсутствует, то на экран во время работы программы будет выведено окно без заголовка. Положение и размер такого окна во время работы программы изменить нельзя
`BorderIcons`	Кнопки управления окном. Значение свойства определяет, какие кнопки управления окном будут доступны пользователю во время работы программы. Значение свойства задается путем присвоения значений уточняющим свойствам `biSystemMenu`, `biMinimaze`, `biMaximaze` и `biHelp`. Свойство `biSystemMenu` определяет доступность кнопки системного меню (значок в заголовке окна), `biMinimaze` — кнопки **Свернуть**, `biMaximaze` — кнопки **Развернуть**, `biHelp` — кнопки вывода справочной информации
`Icon`	Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню
`Color`	Цвет фона. Цвет можно задать, указав название цвета или привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой, выбранным компонентом привязки и меняется при изменении цветовой схемы операционной системы

***Таблица 2.1** (окончание)*

Свойство	Описание
`Font`	Шрифт. Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение свойства `Font` формы приводит к автоматическому изменению свойства `Font` компонента, располагающегося на поверхности формы. То есть компоненты наследуют свойство `Font` от формы (имеется возможность запретить наследование)

Для изменения значений свойств объектов используется вкладка **Properties** окна **Object Inspector**. В левой колонке этой вкладки перечислены свойства объекта, *выбранного* в данный момент, в правой — указаны значения свойств. Имя выбранного объекта отображается в верхней части окна **Object Inspector**.

На вкладке **Properties** свойства объединены в группы по функциональному признаку (названия групп выделены цветом). Например, группа **Visual** содержит свойства, определяющие вид объекта (для формы — заголовок, цвет фона, вид границы), а группа **Layout** — свойства, определяющие положение объекта (для формы — координаты левого верхнего угла). Некоторые свойства, например `Width` и `Height`, отображаются в нескольких группах (**Visual** и **Layout**).

Программист может изменить способ отображения свойств в окне **Object Inspector**. Например, чтобы свойства отображались в алфавитном порядке, в контекстном меню вкладки **Properties** надо выбрать команду **Arrange ▸ by Name**.

Чтобы в заголовке формы вместо `Form1` появилось название программы — текст `Конвертер`, следует изменить значение свойства `Caption`. Чтобы это сделать, надо в окне **Object Inspector** щелкнуть левой кнопкой мыши в строке свойства (в результате будет выделено текущее значение свойства и появится курсор), ввести текст `Конвертер` и нажать клавишу <Enter> (рис. 2.2).

Аналогичным образом можно установить значения свойств `Height` и `Width`, которые определяют высоту и ширину формы. Размер формы, а также размер других компонентов, задают в пикселах (точках). Свойствам `Height` и `Width` надо присвоить значения 215 и 366 соответственно.

Размер формы можно изменить и с помощью мыши, точно так же, как и любого окна, т. е. путем перемещения границы. По окончании перемещения границы значения свойств `Height` и `Width` будут соответствовать установленному размеру формы.

Положение окна на экране в момент его первого появления соответствует положению формы, заданному во время разработки программы. Положение можно задать, установив значение свойств `Top` (отступ от верхней границы экрана) и `Left` (отступ от левой границы экрана) или задав значение свойства `Position`.

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. 2.3).

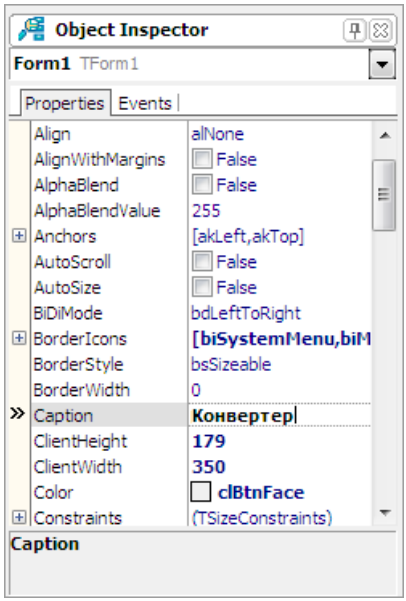


Рис. 2.2. Изменение значения свойства `Caption` путем ввода нового значения

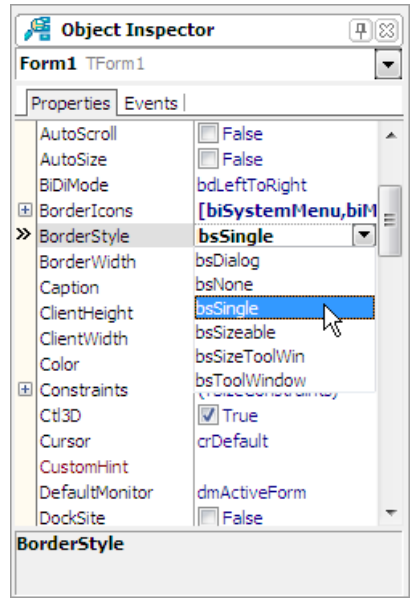


Рис. 2.3. Установка значения свойства путем выбора из списка

Некоторые свойства являются сложными, т. е. их значение определяется совокупностью значений других (уточняющих) свойств. Например, свойство `BorderIcons` определяет кнопки управления окном, которые будут доступны во время работы программы. Значения этого свойства определяются совокупностью значений свойств `biSystemMenu`, `biMinimize`, `biMaximize` и `biHelp`, каждое из которых, в свою очередь, определяет наличие соответствующей командной кнопки в заголовке окна во время работы программы. Перед именами сложных свойств стоит значок "+", в результате щелчка на котором раскрывается список уточняющих свойств (рис. 2.4). Значение уточняющего свойства можно задать обычным образом (ввести значение в поле редактирования или выбрать в списке).

В результате выбора некоторых свойств, например `Font`, в поле значения свойства отображается кнопка, на которой изображены три точки. Это значит, что задать значение свойства можно в дополнительном диалоговом окне, которое появится в результате щелчка на этой кнопке. Например, значение свойства `Font` можно задать путем ввода значений уточняющих свойств (`Name`, `Size`, `Style` и др.), а можно воспользоваться стандартным диалоговым окном **Шрифт**, которое появится в результате щелчка на кнопке с тремя точками (рис. 2.5).

В табл. 2.2 приведены значения свойств стартовой формы программы "Конвертер". Значения остальных свойств формы оставлены без изменения и поэтому в таблице не представлены. Обратите внимание, в именах некоторых свойств есть точка. Это значит, что это значение уточняющего свойства.

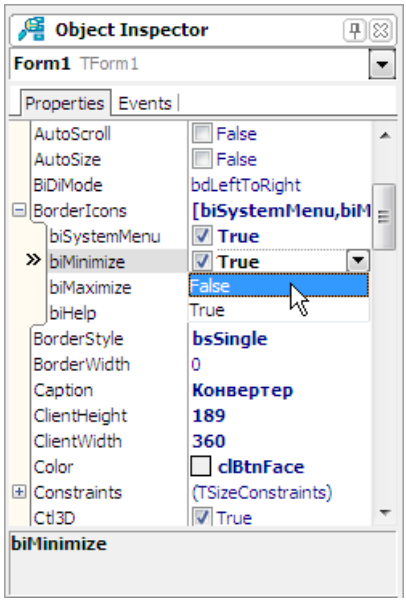


Рис. 2.4. Изменение значения уточняющего свойства

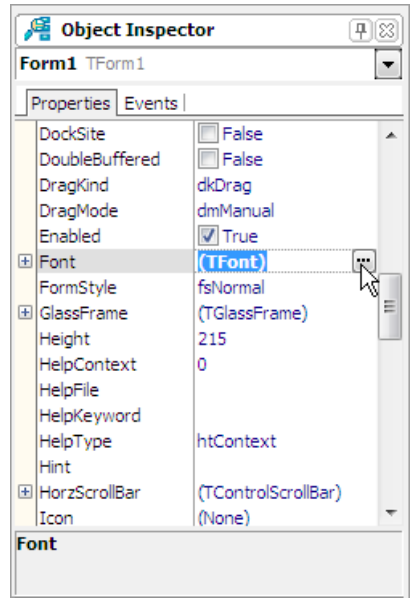


Рис. 2.5. Чтобы задать свойства шрифта, щелкните на кнопке с тремя точками

Таблица 2.2. *Значения свойств стартовой формы*

Свойство	**Значение**	**Комментарий**
`Caption`	`Конвертер`	
`Height`	`215`	
`Width`	`366`	
`BorderStyle`	`bsSingle`	Тонкая граница формы. Во время работы программы пользователь не сможет изменить размер окна путем захвата и перемещения его границы
`Position`	`poDesktopCenter`	Окно программы появится в центре рабочего стола
`BorderIcons.biMaximize`	`False`	В заголовке окна не отображать кнопку **Развернуть**
`Font.Name`	`Tahoma`	
`Font.Size`	`10`	

После того как будут установлены значения свойств формы, она должна выглядеть так, как показано на рис. 2.6. Теперь на форму надо добавить *компоненты*.

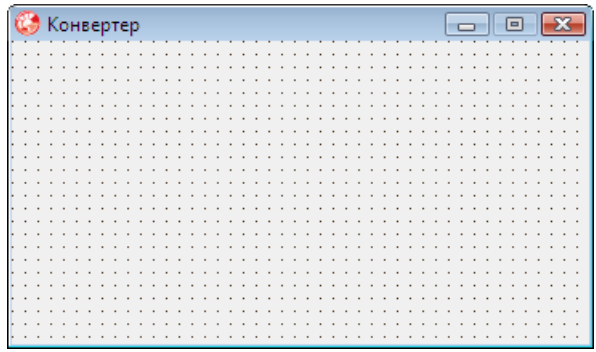


Рис. 2.6. Форма после изменения значений ее свойств

Компоненты

Поля редактирования, поля отображения текста, командные кнопки, списки, переключатели и другие элементы, обеспечивающие взаимодействие программы с пользователем, называют *компонентами пользовательского интерфейса*.

Компоненты, которые программист может использовать в процессе разработки программы, находятся в палитре компонентов (**Tool Palette**). На вкладках **Standard**, **Additional** и **Win32** располагаются часто используемые компоненты пользовательского интерфейса.

Программа "Конвертер" для пересчета цены из долларов в рубли должна получить от пользователя цену в долларах и курс. Для ввода данных с клавиатуры предназначен компонент `Edit`. Поэтому на форму разрабатываемого приложения нужно поместить два компонента `Edit`.

Чтобы на форму добавить компонент `Edit`, надо:

1. В палитре компонентов (окно **Tool Palette**) раскрыть вкладку **Standard**. Сделать щелчок на значке компонента `Edit` (рис. 2.7). Здесь следует обратить внимание, что в палитре компонентов, рядом со значком указывается тип компонента, а не его название.
2. Установить указатель мыши в ту точку формы, в которой должен быть левый верхний угол компонента.
3. Сделать щелчок левой кнопкой мыши.

В результате на форме появляется поле редактирования — компонент `Edit` (рис. 2.8).

Каждому добавленному компоненту среда разработки присваивает имя, которое состоит из названия компонента и его порядкового номера. Например, первый компонент `Edit` получает имя `Edit1`, второй — `Edit2`. Программист путем изменения значения свойства `Name` может поменять имя компонента. Однако в простых программах имена компонентов, как правило, не изменяют.

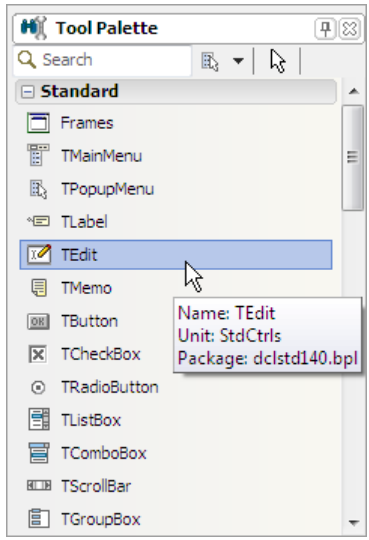


Рис. 2.7. Выбор компонента в палитре (компонент `Edit` — поле редактирования)

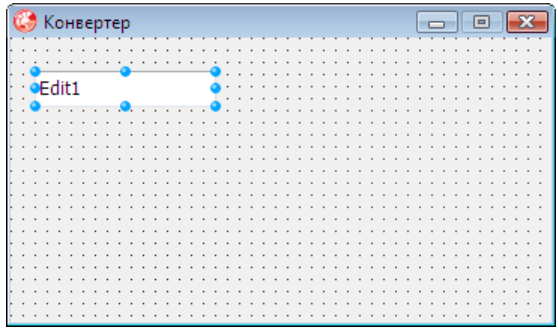


Рис. 2.8. Результат добавления на форму компонента `Edit`

Основные свойства компонента `Edit` приведены в табл. 2.3.

***Таблица 2.3.** Свойства компонента `Edit` (объект типа `TEdit`)*

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Text`	Текст, который находится 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 компонента

Таблица 2.3 *(окончание)*

Свойство	Описание
`Width`	Ширина компонента
`Font`	Шрифт, используемый для отображения текста в поле компонента
`ParentFont`	Признак наследования шрифта от формы. Если значение свойства равно `True`, то для отображения текста в поле компонента используется шрифт формы
`MaxLength`	Количество символов, которое можно ввести в поле редактирования. Если значение свойства равно нулю, ограничения на количество символов нет
`TabOrder`	Определяет порядок перемещения фокуса (курсора) с одного элемента управления на другой в результате нажатия клавиши <Tab>

На рис. 2.9 приведен вид формы после добавления двух полей редактирования. Один из компонентов *выбран* (выделен) — помечен маленькими кружками. Свойства выбранного компонента отображаются в окне **Object Inspector.** Чтобы увидеть и, если надо, изменить свойства другого компонента, нужно этот компонент выбрать — щелкнуть левой кнопкой мыши на изображении компонента или выбрать его имя в раскрывающемся списке, который находится в верхней части окна **Object Inspector** (рис. 2.10). Компонент, свойства которого надо изменить, можно выбрать и в окне **Structure** (рис. 2.11).

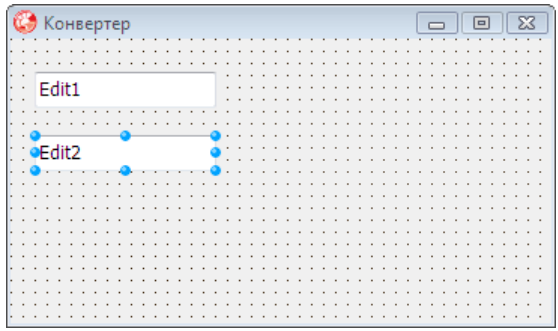


Рис. 2.9. Форма с двумя компонентами

Значения свойств, определяющих размер и положение компонента на поверхности формы, можно изменить с помощью мыши.

Чтобы изменить положение компонента, необходимо установить курсор мыши на его изображение, нажать левую кнопку мыши и, удерживая ее нажатой, переместить компонент в нужную точку формы. Во время перемещения компонента (рис. 2.12) отображаются текущие значения координат левого верхнего угла компонента (значения свойств `Left` и `Top`).

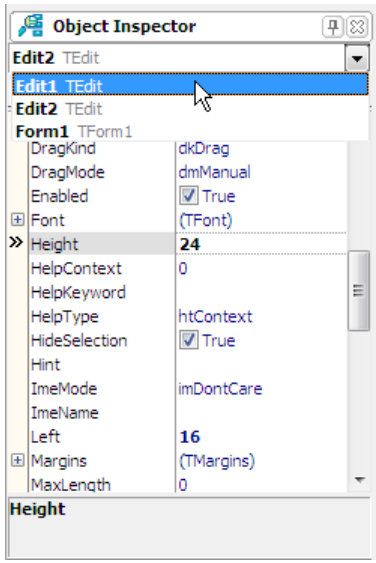


Рис. 2.10. Выбор компонента в окне **Object Inspector**

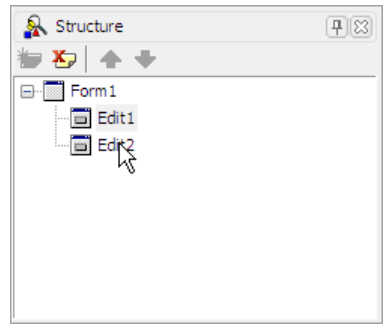


Рис. 2.11. Выбор компонента в окне **Structure**

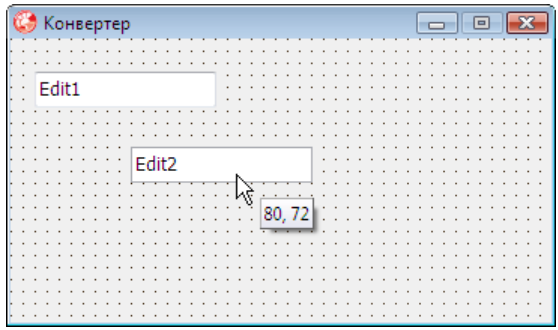


Рис. 2.12. Отображение значений свойств `Left` и `Top` при изменении положения компонента

Для того чтобы изменить размер компонента, необходимо сделать щелчок на его изображении (в результате чего компонент будет выделен), установить указатель мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить положение границы компонента. Во время изменения размера компонента отображаются его текущие размеры: ширина и высота (значения свойств `Width` и `Height`) (рис. 2.13).

В табл. 2.4 приведены значения свойств компонентов `Edit1` и `Edit2` (прочерк показывает, что значением свойства `Text` является пустая строка). Значения остальных свойств компонентов `Edit` оставлены без изменения, и поэтому в таблице не показаны. Компонент `Edit1` предназначен для ввода курса, `Edit2` — цены. Так как значения свойства `Font` компонентов `Edit` не указаны явно, то во время работы программы текст в полях редактирования будет отображаться шрифтом, заданным

для формы. Компонент Edit, как и другие компоненты, наследует значение свойства Font от своего родителя — объекта, на поверхности которого он находится. Поэтому если изменить значение свойства Font формы, автоматически изменится значение свойства Font компонентов, находящихся на форме. Если требуется, чтобы текст в поле компонента отображался другим шрифтом, нужно явно задать значение свойства Font этого компонента. Чтобы запретить автоматическое изменение значения свойства Font компонента при изменении свойства Font формы, надо свойству ParentFont компонента присвоить значение False.

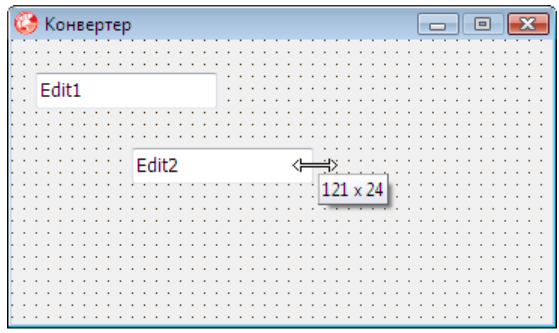


Рис. 2.13. Отображение значений свойств Width и Height при изменении размера компонента

Форма программы "Конвертер" после настройки компонентов Edit приведена на рис. 2.14.

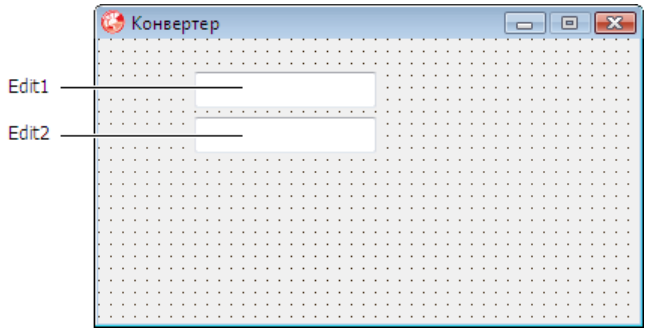


Рис. 2.14. Форма после настройки компонентов Edit

Таблица 2.4. *Значения свойств компонентов* Edit

Компонент	Свойство	Значение
Edit1	Top	64
	Left	22
	Text	-
	TabOder	0

Таблица 2.4 *(окончание)*

Компонент	Свойство	Значение
Edit2	Top	52
	Left	22
	Text	-
	TabOder	1

Отображение текста на поверхности формы обеспечивает компонент `Label`. В рассматриваемой программе текст отображается слева от полей редактирования (информация о назначении полей ввода). Результат расчета также отображается в окне программы. Поэтому в форму надо добавить три компонента `Label` (рис. 2.15).

Добавляются компоненты `Label` на форму точно так же, как и поля редактирования (компонент `Edit`).

Основные свойства компонента `Label` перечислены в табл. 2.5.

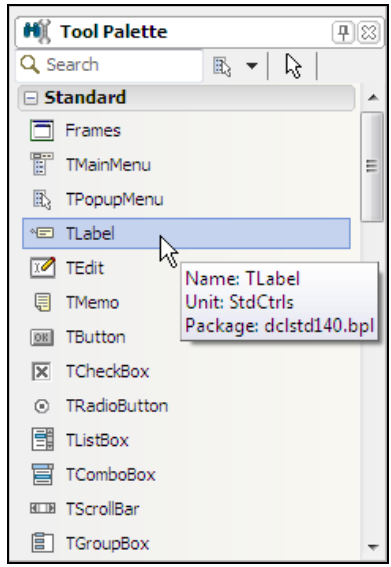


Рис. 2.15. Компонент `Label` — поле отображения текста

Таблица 2.5. *Свойства компонента* `Label`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту
`Caption`	Отображаемый текст

Таблица 2.5 *(окончание)*

Свойство	Описание
Font	Шрифт, используемый для отображения текста
ParentFont	Признак наследования характеристик шрифта от объекта (формы), на котором компонент находится
AutoSize	Признак автоматического изменения размера компонента при изменении текста, отображаемого в поле компонента
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
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства AutoSize должно быть False)

Если поле компонента Label должно содержать несколько строк текста, то перед тем как изменить значение свойства Caption, сначала надо присвоить свойству AutoSize значение False, а свойству WordWrap — True. Затем нужно установить требуемый размер компонента (с помощью мыши или вводом значений свойств Width и Height) и только после этого ввести значение свойства Caption.

На форму разрабатываемого приложения надо добавить три компонента Label. В полях Label1 и Label2 отображается информация о назначении полей ввода, поле Label3 используется для вывода результата расчета. Значения свойств компонентов Label приведены в табл. 2.6.

Таблица 2.6. *Значения свойств компонентов* Label

Компонент	Свойство	Значение
Label1	Left	20
	Top	30
	Caption	Курс
Label2	Left	20
	Top	60
	Caption	Цена

Таблица 2.6 *(окончание)*

Компонент	Свойство	Значение
Label3	Left	16
	Top	152
	AutoSize	False
	Width	297
	Height	24
	Caption	-

После настройки компонентов `Label` форма разрабатываемого приложения должна выглядеть так, как показано на рис. 2.16.

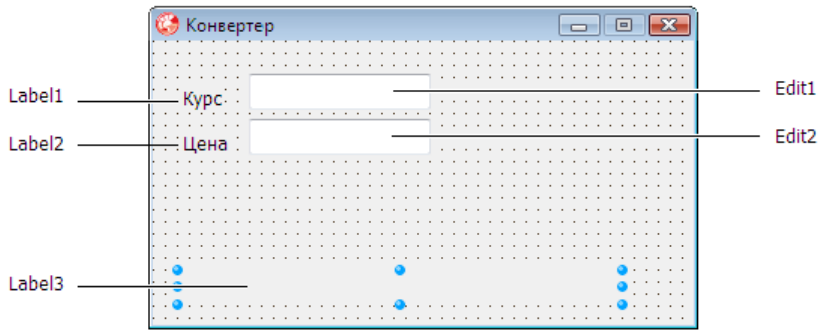


Рис. 2.16. Вид формы после настройки полей отображения текста

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

Командная кнопка, компонент `Button`, добавляется на форму точно так же, как и другие компоненты. Значок компонента `Button` находится на вкладке **Standard** (рис. 2.17). Основные свойства компонента `Button` приведены в табл. 2.7.

Таблица 2.7. Свойства компонента `Button`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Enabled`	Признак доступности кнопки. Кнопка доступна (программа реагирует на ее нажатие), если значение свойства равно `True`, и не доступна, если значение свойства равно `False`

Таблица 2.7 *(окончание)*

Свойство	Описание
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Height`	Высота кнопки
`Width`	Ширина кнопки
TabOrder	Определяет порядок перемещения фокуса (курсора) с одного элемента управления на другой в результате нажатия клавиши <Tab>

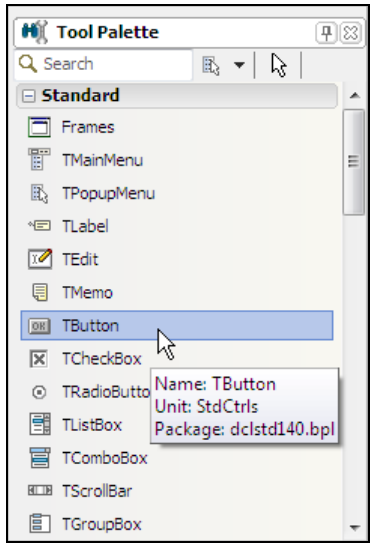


Рис. 2.17. Командная кнопка — компонент `Button`

После того как на форму будут добавлены кнопки, нужно выполнить их настройку. Значения свойств компонентов `Button` приведены в табл. 2.8, окончательный вид формы показан на рис. 2.18.

Таблица 2.8. *Значения свойств компонентов* `Button`

Свойство	Компонент	
	Button1	**Button2**
`Left`	16	110
`Top`	136	136
`Width`	75	75

Таблица 2.8 *(окончание)*

Свойство	Компонент	
	Button1	**Button2**
Height	25	25
Caption	Пересчет	Завершить

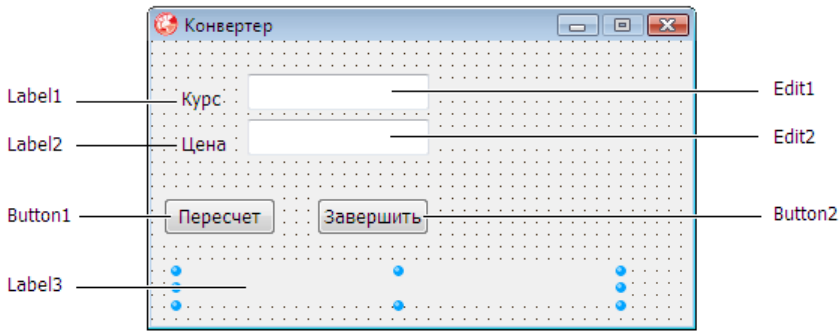


Рис. 2.18. Окончательный вид формы программы "Конвертер"

Завершив работу по созданию формы, можно приступить к программированию — созданию процедур обработки событий.

Событие

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

Событие (event) — это то, что происходит во время работы программы. У каждого события есть имя. Например, щелчок кнопкой мыши — это событие `Click`, двойной щелчок мышью — событие `DblClick`.

В табл. 2.9 приведены некоторые события, возникающие в результате действий пользователя.

Таблица 2.9. *События*

Событие	Описание
`Click`	Щелчок кнопкой мыши
`DblClick`	Двойной щелчек кнопкой мыши

Таблица 2.9 *(окончание)*

Событие	Описание
MouseDown	Нажатие кнопки мыши
MouseUp	Отпускание нажатой кнопки мыши
MouseMove	Перемещение указателя мыши
KeyPress	Нажатие клавиши клавиатуры
KeyDown	Нажатие 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	Отпускание нажатой клавиши клавиатуры
Create	Создание объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
Paint	Событие происходит при появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном
Enter	Получение элементом управления фокуса
Exit	Потеря элементом управления фокуса

Следует понимать, что одни и те же действия, но выполненные над различными объектами, вызывают разные события. Например, щелчок (событие Click) на кнопке **Пересчет** и щелчок на кнопке **Завершить** — это два разных события.

Процедура обработки события

Реакцией на событие должно быть какое-либо действие. В Delphi 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.

Методику создания процедуры обработки события рассмотрим на примере обработки события Click, которое возникает в результате щелчка на кнопке **Пересчет**.

Чтобы приступить к созданию процедуры обработки события, сначала надо выбрать компонент, для которого создается процедура обработки события. Для этого в окне конструктора формы надо сделать щелчок левой кнопкой мыши на нужном компоненте (компонент можно выбрать также в раскрывающемся списке, который находится в верхней части окна **Object Inspector**). Затем в окне **Object Inspector** нужно открыть вкладку **Events**.

В левой колонке вкладки **Events** (рис. 2.19) перечислены события, которые может воспринимать выбранный компонент. Строго говоря, на вкладке **Events** указаны не события, а свойства, значением которых являются имена процедур обработки соответствующих событий. Так, например, значением свойства `OnClick` является имя процедуры обработки события `Click`.

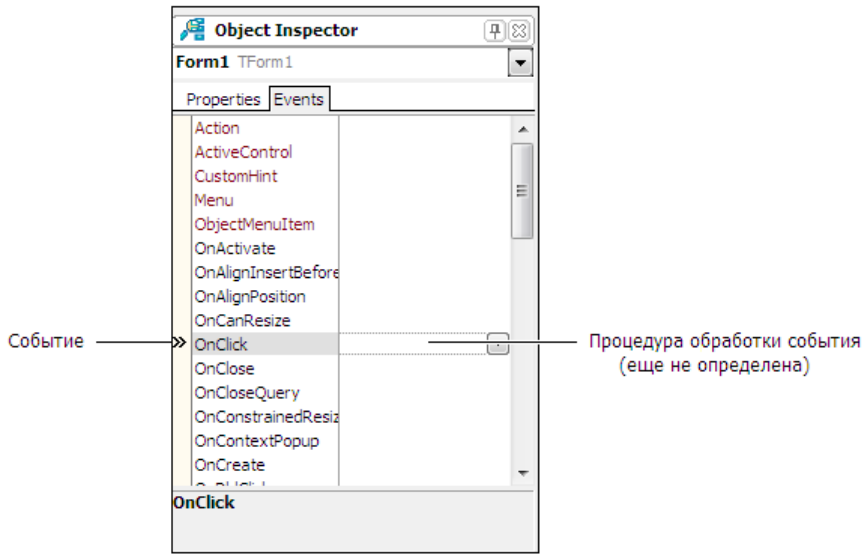


Рис. 2.19. На вкладке **Events** перечислены события, которые может воспринимать компонент

Для того чтобы создать процедуру обработки события, нужно на вкладке **Evens** выбрать событие (сделать щелчок мышью на его имени), ввести имя процедуры обработки события в ставшее доступным поле редактирования (рис. 2.20) и нажать клавишу <Enter>.

В результате этих действий в текст программы (в модуль формы) будет добавлена процедура обработки события и станет доступным окно редактора кода (рис. 2.21), в котором можно набирать инструкции процедуры обработки события. Следует обратить внимание на то, что формируемое средой разработки полное имя процедуры обработки события состоит из двух частей. Первая часть идентифицирует форму, вторая представляет собой непосредственно имя процедуры. Согласно принятому соглашению имя процедуры обработки должно содержать информацию о компоненте, реагирующем на событие, и событии. Например, `Button1Click` — это имя процедуры обработки события `Click` на компоненте `Button1`.

Существует и другой способ создания процедуры обработки события. Сначала на вкладке **Events** надо выбрать событие, процедуру обработки которого необходимо создать, затем сделать двойной щелчок левой кнопкой мыши в поле редактирования, которое находится справа от имени события. В результате имя процедуры обработки события сформирует Delphi (имя процедуры обработки события образу-

ется путем объединения имени компонента, для которого создается процедура обработки события, и имени события, например `Button1Click`).

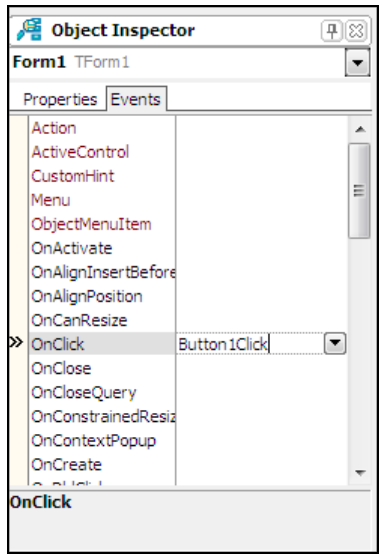


Рис. 2.20. Рядом с именем события надо ввести имя процедуры обработки события

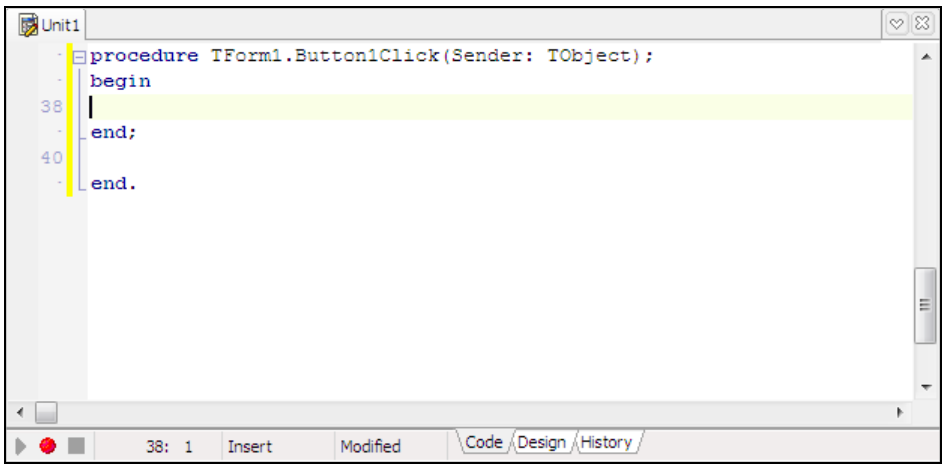


Рис. 2.21. Шаблон процедуры обработки события

В листинге 2.1 приведен текст процедуры обработки события `Click` на кнопке **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 правилами хорошего стиля программирования.

Листинг 2.1. Обработка события `Click` на кнопке *Пересчет*

```
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real;   // цена в долларах
  k:   real;   // курс
  rub: real;   // цена в рублях

begin
  // получить данные из полей редактирования
  k := StrToFloat(Edit1.Text);
  usd := StrToFloat(Edit2.Text);

  // пересчитать цену из долларов в рубли
  rub := usd * k;

  // вывести результат расчета в поле Label3
  Label3.Caption := FloatToStrF(usd,ffGeneral, 6,2) + ' $ = ' +
                    FloatToStrF(rub,ffCurrency,6,2);
end;
```

Процедура `TForm1.Button1Click` пересчитывает цену из долларов в рубли и выводит результат в поле `Label3`. Исходные данные (курс и цена в долларах) вводятся из полей редактирования `Edit1` и `Edit2` путем обращения к свойству `Text`. Значением свойства `Text` является строка, которая находится в поле редактирования. Свойство `Text` строкового типа, поэтому для преобразования строки в число используется функция `StrToFloat`. Следует обратить внимание, что функция `StrToFloat` возвращает результат только в том случае, если строка, переданная функции в качестве параметра, действительно является изображением числа в правильном формате, что предполагает использование запятой (при стандартной для России настройке операционной системы) в качестве разделителя целой и дробной частей числа. Вычисленное значение цены в долларах выводится в поле `Label3` путем присваивания значения свойству `Caption`. Для преобразования числа в строку (свойство `Caption` строкового типа) используется функция `FloatToStrF`. У функции `FloatToStr` четыре параметра: первый — значение, которое надо преобразовать в строку; второй — формат (`ffGeneral` — обычное число, `ffCurrency` — финансовый); третий и четвертый задают соответственно общее количество цифр и количество цифр дробной части.

Замечание

Для преобразования строки в целое число и целого числа в строку используйте соответственно методы-функции `StrToInt` и `IntToStr`.

В листинге 2.2 приведена процедура обработки события `Click` на кнопке **Завершить** (создается она точно так же, как и процедура обработки события `Click` для кнопки **Пересчет**). В результате щелчка на кнопке **Завершить** программа должна завершить работу. Чтобы это произошло, надо закрыть окно программы (форму). Делает это метод `Close`.

Листинг 2.2. Обработка события `Click` на кнопке *Завершить*

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Form1.Close; // закрыть окно
end;
```

Редактор кода

В процессе набора текста программы редактор кода автоматически выделяет элементы программы: полужирным — ключевые слова языка программирования, цветом — константы, курсивом — комментарии. Это делает текст программы выразительным, облегчает восприятие ее структуры. Кроме того, редактор "на лету" проверяет программу на наличие синтаксических ошибок и в случае обнаружения ошибки подчеркивает ее красной волнистой линией (сообщение об обнаруженной ошибке отображается в окне **Structure**).

В процессе разработки программы часто возникает необходимость переключения между окном редактора кода и окном конструктора формы. Выбрать нужное окно можно щелчком на ярлыке **Code** (редактор кода) или **Design** (редактор формы). Для переключения между этими окнами удобно использовать клавишу <F12>.

Система подсказок

Во время набора текста программы редактор кода автоматически выводит подсказки. Например, после набора имени функции редактор кода выводит список ее параметров. Параметр, который в данный момент вводит программист, в подсказке выделяется полужирным шрифтом. Например, если набрать слово `StrToFloat`, которое является именем функции преобразования строки в дробное число, и открывающую скобку, то на экране появится окно, в котором будут перечислены параметры функции (рис. 2.22).

Редактор также выводит список свойств и методов текущего объекта и позволяет выбрать в нем нужное свойство или метод. Например, если в окне редактора кода набрать `Edit1` (имя компонента, поля редактирования) и точку, то на экране появится список свойств и методов класса `TEdit` (рис. 2.23). Программисту остается только выбрать в списке нужное свойство или метод (быстро перейти к нужному

элементу списка можно, нажав клавишу, соответствующую первому символу этого элемента), и нажать клавишу <Enter>.

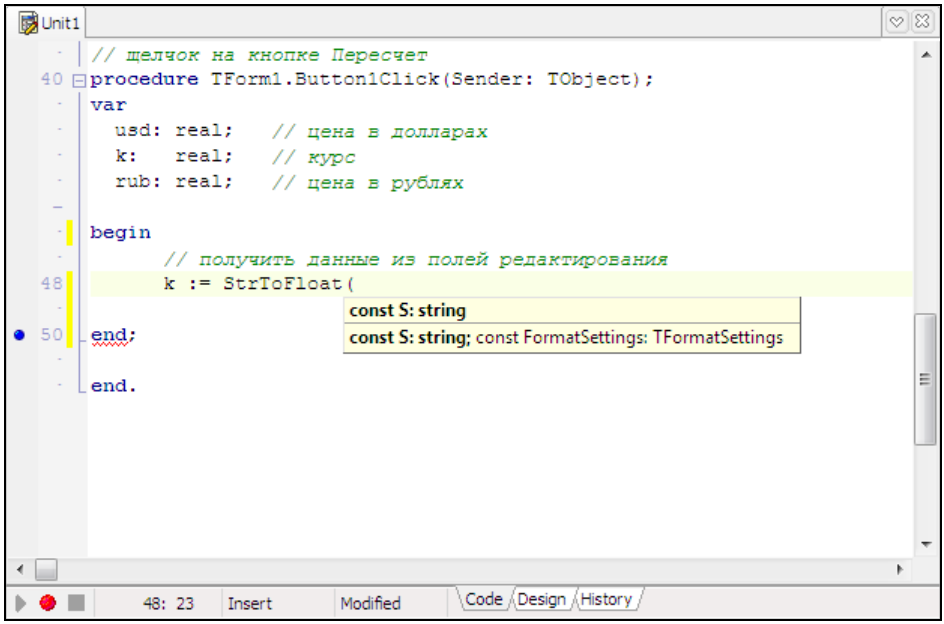


Рис. 2.22. Пример подсказки — список параметров функции

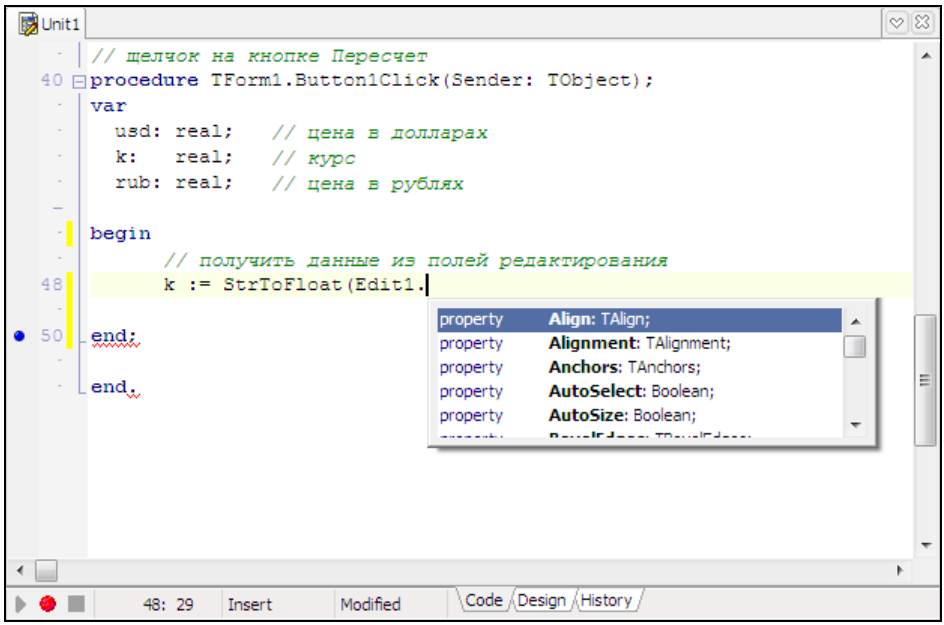


Рис. 2.23. Редактор кода автоматически выводит список свойств и методов текущего объекта

Шаблоны кода

В процессе набора текста программы можно использовать шаблоны кода (Code Templates). Шаблон кода — это инструкция программы, записанная в общем виде. Например, простейший шаблон инструкции `if` выглядит так:

```
if    then
```

Редактор кода предоставляет программисту большой набор шаблонов: объявления классов, функций, инструкций выбора (`if`, `switch`), циклов (`for`, `while`). Для некоторых инструкций, например `if` и `while`, есть несколько вариантов шаблонов.

Часто используемые шаблоны кода редактор вставляет в текст программы автоматически, как только программист наберет ключевое слово. Например, если в окне редактора кода набрать `if` и пробел, то в текст будет вставлен шаблон `if then`, причем курсор останется после `if` (после ввода условия, чтобы установить курсор после `then`, надо нажать клавишу <Enter> или <Tab>).

Все доступные шаблоны кода отображаются в окне **Templates** (рис. 2.24). Чтобы вставить шаблон в текст программы, надо сделать щелчок на имени шаблона.

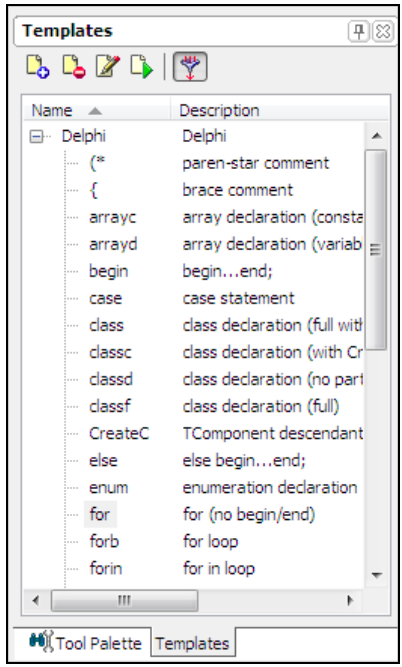


Рис. 2.24. В окне **Templates** отображаются шаблоны кода

Обратите внимание, что по умолчанию окно шаблонов не отображается, и чтобы оно появилось на экране, надо в меню **View** выбрать команду **Templates**.

Доступ к списку шаблонов (рис. 2.25) можно получить, нажав комбинацию клавиш <Ctrl>+<J>. Выбрать шаблон можно обычным образом (прокручивая список) или ввести его имя (например, шаблон `if then` называется `if`, а шаблон `if`

`then begin end` — `ifb`). Выбрав в списке шаблон, надо нажать клавишу <Enter>, шаблон будет вставлен в текст программы.

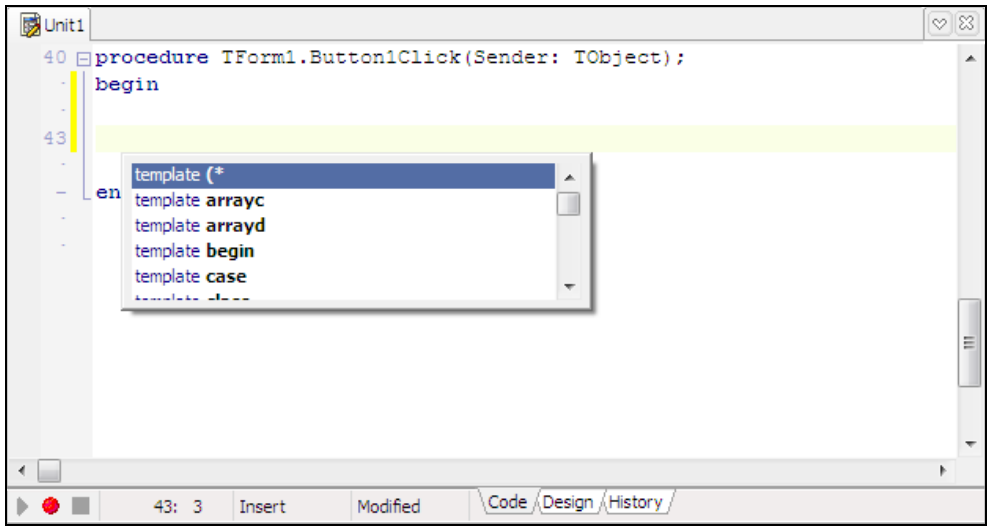


Рис. 2.25. Чтобы получить доступ к списку шаблонов, нажмите клавиши <Ctrl>+<J>

Справочная информация

Во время набора программы можно получить справку о конструкции языка, типе данных, процедуре или функции. Быстро получить доступ к нужному разделу справочной информации можно, набрав в окне редактора кода ключевое слово и нажав клавишу <F1>. Например, чтобы получить справку о функции `FloatToStrF`, надо в окне редактора кода набрать ее имя и нажать клавишу <F1>.

Иногда описанный способ доступа к справочной информации не дает результата (например, если неправильно указано имя функции). В этом случае надо в меню **Help** выбрать команду **Delphi Help**, затем в появившемся окне отображения справочной информации выбрать вкладку **Index** и в поле **Look for** ввести ключевое слово (или несколько первых букв слова). В результате будет сформирован список разделов, связанных с введенным словом.

Сохранение проекта

Проект — это набор файлов, используя которые, компилятор создает exe-файл. В простейшем случае проект образуют: файл описания проекта (dproj-файл), главный модуль (dpr-файл), файл описания формы (dfm-файл), модуль формы (pas-файл) и файл ресурсов (res-файл).

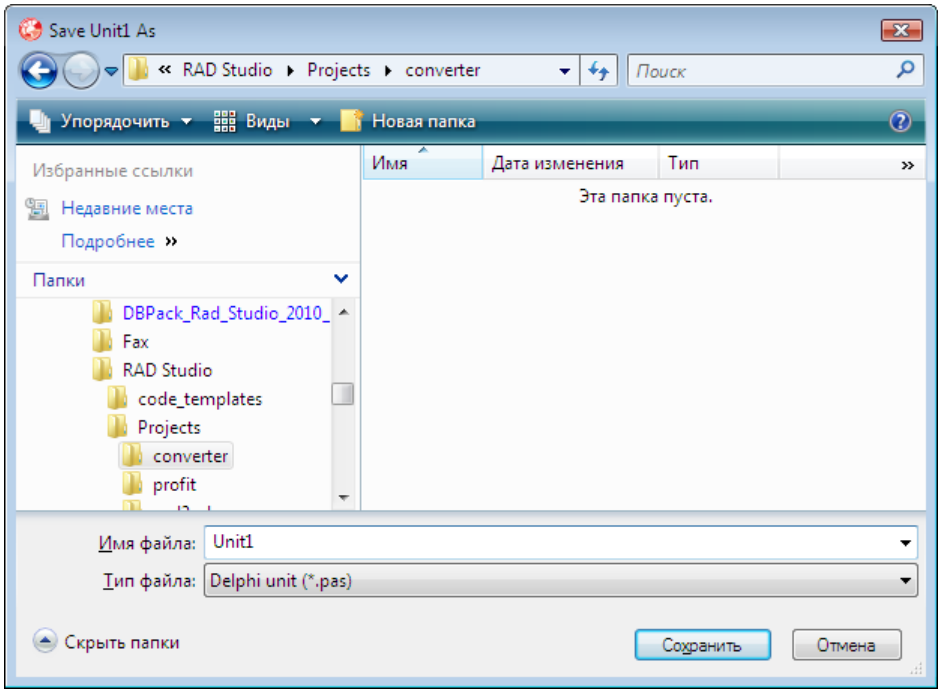


Рис. 2.26. Сохранение модуля формы

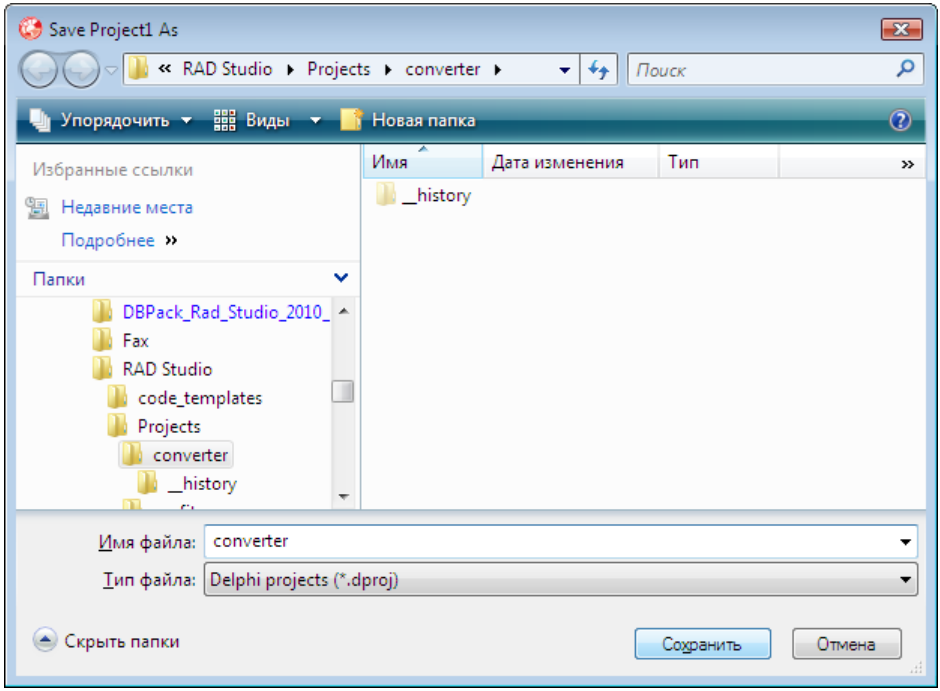


Рис. 2.27. Сохранение проекта

Чтобы сохранить проект, нужно в меню **File** выбрать команду **Save Project As**. Если проект еще ни разу не был сохранен, то сначала на экране появляется окно **Save Unit As**. В этом окне (рис. 2.26) надо выбрать папку, предназначенную для проектов Delphi (по умолчанию это C:\Users*UserName*\Documents\RAD Studio\Projects, где *UserName* — имя пользователя в системе), создать в ней папку для сохраняемого проекта, открыть созданную папку и в поле **Имя файла** ввести имя модуля (обычно в качестве имени модуля указывают имя формы, например `MainForm`) и нажать кнопку **Сохранить**. Затем в появившемся окне **Save Project As** (рис. 2.27) надо ввести имя проекта. Обратите внимание на то, что имя проекта определяет имя exe-файла, который будет создан компилятором.

Структура проекта

Как уже говорилось, проект представляет собой совокупность файлов, которые используются компилятором для генерации выполняемого файла. Основу проекта образуют файл главного модуля (dpr-файл) и один или несколько модулей (для каждой формы Delphi создает отдельный модуль — pas-файл). Помимо файла проекта и модулей в процессе компиляции используются: файл ресурсов (res-файл), файлы описания форм (отдельный dfm-файл для каждой формы), а также файл конфигурации (cfg-файл). Общая информация о проекте хранится в dproj-файле.

Файл главного модуля (чтобы его увидеть, надо в меню **Project** выбрать команду **View Source**) содержит инструкции, обеспечивающие инициализацию приложения.

В качестве примера в листинге 2.3 приведен главный модуль приложения "Конвертер".

Листинг 2.3. Главный модуль приложения "Конвертер" (converter.dpr)

```
program converter;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Начинается файл проекта директивой `program`, в которой указывается имя приложения. Далее, за директивой `uses`, следуют имена модулей, необходимых для создания выполняемого файла: библиотечного модуля `Forms` и модуля `unit1` — стартовой формы.

Строка `{$R *.res}` — это директива компилятору подключить файл ресурсов. В файле ресурсов находится значок приложения. Звездочка показывает, что имя файла ресурсов такое же, как и у файла проекта. Инструкции между `begin` и `end` обеспечивают инициализацию приложения (создание объекта `Application`), создание стартовой формы (объекта `Form1`) и запуск приложения (метод `Run`).

Модули содержат объявления типов, констант, переменных, процедур и функций. В качестве примера в листинге 2.4 приведен модуль стартовой формы программы "Конвертер". Этот модуль содержит объявление класса `TForm1`. Обратите внимание, что инструкция `Application.CreateForm(TForm1, Form1)`, которая находится в файле проекта (см. листинг 2.3), создает объект класса `TForm1`, т. е. стартовую форму приложения "Конвертер".

Листинг 2.4. Модуль стартовой формы программы "Конвертер" (unit1.pas)

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Button1: TButton;
    Button2: TButton;
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real; // цена в долларах
  k: real;   // курс
  rub: real; // цена в рублях
begin
  // получить данные из полей редактирования
  k := StrToFloat(Edit1.Text);
  usd := StrToFloat(Edit2.Text);

  // пересчитать цену из долларов в рубли
  rub := usd * k;

  // вывести результат расчета в поле Label3
  Label3.Caption := FloatToStr(usd) + ' $ = ' +
                    FloatToStr(rub) + ' руб.';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form1.Close; // закрыть окно
end;

end.
```

Начинается модуль словом `unit`, за которым указано имя модуля. Именно это имя упоминается в списке используемых модулей в инструкции `uses` файла проекта.

Модуль состоит из следующих разделов: интерфейса, реализации и инициализации.

В разделе интерфейса (начинается словом `interface`) находятся объявления класса `TForm1` и объекта `Form1`. Таким образом, модуль, который использует модуль `unit1`, может создать форму `Form1` (окно **Конвертер**).

Раздел реализации открывается словом `implementation`. Директива `{$R *.dfm}` указывает компилятору, что описание формы, сформированное Delphi в процессе

ее создания, находится в файле unit1.dfm. Далее следует описание методов класса `TForm1` (процедуры `Button1Click` и `Button2Click` являются методами).

Следует отметить, что модуль проекта (dpr-файл), файл описания формы (dfm-файл), файл ресурсов (res-файл), а также значительное количество инструкций модуля формы (pas-файл) формирует Delphi.

Компиляция

Процесс преобразования исходной программы в выполняемую называется *компиляцией* и состоит из двух этапов: непосредственно компиляции и компоновки. На этапе компиляции выполняется перевод исходной программы (модулей) в некоторое внутреннее представление. На этапе компоновки выполняется сборка (построение) программы.

Процесс компиляции активизируется в результате выбора в меню **Project** команды **Compile**, а также путем запуска программы из среды разработки (команда **Run** или **Run Without Debugging** меню **Run**), если с момента последней компиляции в программу были внесены изменения. Следует обратить внимание, что перед тем как первый раз выполнить компиляцию новой программы, необходимо сохранить проект.

Процесс и результат компиляции отражаются в окне **Compile**. Если в программе синтаксических ошибок нет, то по завершении компиляции в поле **Compiling** отображается слово **Done** (рис. 2.28).

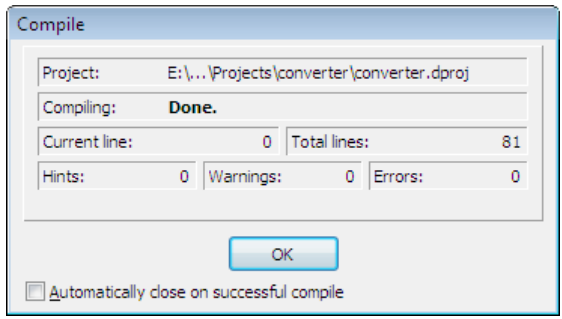


Рис. 2.28. Результат компиляции (в программе ошибок нет)

Если в программе есть ошибки и (или) неточности, то в поле **Compiling** выводится сообщение **There are errors** (рис. 2.29) и отображается информация о количестве синтаксических (поле **Errors**) и семантических (поле **Warnings**) ошибок, а также о количестве неточностей (поле **Hints**). Сами же сообщения об ошибках, предупреждения и подсказки отображаются в окне **Messages** (рис. 2.30). Кроме того, после того как окно **Compile** будет закрыто, в редакторе кода будет выделена строка, в которой находится первая (от начала файла) обнаруженная ошибка.

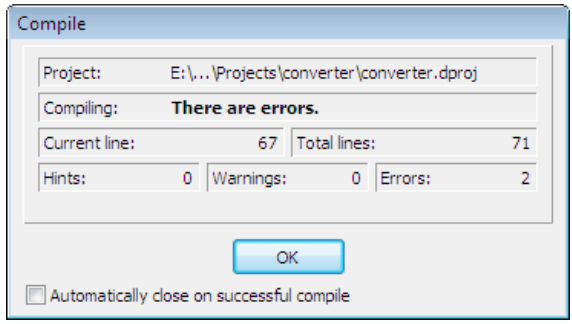


Рис. 2.29. Результат компиляции (в программе есть ошибки)

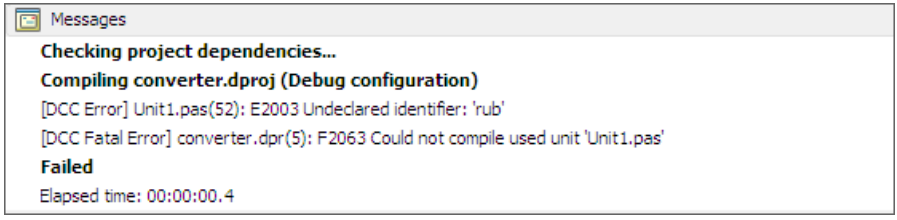


Рис. 2.30. Сообщения об обнаруженных ошибках

Ошибки

Компилятор генерирует выполняемую программу (exe-файл) только в том случае, если в исходной программе нет синтаксических ошибок. Если в программе есть ошибки, то программист должен их устранить. Процесс устранения ошибок носит итерационный характер. Обычно сначала устраняются наиболее очевидные ошибки, например объявляются необъявленные переменные. После очередного внесения изменений в текст программы выполняется повторная компиляция.

В табл. 2.10 приведены сообщения компилятора о типичных ошибках.

***Таблица 2.10.** Сообщения компилятора об ошибках*

Сообщение	Вероятная причина
`Undeclared identifier` (Необъявленный идентификатор)	Используется переменная, не объявленная в разделе `var`. Ошибка при записи имени переменной. Например, в программе объявлена переменная `Sum`, а в тексте программы написано: `Suma`
`Unterminated string` (Незавершенная строка)	При записи строковой константы, например сообщения, не поставлена завершающая кавычка

Таблица 2.10 *(окончание)*

Сообщение	Вероятная причина
`Incompatible types` (Несовместимые типы)	В инструкции присваивания тип выражения не соответствует или не может быть приведен к типу переменной, получающей значение выражения. Тип фактического параметра процедуры или функции не соответствует или не может быть приведен к типу формального параметра
`Missing operator or semicolon` (Отсутствует оператор или точка с запятой)	После инструкции не поставлена точка с запятой
`Could not create output file` (Невозможно создать exe-файл)	Программа, над которой идет работа, запущена (командой **Run ▸ Run Without Debugging**), поэтому существующий exe-файл нельзя заменить новым

Следует обратить внимание, что компилятор не всегда может точно локализовать ошибку. Поэтому, анализируя фрагмент кода, который помечен компилятором как содержащий ошибку, надо обращать внимание и на текст, который находится в предыдущих строках.

Предупреждения и подсказки

При обнаружении в программе неточностей, которые не являются ошибками, компилятор выводит подсказки (Hints) и предупреждения (Warnings).

Например, часто выводимой подсказкой является сообщение об объявленной, но не используемой переменной:

```
Variable ... is declared but never used in ...
```

Действительно, зачем объявлять переменную и не использовать ее?

В табл. 2.11 приведены предупреждения и подсказки компилятора о типичных неточностях в программе.

Таблица 2.11. *Предупреждения и подсказки компилятора*

Сообщение	Причина
`Variable... is declared but never used in ...`	Переменная объявлена, но не используется
`Variable ... might not have been initialized.` (Вероятно, используется неинициализированная переменная)	В программе нет инструкции, которая присваивает переменной начальное значение

Запуск программы

Пробный запуск программы можно выполнить непосредственно из Delphi, не завершая работу со средой разработки. Для этого в меню **Run** надо выбрать команду **Run** или **Run Without Debugging**. Можно также сделать щелчок на кнопке **Run** (рис. 2.31) или нажать клавишу <F9>.

Рис. 2.31. Чтобы запустить программу, сделайте щелчок на кнопке **Run**

Команда **Run** запускает программу в режиме отладки. Команда **Run Without Debugging** запускает программу в обычном режиме, даже в том случае, если в ней есть информация, необходимая для отладки (заданы точки останова, указаны переменные, значения которых надо контролировать). Следует обратить внимание, что процесс запуска программы командой **Run Without Debugging** происходит быстрее.

Исключения

Ошибки, возникающие во время работы программы, называют *исключениями*. В большинстве случаев причиной исключений являются неверные данные. Например, если в поле **Курс** программы "Конвертер" ввести строку `30.09` и сделать щелчок на кнопке **Пересчет**, то на экране появится сообщение (рис. 2.32): `'30.09' is not valid floating point value` (30.09 не является дробным числом).

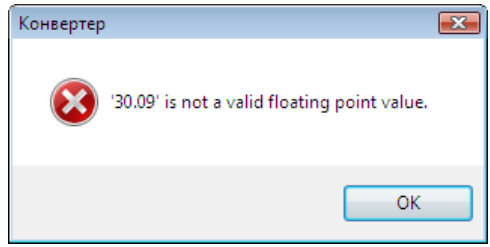


Рис. 2.32. Пример сообщения о возникновении ошибки исключения

Причина возникновения исключения описанной ошибки в следующем. Преобразование строки, введенной в поле редактирования, в число выполняет функция `StrToFloat`. Эта функция работает правильно, если ее параметром является строковое представление дробного числа, что при стандартной для России настройке

Windows предполагает использование в качестве десятичного разделителя запятой. В рассматриваемом примере строка `30.09` не является строковым представлением дробного числа, поэтому и возникает исключение.

Если программа запущена из среды разработки в режиме отладки (команда **Run** из меню **Run**), то при возникновении исключения выполнение программы приостанавливается и на экране появляется окно **Debugger Exception Notification**, в котором, помимо сообщения об ошибке, указывается тип исключения (рис. 2.33). Щелчок на кнопке **Break** приостанавливает выполнение программы в реальном времени и переводит ее в режим выполнения по шагам (при этом в окне редактора кода выделяется инструкция, при выполнении которой произошла ошибка). Щелчок на кнопке **Continue** запускает программу с той точки, в которой была приостановлена ее работа.

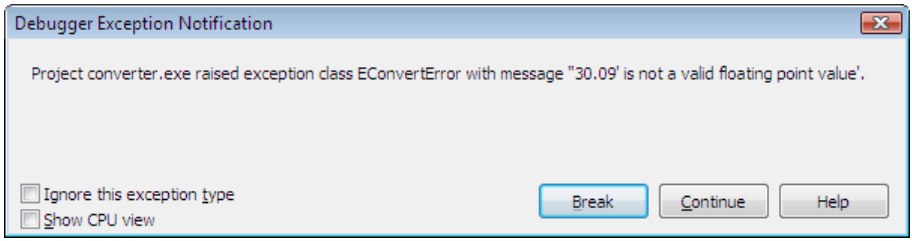


Рис. 2.33. Пример сообщения о возникновении исключения

Для того чтобы остановить программу, во время работы которой возникло исключение, надо в меню **Run** выбрать команду **Program Reset**.

Обработка исключения

Обработку исключений берет на себя автоматически добавляемый в выполняемую программу код, который обеспечивает вывод сообщения об ошибке и завершение процедуры, при выполнении которой возникло исключение. Вместе с тем программист может поместить в программу код, который выполнит обработку исключения.

Инструкция обработки исключения в общем виде выглядит так:

```
try
  // здесь инструкции, при выполнении которых может
  // произойти исключение
except
  on Исключение do
    begin
      // здесь инструкции обработки исключения
    end;
end;
```

Ключевое слово `try` указывает, что далее следуют инструкции, при выполнении которых возможно возникновение исключений, и что обработку этих исключений берет на себя программа. Слово `except` обозначает начало секции обработки исключений. После слова `on` указывается исключение, обработку которого берет на себя программа, а после `do` — инструкции, обеспечивающие обработку исключения. Нужно обратить внимание, что инструкции, следующие за той, при выполнении которой возникло исключение, после обработки исключения не выполняются.

В табл. 2.12 перечислены часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 2.12. *Типичные исключения*

Тип исключения	Возникает
`EConvertError`	При выполнении преобразования строки в число, если преобразуемая величина не может быть приведена к требуемому виду. Чаще всего возникает при преобразовании строки в дробное число, если в качестве разделителя целой и дробной частей указан неверный символ
`EZeroDivide`	Деление на ноль. При выполнении операции деления, если делитель равен нулю (если и делитель, и делимое равны нулю, то возникает исключение `EInvalidOp`)
`EFOpenError`	При обращении к файлу, например при попытке загрузить файл иллюстрации с помощью метода `LoadFromFile`. Наиболее частой причиной является отсутствие требуемого файла или, в случае использования сменного диска, отсутствие диска в накопителе
`EInOutError`	При обращении к файлу, например при попытке открыть для чтения несуществующий файл
`EOLEException`	При выполнении операций с базой данных, например при попытке открыть несуществующую базу данных, если для доступа к базе данных используются ADO-компоненты (чтобы иметь возможность обработки этого исключения, в директиву `uses` надо добавить ссылку на модуль `ComObj`)

В качестве примера использования инструкции `try` в листинге 2.5 приведена процедура обработки события `Click` на кнопке **Пересчет** окна программы "Конвертер". При возникновении исключения `EConvertError` программа определяет причину (незаполненное поле или неверный формат данных) и выводит соответствующее сообщение (рис. 2.34).

Листинг 2.5. Щелчок на кнопке *Пересчет* (с обработкой исключения)

```
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real;   // цена в долларах
```

```
  k:   real;   // курс
  rub: real;   // цена в рублях

begin
  try
    // получить данные из полей редактирования
    k := StrToFloat(Edit1.Text);
    usd := StrToFloat(Edit2.Text);

    // пересчитать цену из долларов в рубли
    rub := usd * k;
    // вывести результат расчета в поле Label4
    Label3.Caption := FloatToStr(usd) + '$ = ' +
                      FloatToStr(rub) + 'руб.';
  except
    if (Length(Edit1.Text) = 0) or (Length(Edit2.Text)=0) then
        MessageDlg('Надо ввести данные в оба поля',
                   mtWarning,[mbOk],0)
    else
        MessageDlg('При вводе дробных чисел используйте запятую',
                   mtError,[mbOk],0);
  end;
end;
```

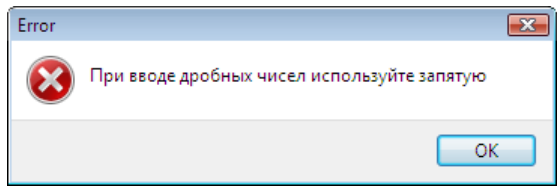


Рис. 2.34. Пример сообщения об ошибке (диалог `MessageDlg`)

В приведенной процедуре обработки события `Click` для вывода сообщения о неверных данных используется функция `MessageDld`, инструкция вызова которой в общем виде выглядит так:

```
r := MessageDlg(Сообщение, Тип, Кнопки, РазделСправки)
```

где:

- *Сообщение* — текст сообщения;
- *Тип* — тип сообщения. Сообщение может быть информационным (`mtInformation`), предупреждающим (`mtWarning`) или сообщением об ошибке (`mtError`). Каждому типу сообщения соответствует значок (табл. 2.13);
- *Кнопки* — список кнопок, отображаемых в окне сообщения (табл. 2.14);

- *РазделСправки* — идентификатор раздела справочной информации, который появится на экране, если пользователь нажмет клавишу <F1>. Если вывод справки не предусмотрен, то в качестве параметра следует указать ноль.

Значение, возвращаемое функцией MessageDlg (табл. 2.15), позволяет определить, какая кнопка была нажата пользователем для завершения диалога. Если в окне сообщения отображается одна кнопка (очевидно, что в этом случае не нужно проверять, какую кнопку нажал пользователь), то функцию MessageDld можно вызвать как процедуру.

Таблица 2.13. *Сообщения*

Сообщение	Тип сообщения	Значок
Warning (Внимание)	mtWarning	
Error (Ошибка)	mtError	
Information (Информация)	mtInformation	
Confirmation (Подтверждение)	mtConfirmation	

Таблица 2.14. *Идентификаторы кнопок*

Идентификатор	Кнопка
mbYes	Yes
mbNo	No
mbOK	OK
mbCancel	Cancel
mbHelp	Help

Таблица 2.15. *Значения функции* MessageDlg

Значение	Диалог завершен нажатием кнопки
mrYes	Yes
mrOk	Ok
mrNo	No
mrCancel	Cancel

Для вывода сообщения вместо функции `MessageDlg` можно использовать процедуру `ShowMessage`. Эта процедура выводит окно с сообщением и кнопкой **OK**. На рис. 2.35 приведен пример окна сообщения — результат выполнения инструкции `ShowMessage('Надо ввести данные в оба поля').`

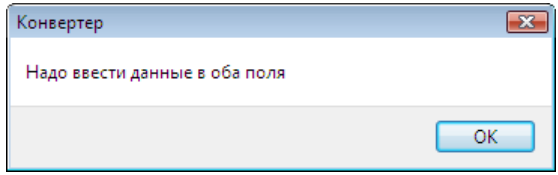


Рис. 2.35. Сообщение, выведенное процедурой `ShowMessage`

Обратите внимание, что в заголовке окна отображается имя приложения, указанное на вкладке **Application** окна **Project Options** (окно становится доступным в результате выбора в меню **Project** команды **Options**) или, если имя приложения не задано, имя проекта.

Внесение изменений

Программу "Конвертер" можно усовершенствовать. Например, сделать так, чтобы в поля редактирования пользователь мог ввести только правильную информацию, и чтобы в результате нажатия клавиши <Enter> в поле **Курс** курсор переходил в поле **Цена**, а при нажатии этой же клавиши в поле **Цена** становилась активной кнопка **Пересчет**. Можно сделать так, чтобы кнопка **Пересчет** была доступной только в том случае, если данные есть в обоих полях редактирования.

Чтобы внести изменения в программу, нужно открыть соответствующий проект. Сделать это можно обычным способом, выбрав в меню **File** команду **Open Project**. Можно выбрать нужный проект из списка проектов, над которыми в последнее время работал программист. Этот список становится доступным в результате выбора в меню **File** команды **Reopen**.

Чтобы программа "Конвертер" работала так, как было описано ранее, надо создать процедуры обработки событий `KeyPress` и `Change` для полей редактирования (компонентов `Edit1` и `Edit2`). Процедура обработки события `KeyPress` (для каждого компонента своя) обеспечивает фильтрацию вводимых пользователем символов. Она проверяет символ нажатой клавиши (символ передается в процедуру обработки события через параметр `Key`) и, если символ "запрещен", заменяет его на так называемый нуль-символ (в результате запрещенный символ в поле редактирования не отображается). Процедура обработки события `Change` (событие возникает, если текст, находящийся в поле редактирования, изменился, например в результате нажатия какой-либо клавиши в поле редактирования) управляет доступностью кнопки **Пересчет**. Она проверяет, есть ли данные в полях редактирования, и, если в ка-

ком-либо из полей данных нет, присваивает свойству `Enabled` кнопки `Button1` значение `False` и тем самым делает кнопку недоступной. Следует обратить внимание, что действие, которое надо выполнить, если изменилось содержимое поля `Edit1`, ничем не отличается от действия, которое надо выполнить, если изменилось содержимое поля `Edit2`. Поэтому обработку события `Change` для обоих компонентов выполняет *одна* процедура. Чтобы одна процедура могла обрабатывать события разных компонентов, сначала надо создать процедуру обработки события для одного компонента, а затем указать имя этой процедуры в качестве имени процедуры обработки события другого компонента.

В листинге 2.6 приведен модуль главной формы усовершенствованной программы "Конвертер". Обратите внимание, что в процедуре обработки события `Click` кнопки **Пересчет** нет инструкций обработки исключений. Исключения не могут возникнуть, т. к. пользователь просто не сможет ввести в поля редактирования неверные данные.

Листинг 2.6. Модуль главной формы программы "Конвертер" (usd2rub.pas)

```
// нажатие клавиши в поле Курс
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    '0'..'9', #8: ;  // цифры и <Backspace>

       { Обработку десятичного разделителя сделаем "интеллектуальной".
         Заменим разделитель (точку или запятую) символом
         DecimalSeparator, который должен
         использоваться при записи дробных чисел }
    '.',',':
      begin
        Key := DecimalSeparator;
        // проверим, введен ли уже в поле Edit десятичный разделитель
        if pos(DecimalSeparator,Edit1.Text) <> 0
          then Key := #0;
      end;
    #13: Edit2.SetFocus;   // <Enter> — курсор в поле Edit2
    else Key := #0;        // остальные символы запрещены
  end;
end;

// нажатие клавиши в поле Цена
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
```

```
begin
  case Key of
    '0'..'9', #8: ;  // цифры и <Backspace>
    '.',',':
      begin
        Key := DecimalSeparator;
        // проверим, введен ли уже в поле Edit десятичный разделитель
        if pos(DecimalSeparator,Edit1.Text) <> 0
          then Key := #0;
      end;
    #13: Button1.SetFocus; // сделать активной кнопку Пересчет
    else Key := Char(0);   // остальные символы запрещены
  end;
end;

// текст, находящийся в поле редактирования, изменился
procedure TForm1.EditChange(Sender: TObject);
begin
  // проверим, есть ли данные в полях редактирования
  if (Length(Edit1.Text) = 0) or (Length(Edit1.Text) = 0)
    then Button1.Enabled := False   // кнопка Пересчет недоступна
    else Button1.Enabled := True;   // кнопка Пересчет доступна
end;

// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real;   // цена в долларах
  k:   real;   // курс
  rub: real;   // цена в рублях

begin
  // получить данные из полей редактирования
  k := StrToFloat(Edit1.Text);
  usd := StrToFloat(Edit2.Text);

  // пересчитать цену из долларов в рубли
  rub := usd * k;
  // вывести результат расчета в поле Label3
  Label3.Caption := FloatToStrF(usd, ffNumber, 6,2) + ' $ = ' +
                    FloatToStrF(rub,ffCurrency,6,2);
end;
```

```
// щелчок на кнопке Завершить
procedure TForm1.Button2Click(Sender: TObject);
begin
  Form1.Close; // закрыть окно
end;
```

Настройка приложения

После того как приложение будет отлажено, можно выполнить его окончательную настройку: задать название программы и значок, который будет ее изображать в папке, на рабочем столе и на панели задач (во время работы программы).

Настройка приложения выполняется в окне **Project Options** (рис. 2.36), которое становится доступным в результате выбора в меню **Project** команды **Options**.

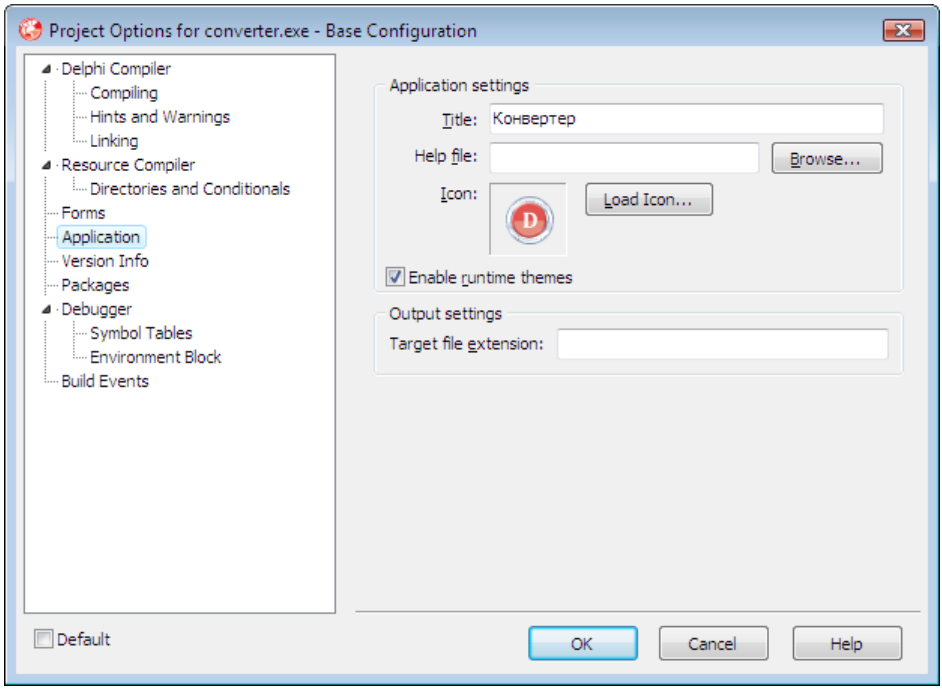


Рис. 2.36. Название программы надо ввести в поле **Title**, затем щелкнуть на кнопке **Load Icon** и выбрать значок

Название программы (его надо ввести в поле **Title**) отображается во время ее работы на панели задач, а также в заголовках окон сообщений, выводимых функцией `ShowMessage` (если название приложения не задано, то на панели задач и в заголовках окон сообщений отображается имя выполняемого файла).

Значок приложения изображает exe-файл программы в папке, на рабочем столе и в меню команд. Чтобы задать значок приложения, надо щелкнуть на кнопке **Load Icon** и, используя стандартное окно просмотра папок, найти подходящий значок (файл с расширением ico).

Программист может создать для своего приложения уникальный значок. Сделать это можно, например, с помощью утилиты Borland Image Editor, но, к сожалению, в комплект поставки Delphi XE она не входит.

Установка приложения на другой компьютер

Приложение, созданное в Delphi XE, можно перенести на другой компьютер, например, с помощью флэш-накопителя. Однако следует обратить внимание на то, что по умолчанию программа компилируется в режиме использования динамических библиотек (runtime packages) и поэтому она будет работать на другом компьютере только в том случае, если на нем есть необходимые библиотеки. Таким образом, помимо самой программы (exe-файла) на компьютер пользователя надо установить динамические библиотеки, необходимые для работы программы. Это как минимум так называемая библиотека времени выполнения (Runtime Library, файл rtl150.bpl) и библиотека визуальных компонентов (Visual Components Library, файл vcl150.bpl). Указанные файлы следует поместить в папку приложения или, если библиотеки будут использоваться несколькими программами, — в папку C:\Windows\System32.

Замечание

Имена библиотек, используемых программами, созданными в Delphi, состоят из двух частей. Первая часть имени, например rtl или vcl, несет информацию о назначении библиотеки (rtl — Runtime Library, vcl — Visual Components Library). Вторая часть имени содержит информацию о версии библиотеки. Например, 150 — это библиотека Delphi XE, 140 — Delphi 2010, 70 — Delphi 7.

Следует обратить внимание на то, что весь необходимый для работы программы код можно включить в exe-файл (в этом случае файлы динамических библиотек не нужны). Чтобы это сделать, надо, перед тем как выполнить компиляцию, в окне **Project Options** (оно становится доступным в результате выбора в меню **Project** команды **Options**) развернуть раздел **Packages** и сбросить флажок **Build with runtime packages** (рис. 2.37).

Профессиональный подход к разработке программного обеспечения предполагает, что программист создает не только приложение, но и программу, обеспечивающую установку приложения на компьютер пользователя (установщик). Создать установщик можно, например, с помощью утилиты InstallAware *(см. главу 9)*.

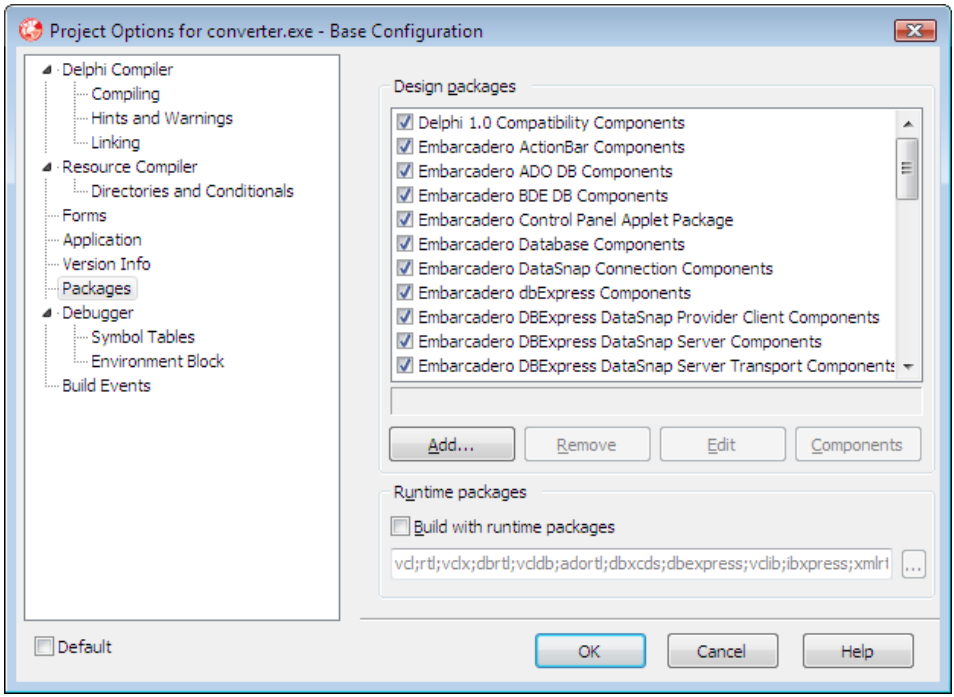


Рис. 2.37. Чтобы программа не использовала динамические библиотеки, надо сбросить флажок **Build with runtime packages**

ГЛАВА 3

Компоненты

В этой главе на конкретных примерах демонстрируется назначение основных (базовых) компонентов, а также компонентов для Windows Vista.

Базовые компоненты

К базовым можно отнести компоненты, обеспечивающие взаимодействие с пользователем (`Edit`, `Label`, `Button` и др.), отображение иллюстраций (`Image`), стандартных диалоговых окон (`SaveDialog`, `OpenDialog`) и некоторые другие, например `Timer`. Базовые компоненты находятся на вкладках **Standard**, **Additional**, **Win32**, **System** и **Dialogs**.

Label

Компонент `Label`, его значок (рис. 3.1) находится на вкладке **Standard**, предназначен для отображения текста. Чтобы задать текст, отображаемый в поле компонента, надо присвоить значение свойству `Caption`. Сделать это можно как при разработке формы, так и во время работы программы.

Свойства компонента приведены в табл. 3.1.

Рис. 3.1. Значок компонента `Label`

Таблица 3.1. *Свойства компонента* `Label`

Свойство	Описание
`Name`	Имя (идентификатор) компонента
`Caption`	Отображаемый текст
`Left`	Расстояние от левой границы поля вывода до левой границы формы

***Таблица 3.1** (окончание)*

Свойство	Описание
`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 (значение свойства `AutoSize` должно быть `False`)
`Alignment`	Задает способ выравнивания текста внутри поля. Текст может быть выровнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Font`	Шрифт, используемый для отображения текста. Уточняющие свойства определяют шрифт (`Name`), размер (`Size`), стиль (`Style`) и цвет символов (`Color`)
`ParentFont`	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно `True`, то текст выводится шрифтом, установленным для формы
`Color`	Цвет фона области вывода текста
`Transparent`	Управляет отображением фона области вывода текста. Значение `True` делает область вывода текста прозрачной (область вывода не закрашивается цветом, заданным свойством `Color`)
`Visible`	Позволяет скрыть текст (`False`) или сделать его видимым (`True`)

Возможности компонента `Label` демонстрирует программа "Доход" (ее форма приведена на рис. 3.2, а текст процедуры обработки события `Click` на кнопке **Ok** — в листинге 3.1).

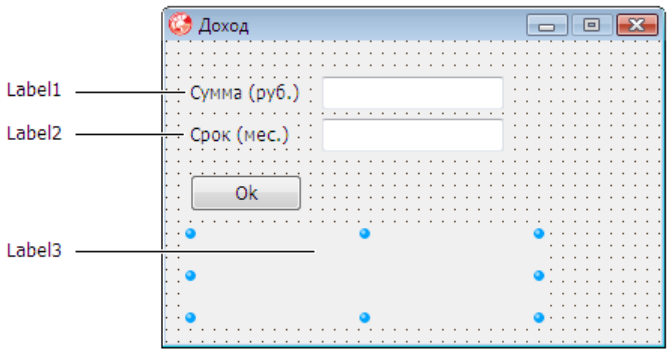


Рис. 3.2. Форма программы "Доход"

Она показывает, как во время работы программы изменить цвет текста, отображаемого в поле компонента, как вывести в поле компонента значение переменной, а также как разбить отображаемый текст на строки.

Программа вычисляет доход по вкладу. Если пользователь оставит какое-либо из полей незаполненным, то в результате щелчка на кнопке **Ok** в поле компонента `Label3` красным цветом отображается сообщение об ошибке. Если все поля формы заполнены, то в поле компонента `Label3` в две строки выводится результат расчета. Разбиение текста на строки обеспечивает символ "новая строка" (его код равен 10). Добавить нужный символ в формируемую строку можно с помощью функции `Chr`. Вместо функции `Chr` можно указать код символа, поставив перед значением "решетку" (#). Именно этот способ и используется в рассматриваемой программе. Значения свойств компонентов `Label` приведены в табл. 3.2.

Таблица 3.2. *Значения свойств компонентов* `Label`

Компонент	Свойство	Значение
Label1	Caption	Сумма (руб.)
Label2	Caption	Срок (мес.)
Label3	Caption	
	AutoSize	False
	WordWrap	True
	Width	233
	Height	57

Листинг 3.1. Обработка события `Click` на кнопке *Ok*

```
// щелчок на кнопке Ok
procedure TForm1.Button1Click(Sender: TObject);
var
  sum, pr: real;       // сумма, процентная ставка
  srok: integer;       // срок вклада, месяцев
  dohod, sum2 : real;  // доход и сумма в конце срока вклада

begin
  if (Length(Edit1.Text) = 0) or (Length(Edit2.Text) = 0)
  then begin
        Label3.Font.Color := clMaroon;  // темно-красный
        Label3.Caption := 'Надо заполнить все поля формы';
       end
```

```
  else begin
        sum := StrToFloat(Edit1.Text);
        srok := StrToInt(Edit2.Text);

        // определить процентную ставку
        case srok of
          1..3:   pr := 9.5;
          4..6:   pr := 11;
          7..12:  pr := 12.5;
          13..24: pr := 14;
          else
                  pr:=18;
        end;

        dohod :=  sum * (pr / 100/ 12) * srok;
        sum2 := sum + dohod;

        Label3.Font.Color := clNavy;
        Label3.Caption :=
           'Сумма: ' + FloatToStrF(sum,ffCurrency,6,2) + #10 +
           'Процент (годовых):' + FloatToStrF(pr,ffNumber,2,2) +  #10 +
           'Доход: ' +  FloatToStrF(dohod, ffCurrency,6,2) +  #10 +
           'Сумма в конце срока: ' + FloatToStrF(sum2,ffCurrency,6,2);

      end;
end;
```

Edit

Компонент `Edit`, его значок (рис. 3.3) находится на вкладке **Standard**, обеспечивает ввод и редактирование текста. Свойства компонента приведены в табл. 3.3.

Рис. 3.3. Значок компонента `Edit`

Таблица 3.3. *Свойства компонента* `Edit`

Свойство	Описание
`Name`	Имя (идентификатор) компонента
`Text`	Текст, находящийся в поле ввода и редактирования
`Left`	Расстояние от левой границы компонента до левой границы формы

Таблица 3.3 *(окончание)*

Свойство	Описание
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота
`Width`	Ширина
`Font`	Шрифт, используемый для отображения текста
`ParentFont`	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно `True`, то при изменении свойства `Font` формы автоматически меняется значение свойства `Font` компонента
`Enabled`	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно `False`, то текст в поле редактирования изменить нельзя
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

В поле компонента `Edit` отображаются все символы, которые пользователь набирает на клавиатуре. Вместе с тем программист может, создав процедуру обработки события `KeyPress`, запретить ввод (отображение) некоторых символов.

Использование компонента `Edit` для ввода данных различного типа демонстрирует программа "Компонент Edit" (ее форма приведена на рис. 3.4, а текст — в листинге 3.2). Программа спроектирована таким образом, что в режиме ввода текста в поле редактирования можно ввести любой символ, в режиме ввода целого числа — только цифры и знак "–" (если это первый символ). В режиме ввода дробного числа кроме цифр и знака "–" в поле компонента можно ввести символ-разделитель (запятую или точку, в зависимости от настройки операционной системы).

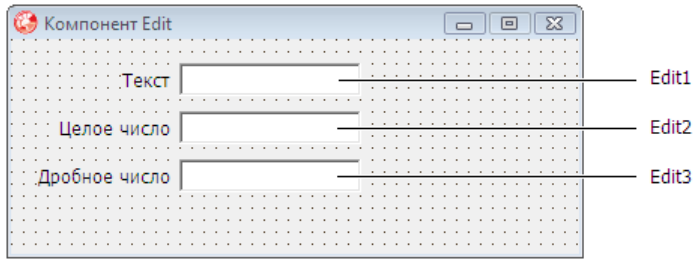


Рис. 3.4. Форма программы "Компонент Edit"

Листинг 3.2. Компонент `Edit`

```
// клавиша нажата в поле "Текст"
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
```

```
begin
  if key = #13 // клавиша <Enter>
    then Edit2.SetFocus; // переместить курсор в поле Edit2
end;

// клавиша нажата в поле "Целое число"
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
       '0'..'9', #8: ;      // цифры и <Backspace>
       #13: Edit3.SetFocus; // <Enter> — переместить курсор в поле Edit3
       '-': if Length(Edit2.Text) <> 0 then Key := #0;
       else Key := #0;     // остальные символы не отображать
  end;
end;

// клавиша нажата в поле "Дробное число"
procedure TForm1.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    '0'..'9', #8: ;  // цифры и <Backspace>

    '.',',':
       begin
         // DecimalSeparator — глобальная переменная, в которой
         // находится символ "десятичный разделитель"
         Key := DecimalSeparator;
         if pos(DecimalSeparator,Edit3.Text) <> 0
           then Key := #0;
       end;
    '-': if Length(Edit3.Text) <> 0 then Key := #0;
    else Key := #0; // остальные символы не отображать
  end;
end;
```

Button

Компонент `Button`, его значок (рис. 3.5) находится на вкладке **Standard**, представляет собой командную кнопку. Свойства компонента приведены в табл. 3.4.

Рис. 3.5. Значок компонента `Button`

Таблица 3.4. *Свойства компонента* `Button`

Свойство	Описание
`Name`	Имя (идентификатор) компонента
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы
`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 (в результате щелчка на кнопке событие `Click` не возникает)
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Hint`	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, значение свойства `ShowHint` должно быть `True`)
`ShowHint`	Разрешает (`True`) или запрещает (`False`) отображение подсказки при позиционировании указателя на кнопке

Использование компонента `Button` демонстрирует программа "Версты—километры" (ее форма приведена на рис. 3.6, а текст — в листинге 3.3). Программа пересчитывает расстояние из километров в версты. Расчет и отображение результата выполняет процедура обработки события `Click` на кнопке **OK**. Следует обратить внимание, что кнопка **OK** доступна только в том случае, если в поле редактирования находятся данные (хотя бы одна цифра). Управляет доступностью кнопки процедура обработки события `Change` компонента `Edit1`. Процедура контролирует количество символов, которое находится в поле редактирования, и, если в поле нет ни одной цифры, присваивает значение `False` свойству `Enabled` и тем самым делает кнопку недоступной. В процессе создания формы свойству `Enabled` кнопки надо присвоить значение `False`.

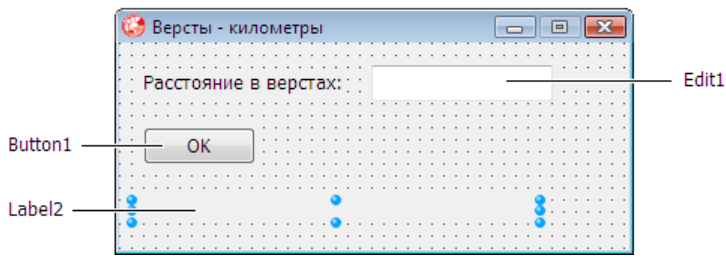


Рис. 3.6. Форма программы "Версты—километры"

Листинг 3.3. Версты—километры

```
// текст в поле редактирования изменен
procedure TForm1.Edit1Change(Sender: TObject);
begin
  if Length(Edit1.Text) = 0 then
      Button1.Enabled := False
  else
      Button1.Enabled := True;
end;

// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  km: real;  // расстояние в километрах
  v: real;   // расстояние в верстах
begin
  km := StrToFloat(Edit1.Text);
  v := km / 1.0668;
  Label2.Caption := Edit1.Text + ' км это - ' +
                    FloatToStrF(v,ffFixed,7,2) + ' верст';
end;

// нажатие клавиши в поле редактирования
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    case Key of
      '0'..'9', #8: ;  // цифры и <Backspace>

      '.',',':
         begin
           // DecimalSeparator — глобальная переменная, в которой
           // находится символ "десятичный разделитель"
           Key := DecimalSeparator;
           if pos(DecimalSeparator,Edit1.Text) <> 0
             then Key := #0;
         end;
      #13: Button1.SetFocus;
      else Key := #0; // остальные символы не отображать
    end;
end;
```

CheckBox

Компонент CheckBox, его значок (рис. 3.7) находится на вкладке **Standard**, представляет собой флажок, который может находиться в одном из двух состояний: выбранном или невыбранном. Часто вместо "выбранный" говорят "установленный", а вместо "невыбранный" — "сброшенный". Рядом с флажком обычно находится поясняющий текст. Свойства компонента CheckBox приведены в табл. 3.5.

Рис. 3.7. Значок компонента CheckBox

Таблица 3.5. *Свойства компонента* CheckBox

Свойство	Описание
Name	Имя (идентификатор) компонента
Caption	Комментарий (текст, который находится справа от флажка)
Checked	Состояние флажка: если флажок установлен (в квадратике есть "галочка"), то значение Checked равно True; если флажок сброшен (нет "галочки"), то значение Checked равно False
State	Состояние флажка. В отличие от свойства Checked, позволяет различать установленное, сброшенное и промежуточное состояния. Состояние флажка определяет одна из констант: cbChecked (установлен), cbUnChecked (сброшен), cbGrayed (серый, неопределенное состояние)
AllowGrayed	Свойство определяет, может ли флажок находиться в промежуточном состоянии: если значение AllowGrayed равно False, то флажок может быть только установленным или сброшенным; если значение AllowGrayed равно True, то допустимо промежуточное состояние, когда флажок и не установлен, и не сброшен. Если компонент находится в промежуточном состоянии, то он окрашен в серый (gray) цвет
Left	Расстояние от левой границы флажка до левой границы формы
Top	Расстояние от верхней границы флажка до верхней границы формы
Height	Высота поля вывода поясняющего текста
Width	Ширина поля вывода поясняющего текста
Font	Шрифт, используемый для отображения поясняющего текста
ParentFont	Признак наследования характеристик шрифта родительской формы

Использование компонентов CheckBox демонстрирует программа "Комплектация" (ее форма приведена на рис. 3.8, а текст — в листинге 3.4). Программа позволяет посчитать цену автомобиля в комплектации, выбранной пользователем. Значения свойств компонентов CheckBox приведены в табл. 3.6.

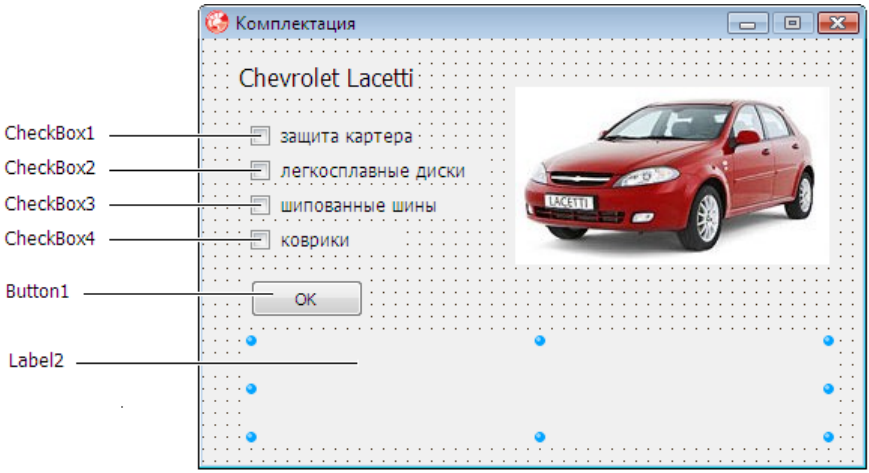


Рис. 3.8. Форма программы "Комплектация"

Таблица 3.6. *Значения свойств компонентов* `CheckBox`

Компонент	Свойство	Значение
CheckBox1	Caption	защита картера
	Checked	False
CheckBox2	Caption	легкосплавные диски
	Checked	False
CheckBox3	Caption	шипованные шины
	Checked	False
CheckBox4	Caption	коврики
	Checked	False

Листинг 3.4. Комплектация

```
procedure TForm1.Button1Click(Sender: TObject);
var
  cena: real;      // цена в базовой комплектации
  dop: real;       // сумма за доп. оборудование
  discount: real; // скидка
  total: real;     // общая сумма

  st: string;      // сообщение — результат расчета
begin
```

```
  cena := 415000;
  dop := 0;

  if (CheckBox1.Checked)
    then      // защита картера
      dop := dop + 4500;

  if (CheckBox2.Checked)
    then      // зимние шины
      dop := dop +  12000;

  if (CheckBox3.Checked)
    then      // литые диски
      dop := dop + 12000;

  if (CheckBox4.Checked)
    then      // коврики
       dop := dop + 1200;

  total := cena + dop;

  st := 'Цена в выбранной комплектации: ' +
        FloatToStrF(total, ffCurrency, 6,2);

  if (dop <> 0) then
     st := st + #10+ 'В том числе доп. оборудование: ' +
           FloatToStrF(dop, ffCurrency, 6,2);

  if ((CheckBox1.Checked) and (CheckBox2.Checked) and
      (CheckBox3.Checked) and (CheckBox4.Checked))
    then
      begin
        // скидка предоставляется, если выбраны все опции
        discount := dop * 0.1;
        total := total - discount;
        st := st + #10 + 'Скидка на доп. оборудование (10%): ' +
              FloatToStrF(discount, ffCurrency, 6,2) + #10 +
              'Итого: ' + FloatToStrF(total, ffCurrency, 6,2);
      end;

  Label2.Caption := st;
end;
```

RadioButton

Компонент RadioButton, его значок (рис. 3.9) находится на вкладке **Standard**, представляет собой кнопку (переключатель), состояние которой зависит от состояния других компонентов RadioButton, находящихся на форме. Обычно компоненты RadioButton объединяют в группу (достигается это путем размещения нескольких компонентов в поле компонента GroupBox). В каждый момент времени только одна из кнопок группы может находиться в выбранном состоянии (возможна ситуация, когда ни одна из кнопок не выбрана). Состояние кнопок, принадлежащих одной группе, не зависит от состояния кнопок, принадлежащих другой группе. Свойства компонента RadioButton приведены в табл. 3.7.

Рис. 3.9. Значок компонента RadioButton

Таблица 3.7. *Свойства компонента* RadioButton

Свойство	Описание
Name	Имя (идентификатор) компонента
Caption	Комментарий (текст, который находится справа от кнопки)
Checked	Состояние. Определяет вид кнопки: True — кнопка выбрана, False — кнопка не выбрана (выбрана другая кнопка группы)
Left	Расстояние от левой границы флажка до левой границы формы
Top	Расстояние от верхней границы флажка до верхней границы формы
Height	Высота поля вывода поясняющего текста
Width	Ширина поля вывода поясняющего текста
Font	Шрифт, используемый для отображения поясняющего текста
ParentFont	Признак наследования характеристик шрифта родительской формы

Использование компонента RadioButton демонстрирует программа "Фото" (ее форма приведена на рис. 3.10, а текст — в листинге 3.5). Программа вычисляет стоимость печати фотографий. Значения свойств компонентов RadioButton приведены в табл. 3.8.

Таблица 3.8. *Значения свойств компонентов* RadioButton

Компонент	Свойство	Значение
RadioButton1	Caption	9 x 12
	Checked	True

Таблица 3.8 *(окончание)*

Компонент	Свойство	Значение
RadioButton2	Caption	10 x 15
	Checked	False
RadioButton3	Caption	18 x 24
	Checked	False

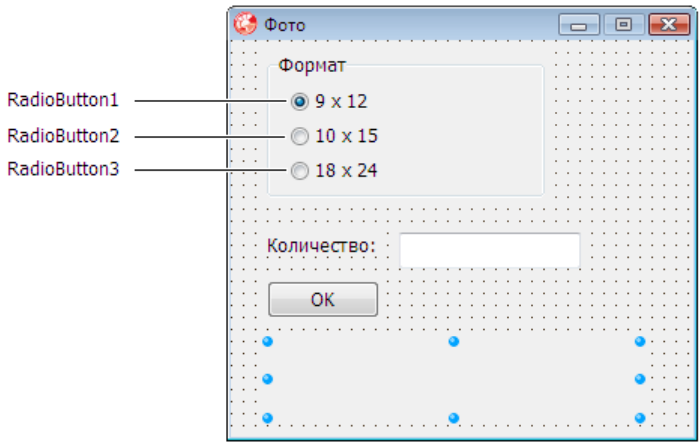


Рис. 3.10. Форма программы "Фото"

Листинг 3.5. Фото

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  cena: real;     // цена 1 шт.
  kol: integer;   // количество
  sum: real;      // сумма

  begin
  // определить цену одной фотографии
  if RadioButton1.Checked then
      // 9x12
      cena := 3.50;

  if RadioButton2.Checked then
      // 10x15
      cena := 4.50;
```

```
  if RadioButton3.Checked then
      // 18x24
      cena := 14;

   kol := StrToInt(Edit1.Text);

   sum := cena * kol;

   Label2.Caption := 'Цена: ' + FloatToStrF(cena,ffCurrency,3,2)+ #10 +
                     'Кол-во: ' + IntToStr(kol) + #10 +
                     'Сумма заказа: ' + FloatToStrF(sum,ffCurrency,3,2);
end;

// изменилось содержимое поля редактирования
procedure TForm1.Edit1Change(Sender: TObject);
begin
  // если количество фотографий не задано (поле Edit1 пустое),
  // сделаем кнопку OK недоступной
  if Length(Edit1.Text) = 0 then
    // в поле нет текста
    Button1.Enabled := False
  else
    // текст в поле есть
    Button1.Enabled := True;
end;

// нажатие клавиши в поле Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  // по условию задачи в поле "Количество"
  // можно ввести только целое число
  case Key of
     '0'..'9', #8: ;          // цифры и <Backspace>
     #13: Button1.SetFocus;   // <Enter> — переместить фокус на Button1
     else Key := #0;          // остальные символы не отображать
  end;
end;
```

ComboBox

Компонент `ComboBox`, его значок (рис. 3.11) находится на вкладке **Standard**, представляет собой комбинацию поля редактирования и списка, что дает возмож-

ность ввести данные путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 3.9.

Рис. 3.11. Значок компонента `ComboBox`

***Таблица 3.9.** Свойства компонента `ComboBox`*

Свойство	Описание
`Name`	Имя (идентификатор) компонента
`Style`	Вид компонента: `csDropDown` — поле ввода и раскрывающийся список (данные можно ввести в поле редактирования или выбрать в списке); `csDropDownList` — только раскрывающийся список; `csSimple` — только поле редактирования
`Text`	Текст, находящийся в поле ввода/редактирования
`Items`	Элементы списка — массив строк
`Count`	Количество элементов списка
`ItemIndex`	Номер элемента, выбранного в списке (элементы нумеруются с нуля). Если ни один из элементов списка не выбран, то значение свойства равно –1
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента
`DropDownCount`	Количество элементов, отображаемых в раскрытом списке. Если количество элементов списка больше `DropDownCount`, то появляется вертикальная полоса прокрутки
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента (поля ввода/редактирования)
`Width`	Ширина компонента
`Font`	Шрифт, используемый для отображения элементов списка
`ParentFont`	Признак наследования свойств шрифта родительской формы

Список, отображаемый в поле компонента, можно сформировать во время создания формы или во время работы программы. Чтобы сформировать список во время создания формы, надо выбрать свойство `Items`, щелкнуть на находящейся в поле значения свойства кнопке и в окне **String List Editor** ввести элементы списка (рис. 3.12).

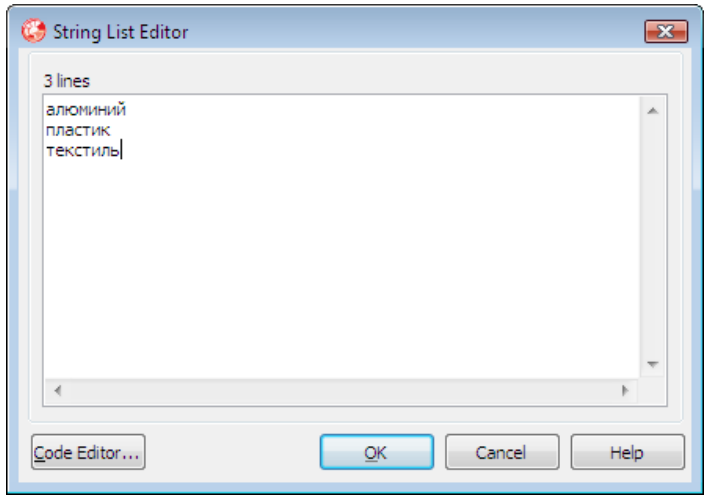


Рис. 3.12. Формирование списка компонента во время разработки формы

Чтобы сформировать список во время работы программы (добавить в список элемент), надо применить метод `Add` к свойству `Items`. Например:

```
ComboBox1.Items.Add('алюминий');
ComboBox1.Items.Add('пластик');
ComboBox1.Items.Add('текстиль');
```

Использование компонента `ComboBox` демонстрирует программа "Жалюзи" (ее форма приведена на рис. 3.13). Материал, из которого должны быть сделаны жалюзи, выбирается в списке `ComboBox`. Настройку компонента `ComboBox` выполняет конструктор формы. Процедура обработки события `Click` на командной кнопке `Button1` и процедура обработки события `Create` формы приведены в листинге 3.6.

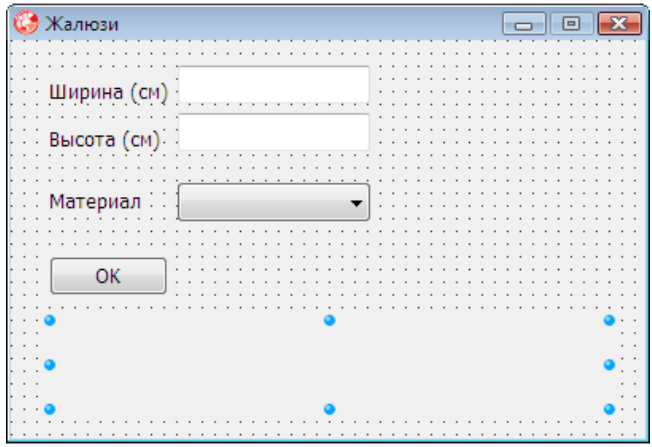


Рис. 3.13. Форма программы "Жалюзи"

Листинг 3.6. Жалюзи

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  w,h: real;   // ширина, высота (см)
  s: real;     // площадь (кв. м)
  c: real;     // цена за 1 кв. м
  summ: real;  // сумма
  st: string;  // сообщение

begin
  if   (Length(Edit1.Text) = 0)   // не задана ширина
    or (Length(Edit1.Text) = 0)   // не задана высота
    or (ComboBox1.ItemIndex = -1) // не выбран материал
  then begin
         ShowMessage('Надо задать ширину, высоту и выбрать материал');
         exit;
       end;

  // вычислить площадь
  w := StrToFloat(Edit1.Text);
  h := StrToFloat(Edit2.Text);
  s := w * h / 10000;

  // определить цену
  // ItemIndex — номер элемента, выбранного в списке
  case ComboBox1.ItemIndex of
    0: c := 700; // алюминий
    1: c := 450; // пластик
    2: c := 300; // текстиль
  end;

  // расчет
  summ := s * c;

  st := 'Размер: ' + Edit1.Text + 'x' + Edit2.Text + ' см' + #10 +
        'Материал: ' + ComboBox1.Text + #10 +
        'Сумма: ' + FloatToStrF(summ, ffCurrency, 6,2);

  Label4.Caption := st;
end;
```

```
// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Style := csDropDownList;
  ComboBox1.Items.Add('алюминий');
  ComboBox1.Items.Add('пластик');
  ComboBox1.Items.Add('текстиль');
end;
```

ListBox

Компонент `ListBox`, его значок (рис. 3.14) находится на вкладке **Standard**, представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 3.10.

Рис. 3.14. Значок компонента `ListBox`

Таблица 3.10. *Свойства компонента* `ListBox`

Свойство	Описание
`Name`	Имя (идентификатор) компонента
`Items`	Элементы списка
`Count`	Количество элементов списка
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента
`ItemIndex`	Номер выбранного элемента (элементы списка нумеруются с нуля). Если в списке ни один из элементов не выбран, то значение свойства равно –1
`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
`ParentFont`	Признак наследования свойств шрифта родительской формы

Список, отображаемый в поле компонента, можно сформировать при создании формы или во время работы программы. Чтобы сформировать список во время создания формы, надо выбрать свойство `Items`, щелкнуть на находящейся в поле

значения свойства кнопке и в окне **String List Editor** ввести элементы списка. Формирование списка во время работы программы обеспечивает метод `Add` свойства `Items`.

Использование компонента `ListBox` демонстрирует программа "Просмотр иллюстраций" (ее окно приведено на рис. 3.15, форма представлена на рис. 3.16, а текст — в листинге 3.7). Программа позволяет просмотреть фотографии (JPEG-файлы).

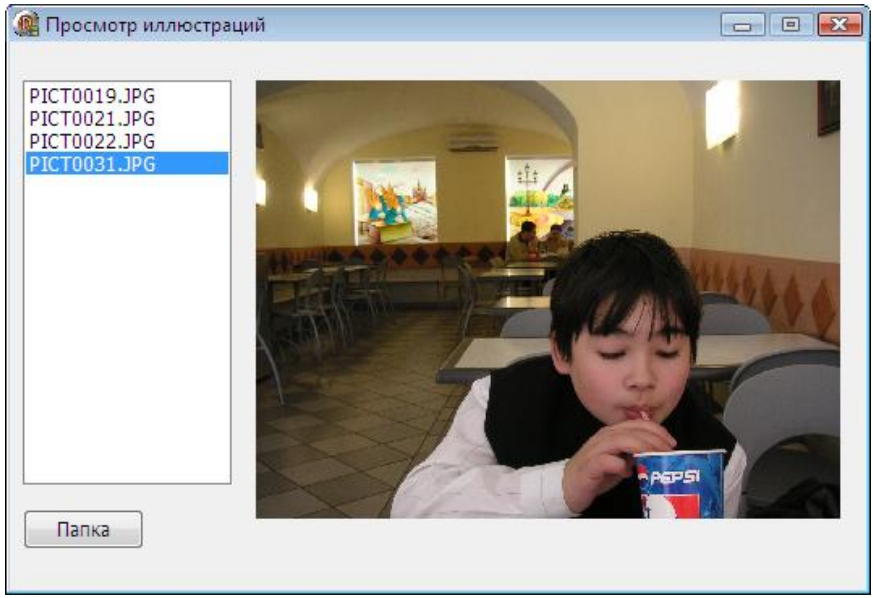


Рис. 3.15. Окно программы "Просмотр иллюстраций"

Компонент `ListBox` используется для выбора фотографии. Заполняет список компонента `ListBox` процедура `FillListBox` (ее объявление надо поместить в секцию `Protected` объявления формы). В начале работы программы процедуру `FillListBox` вызывает процедура обработки события `Create` формы. Процедура обработки события `Click` на кнопке **Папка** отображает стандартное окно **Обзор папок** (отображение диалога обеспечивает процедура `SelectDirectory`), затем вызывает функцию `FillListBox`. Фотография отображается в поле компонента `Image` *(см. разд. "Image" далее в этой главе)*. Для того чтобы иллюстрация отображалась без искажения, свойству `AutoSize` компонента `Image` надо присвоить значение `False`, а свойству `Proportional` — `True`. Отображение выбранной в списке иллюстрации осуществляет процедура обработки события `Click`, которое происходит в результате щелчка на элементе списка или перемещения указателя текущего элемента списка с помощью клавиш управления курсором. Следует обратить внимание, что в директиву `uses` модуля формы надо добавить ссылки на модули `FileCtrl` и `Jpeg`.

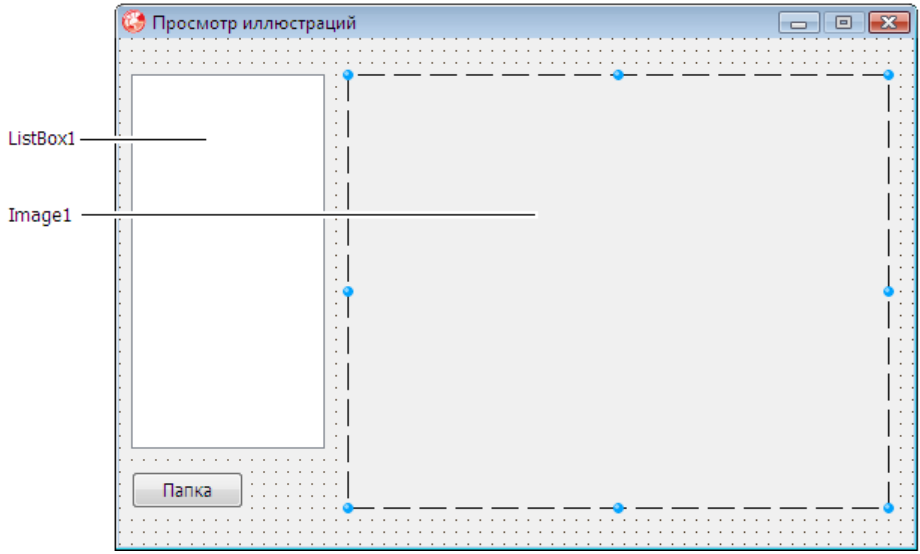


Рис. 3.16. Форма программы "Просмотр иллюстраций"

Листинг 3.7. Просмотр иллюстраций

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, FileCtrl, Jpeg;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    Button1: TButton;
    Image1: TImage;
    procedure ListBox1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    Path: string; // папка, которую выбрал пользователь
    Procedure FillListBox;
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

// заполняет список компонента ListBox (формирует список JPEG-файлов)
procedure TForm1.FillListBox;
var
  SearchRec: TSearchRec;  // результат поиска файла
  r: integer;
begin
  r := FindFirst(Path + '*.jpg', faAnyFile, SearchRec);
  if r = 0 then
  begin
    // в каталоге Path есть по крайней мере один JPEG-файл
    ListBox1.Items.Clear;
    ListBox1.Items.Add(SearchRec.Name);
    while 0 = FindNext(SearchRec) do
      begin
        ListBox1.Items.Add(SearchRec.Name);
      end;

    ListBox1.ItemIndex := 0;
    Image1.Picture.LoadFromFile(Path +
                                ListBox1.Items[ListBox1.ItemIndex]);
  end
  else begin
         // в выбранном каталоге нет илюстраций
         ListBox1.Items.Clear; // очистить список
         Image1.Picture.Bitmap.FreeImage;
       end;
end;

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  FillListBox;
end;
```

```
// щелчок в поле компонента ListBox
procedure TForm1.ListBox1Click(Sender: TObject);
var
  Filename: string;
begin
  FileName := Path + ListBox1.Items[ListBox1.ItemIndex];
  Image1.Picture.LoadFromFile(Filename);
end;

// щелчок на кнопке "Папка"
procedure TForm1.Button1Click(Sender: TObject);
begin
  if SelectDirectory('Выберите каталог', '',Path) then
  begin
    Path := Path + '\';
    Form1.Caption := 'Прсмотр иллюстраций - ' + Path;
    FillListBox;
  end;
end;

end.
```

Memo

Компонент Memo, его значок (рис. 3.17) находится на вкладке **Standard**, представляет собой элемент редактирования текста, который может состоять из нескольких строк. Свойства компонента приведены в табл. 3.11.

Рис. 3.17. Значок компонента Memo

Таблица 3.11. *Свойства компонента* Memo

Свойство	Описание
Name	Имя (идентификатор) компонента
Text	Текст, находящийся в поле Memo. Рассматривается как единое целое
Lines	Массив строк, соответствующий содержимому поля. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
Left	Расстояние от левой границы поля до левой границы формы
Top	Расстояние от верхней границы поля до верхней границы формы

Таблица 3.11 *(окончание)*

Свойство	Описание
`Width`	Ширина поля
`Height`	Высота поля
`Font`	Шрифт, используемый для отображения вводимого текста
`ParentFont`	Признак наследования свойств шрифта родительской формы
`ReadOnly`	Разрешает (`False`) или запрещает (`True`) редактирование текста, находящегося в поле редактирования
`ScrollBars`	Задает отображаемые полосы прокрутки (`ssVertical` — только вертикальная; `ssHorizontal` — только горизонтальная; `ssBoth` — вертикальная и горизонтальная; `ssNone` — полосы прокрутки не отображать)
`Modified`	Индикатор: `True` — текст изменен

Использование компонента `Memo` для отображения и редактирования текста, загруженного из файла, демонстрирует программа "Просмотр/редактирование" (ее форма представлена на рис. 3.18). Текст процедур обработки событий `Activate` и `CloseQuery` приведен в листинге 3.8. Загрузку текста из файла kurs.txt выполняет процедура обработки события `Activate` формы. Процедура обработки события `CloseQuery`, которое возникает в результате щелчка на кнопке **Закрыть**, проверяет, внесены ли в текст какие-либо изменения, и, если текст изменен (значение свойства `Modified` равно `True`), предлагает сохранить его (сообщение выводит функция `MessageDlg`).

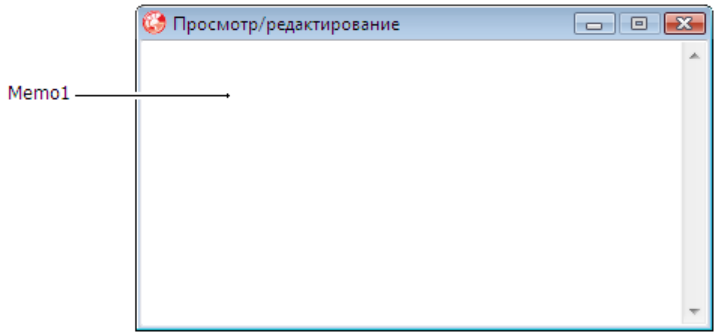


Рис. 3.18. Форма программы "Просмотр/редактирование"

Листинг 3.8. Просмотр/редактирование

```
const
  TEXTFILE = 'kurs.txt';
```

```
procedure TForm1.FormActivate(Sender: TObject);
begin
  // загрузить в компонент Memo1 текст из файла
  try
      Memo1.Lines.LoadFromFile(TEXTFILE);
      Form1.Caption := 'Просмотр/редактирование - ' + TEXTFILE
  except
      on e: EFOpenError do
          begin
              MessageDlg('Нет файла ' + TEXTFILE, mtError,[mbYes],0);
              Memo1.Enabled := False;
          end;
  end;
end;

// пользователь сделал щелчок на кнопке "Закрыть"
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var
  r: integer;
begin
  if Memo1.Modified  then
  begin
     // содержимое поля редактирования изменено
     r := MessageDlg('Файл изменен. Сохранить изменения?',
                     mtWarning,[mbYes,mbNo,mbCancel],0);
     case r of
       mrYes:    // записать текст в файл
                 Memo1.Lines.SaveToFile(TEXTFILE);
       mrCancel: // продолжить работу с программой
                 CanClose := False;
     end;
  end;
end;
```

Timer

Компонент `Timer`, его значок (рис. 3.19) находится на вкладке **System**, генерирует последовательность событий `Timer`.

Рис. 3.19. Значок компонента `Timer`

Компонент является невизуальным, т. е. во время работы программы не отображается на форме. Свойства компонента приведены в табл. 3.12.

Таблица 3.12. *Свойства компонента* `Timer`

Свойство	Описание
`Interval`	Период генерации события `Timer`. Задается в миллисекундах
`Enabled`	Разрешает (`True`) или запрещает (`False`) генерацию события `Timer`

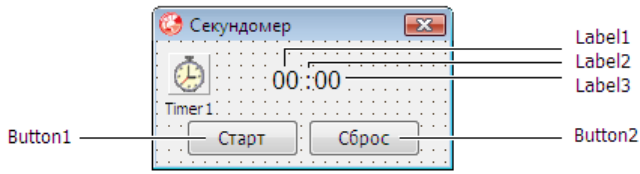


Рис. 3.20. Форма программы "Секундомер"

Использование компонента `Timer` демонстрирует программа "Секундомер" (ее форма приведена на рис. 3.20, а текст — в листинге 3.9). В процессе создания формы свойству `Interval` компонента `Timer1` надо присвоить значение `500`. Кнопка `Button1` предназначена как для запуска секундомера, так и для его остановки. В начале работы программы значение свойства `Enabled` компонента `Timer1` равно `False`, поэтому таймер не генерирует события `Timer`. Процедура обработки события `Click` на кнопке `Button1` проверяет состояние таймера и, если таймер не работает, присваивает свойству `Enabled` таймера значение `True` и тем самым запускает его. Процедура обработки события `Timer`, которое возникает с периодом 0,5 с, инвертирует значение свойства `Visible` компонента `Label2` (в результате чего двоеточие мигает) и, если двоеточие отображается, увеличивает счетчик времени, а также выводит в поле компонентов `Label1` и `Label3` значения счетчиков минут и секунд. Следует обратить внимание, что объявления счетчиков минут и секунд (переменных `s` и `m`) надо поместить в секцию `private` объявления формы. Если секундомер работает, то щелчок на кнопке `Button1` останавливает секундомер.

Листинг 3.9. Секундомер

```
// щелчок на кнопке "Старт/Стоп"
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Timer1.Enabled
    then
      // секундомер работает
```

```
      begin
        Timer1.Enabled := False; // остановить таймер
        Button1.Caption := 'Старт';
        Button2.Enabled := True;
      end
    else
      // секундомер остановлен
      begin
        Timer1.Enabled := True; // пустить таймер
        Button1.Caption := 'Стоп';
        Button2.Enabled := False;
      end;

end;

// сигнал таймера (событие Timer)
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // таймер генерирует событие Timer каждые 0,5 с

  // показать/скрыть двоеточие
  Label2.Visible := not Label2.Visible;

  if not Label2.Visible
    then exit;

  // в эту точку попадаем каждую секунду
  if s = 59 then
  begin
    inc(m);
    Label1.Caption := IntToStr(m);
    s := 0;
  end
  else
    inc(s);

  // отобразить секунды
  if s < 10 then
    Label3.Caption := '0'+ IntToStr(s)
  else
    Label3.Caption := IntToStr(s);
end;
```

```
// щелчок на кнопке "Сброс"
procedure TForm1.Button2Click(Sender: TObject);
begin
  s := 0;
  m := 0;
  Label1.Caption := '00';
  Label3.Caption := '00';
end;
```

Panel

Компонент `Panel`, его значок (рис. 3.21) находится на вкладке **Standard**, представляет собой панель, на поверхность которой можно поместить другие компоненты. Обычно панель используют для привязки компонентов к границе окна (при изменении размера окна компоненты, которые находятся на панели, не меняют своего положения относительно границы окна, к которой "привязана" панель).

Некоторые свойства компонента `Panel` приведены в табл. 3.13.

Рис. 3.21. Значок компонента `Panel`

Таблица 3.13. *Свойства компонента* `Panel`

Свойство	Описание
`Align`	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (`alLeft`), правой (`alRight`), верхней (`alTop`) или нижней (`alBottom`) границе
`BevelOuter`	Внешняя "фаска" панели. Если значение свойства равно `bvNone`, то фаска не отображается и поверхность панели находится на одном уровне с поверхностью формы. Если значение свойства равно `bvLowered`, то поверхность панели утоплена; если `bvRaised`, то поверхность панели приподнята над поверхностью формы
`Enabled`	Свойство позволяет сделать недоступными (`False`) все компоненты, которые находятся на панели

На рис. 3.22 приведено окно программы "Просмотр иллюстраций" *(см. разд. "Image" далее в этой главе)*, в котором кнопки управления (компоненты `SpeedButton`) размещены на панели. Панель привязана к нижней границе окна (значение свойства `Align` равно `alBottom`), поэтому даже в том случае, если пользователь развернет окно программы на весь экран, панель и, следовательно, кнопки все равно будут находиться в нижней части окна.

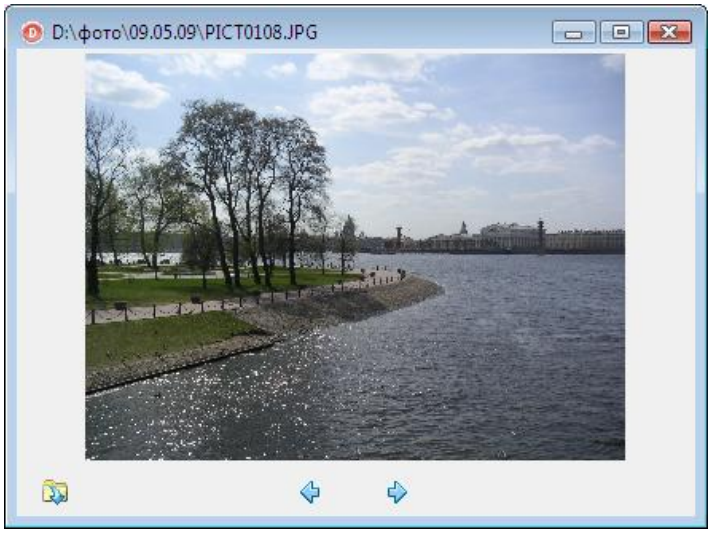


Рис. 3.22. Программа "Просмотр иллюстраций"

ControlBar

Компонент `ControlBar`, его значок (рис. 3.23) находится на вкладке **Additional**, представляет собой так называемую панель инструментов, на поверхность которой можно поместить другие компоненты. Обычно панель инструментов находится в верхней части окна, а на ее поверхности размещены командные кнопки.

Некоторые свойства компонента `TControlBar` приведены в табл. 3.14.

Рис. 3.23. Значок компонента `ControlBar`

Таблица 3.14. *Свойства компонента* `ControlBar`

Свойство	Описание
`Align`	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (`alLeft`), правой (`alRight`), верхней (`alTop`) или нижней (`alBottom`) границе
`AutoSize`	Если значение свойства равно `True`, то высота панели инструментов автоматически устанавливается равной высоте командных кнопок, находящихся на панели
`DrawingStyle`	Способ закраски панели: градиент (`dsGradient`) или сплошная закраска одним цветом (`dsNormal`).

Таблица 3.14. *Свойства компонента* `ControlBar`

Свойство	Описание
`GradientDirection`	Направление градиента закраски: `gdVertical` — по вертикали; `gdHorizontal` — по горизонтали. Цвета градиентной закраски определяют свойства `GradientStartColor` и `GradientEndColor`

SpeedButton

Компонент `SpeedButton`, его значок (рис. 3.24) находится на вкладке **Additional**, представляет собой командную кнопку, на которой находится картинка. Обычно компоненты `SpeedButton` размещают на поверхности панели (компонента `Panel` или `TollBar`). Свойства компонента `SpeedButton` приведены в табл. 3.15.

Рис. 3.24. Значок компонента `SpeedButton`

Таблица 3.15. *Значения свойств компонента* `SpeedButton`

Свойство	Описание
`Name`	Имя (идентификатор) компонента
`Glyph`	Битовый образ, в котором находятся картинки для каждого из возможных состояний кнопки (доступна, недоступна, нажата, зафиксирована)
`NumGlyphs`	Количество картинок в битовом образе `Glyph`
`Flat`	Определяет вид кнопки (наличие границы). Если значение свойства равно `True`, то граница кнопки появляется только при позиционировании указателя мыши на кнопке
`GroupIndex`	Идентификатор группы кнопок. Кнопки, имеющие одинаковый идентификатор группы, работают подобно переключателям (`RadioButton`): нажатие одной из кнопок группы вызывает срабатывание других кнопок этой группы. Чтобы кнопку можно было зафиксировать, значение свойства `GroupIndex` не должно быть равно 0
`Down`	Идентификатор состояния кнопки. Изменить значение свойства можно, если значение свойства `GroupIndex` не равно 0
`AllowAllUp`	Свойство определяет возможность отжать кнопку. Если кнопка нажата и значение свойства равно `True`, то кнопку можно отжать
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы

Таблица 3.15 *(окончание)*

Свойство	Описание
`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
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Hint`	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, значение свойства `ShowHint` должно быть `True`)
`ShowHint`	Разрешает (`True`) или запрещает (`False`) отображение подсказки при позиционировании указателя на кнопке

Свойство `Glyph` представляет собой *битовый образ*, в котором находятся картинки для каждого из возможных состояний кнопки (доступна, недоступна, нажата, зафиксирована). Структура битового образа для компонента `SpeedButton` приведена на рис. 3.25.

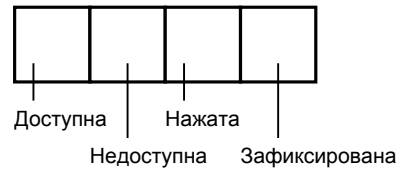


Рис. 3.25. Структура битового образа кнопки `SpeedButton`

Чтобы задать битовый образ, надо в окне **Object Inspector** выбрать свойство `Glyph`, сделать щелчок на кнопке с тремя точками, в окне **Picture Editor** щелкнуть на кнопке **Load** и в окне **Load Picture** выбрать BMP-файл, в котором находится битовый образ. В качестве примера на рис. 3.26 приведен битовый образ для кнопки **Дальше**.

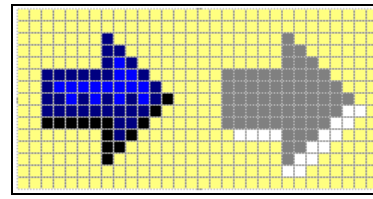


Рис. 3.26. Битовый образ для кнопки **Дальше**

Следует обратить внимание, что левый нижний пиксел картинки задает "прозрачный" цвет — элементы рисунка, окрашенные этим цветом, на поверхности кнопки не отображаются. Также необходимо отметить, что BMP-файл, из которого был загружен битовый образ во время разработки формы, во время работы программы не нужен.

На рис. 3.27 приведена форма программы "Просмотр иллюстраций", в которой для выбора папки, перехода к следующей и возврата к предыдущей иллюстрациям используются кнопки `SpeedButton`. Обратите внимание, что сначала на форму нужно поместить компонент `Panel` и настроить его (присвоить значение свойству `Align`). После этого на поверхность панели нужно поместить командные кнопки. Компонент `Image` настраивается последним. Значения свойств компонентов приведены в табл. 3.16.

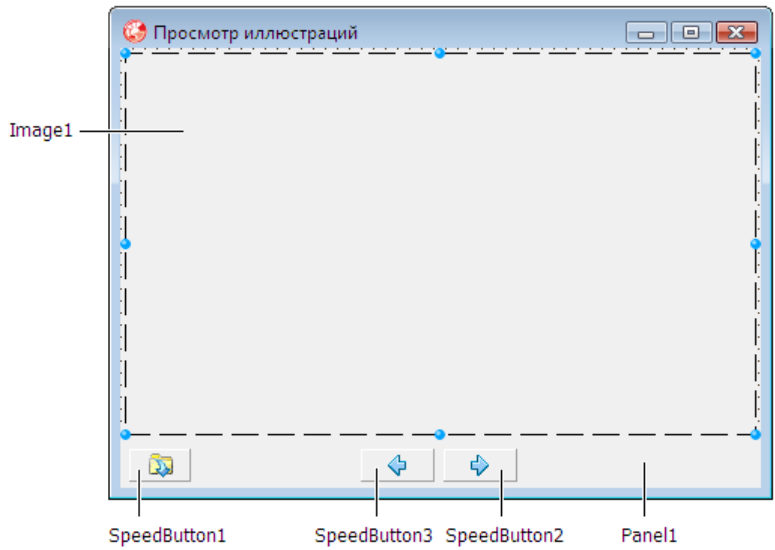


Рис. 3.27. Форма программы "Просмотр иллюстраций"

***Таблица 3.16.** Значения свойств компонентов*

Компонент	Свойство	Значение
`Panel1`	`Align`	`alBottom`
	`Height`	`32`
`SpeedButton1`	`Width`	`49`
	`Height`	`22`
	`Flat`	`True`
	`Glyph`	

Таблица 3.16 *(окончание)*

Компонент	Свойство	Значение
SpeedButton1	NumGlyphs	1
	Hint	Выбор папки
	ShowHint	True
SpeedButton2	Width	49
	Height	22
	Flat	True
	Glyph	
	NumGlyphs	2
	Hint	Следующая
	ShowHint	True
SpeedButton3	Width	49
	Height	22
	Flat	True
	Glyph	
	NumGlyphs	2
	Hint	Предыдущая
	ShowHint	True
Image1	Proportional	True
	Align	alClient
	AlignWithMargins	True
	Margins.Bottom	3
	Margins.Left	3
	Margins.Right	3
	Margins.Top	3

StatusBar

Компонент `StatusBar`, его значок (рис. 3.28) находится на вкладке **Win32**, представляет собой область вывода служебной информации (область состояния). Обычно в области состояния находится несколько панелей.

Свойства компонента `StatusBar` приведены в табл. 3.17.

Рис. 3.28. Значок компонента `StatusBar`

Таблица 3.17. *Свойства компонента* `StatusBar`

Свойство	Описание
`Panels`	Коллекция объектов типа `TStatusPanel` (см. табл. 3.18), каждый из которых представляет собой панель, отображаемую в области состояния
`SimpleText`	Текст, который отображается в поле компонента, если значение свойства `SimplePanel` равно `True`
`SimplePanel`	Тип компонента. Если значение свойства равно `True`, то в поле компонента отображается текст, заданный значением свойства `SimpleText`. Если значение свойства равно `False`, в поле компонента отображаются панели

Таблица 3.18. *Свойства объекта* `TStatusPanel`

Свойство	Описание
`Text`	Текст, отображаемый на панели
`Width`	Ширина панели

Чтобы добавить в область состояния панель, надо в окне **Object Inspector** выбрать свойство `Panels`, щелчком на кнопке с тремя точками, которая находится в области значения свойства, раскрыть окно **Editing** и в этом окне щелкнуть на кнопке **Add New** (рис. 3.29).

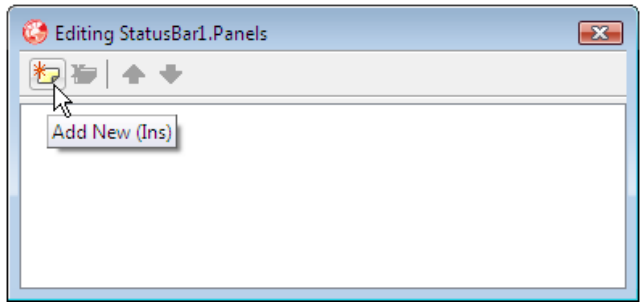


Рис. 3.29. Чтобы добавить в область состояния панель, надо сделать щелчок на кнопке **Add New**

UpDown

Компонент UpDown, его значок (рис. 3.30) находится на вкладке **Win32**, представляет собой две кнопки, нажимая которые можно изменить значение внутренней переменной-счетчика. Обычно компонент UpDown используется в связке с компонентом Edit, что позволяет ввести значение в поле редактирования обычным образом или изменить содержимое поля редактирования с помощью кнопок компонента UpDown. Свойства компонента приведены в табл. 3.20.

Рис. 3.30. Значок компонента UpDown

Таблица 3.20. *Свойства компонента* UpDown

Свойство	Описание
Position	Счетчик. Значение свойства изменяется в результате щелчка на кнопке Up (увеличивается) или Down (уменьшается). Диапазон изменения определяют свойства Min и Max, величину изменения — свойство Increment
Min	Нижняя граница диапазона изменения свойства Position
Max	Верхняя граница диапазона изменения свойства Position
Increment	Величина, на которую изменяется значение свойства Position в результате щелчка на одной из кнопок компонента
Associate	Определяет компонент (например, Edit или Label), используемый в качестве индикатора значения свойства Position. Если в качестве индикатора используется компонент Edit, то при изменении содержимого поля редактирования автоматически меняется значение свойства Position
Orientation	Задает ориентацию кнопок компонента. Кнопки могут быть ориентированы вертикально (udVertical) или горизонтально (udHorizontal)

Использование компонента UpDown демонстрирует программа "Будильник" (ее форма показана на рис. 3.31, значения свойств компонентов приведены в табл. 3.21, текст программы — в листинге 3.10). Основную работу выполняет процедура обработки события Timer, которое с периодом 1 с генерирует таймер. Процедура сравнивает текущее время (значение функции Now) со временем, на которое установлен будильник (значение переменной AlarmTime). Сравнение выполняет функция CompareTime. Чтобы эта функция, а также функции HourOf и MinuteOf были доступны, в директиву uses надо поместить ссылку на модуль DataUtils. Воспроизведение звукового сигнала обеспечивает функция PlaySound. В качестве ее первого параметра указан стандартный звуковой сигнал

(файлы, в которых находятся звуки, используемые операционной системой, размещаются в каталоге c:\Windows\Media).

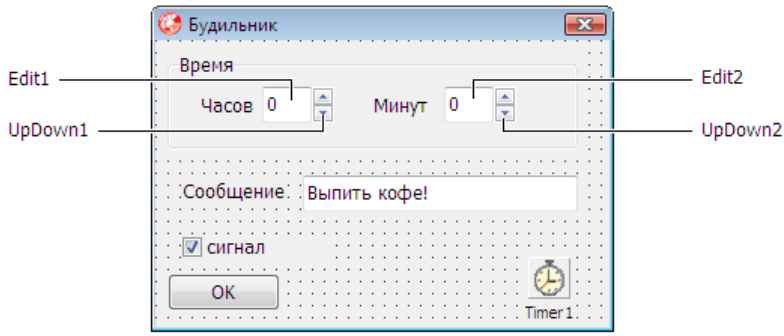


Рис. 3.31. Форма программы "Будильник"

Таблица. 3.21. *Значения свойств компонентов*

Компонент	Свойство	Значение
UpDown1	Min	0
	Max	23
	Associate	Edit1
UpDown2	Min	0
	Max	59
	Associate	Edit2

Листинг 3.10. Будильник

```
uses DateUtils, mmsystem;

var
  AlarmTime: TDateTime; // время сигнала

procedure TForm1.FormCreate(Sender: TObject);
begin
  // для доступа к MinuteOf и HourOf в директиву
  // uses надо добавить ссылку на DateUtils
  UpDown1.Position := HourOf(Now);
  UpDown2.Position := MinuteOf(Now);
end;
```

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  h,m: word; // час, минуты
begin
  Timer1.Enabled := False;
  h := StrToInt(Edit1.Text);
  m := StrToInt(Edit2.Text);
  AlarmTime := EncodeTime(h,m,0,0);

  Timer1.Enabled := True; // пуск таймера
  Form1.Hide; // свернуть окно программы
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  { Функция CompareTime позволяет сравнить два значения типа TTime:
    A      B       CompareTime(A,B)
  13:40  14:40          -1
  14:40  14:40           0
  15:40  14:40           1
  }
  if CompareTime(Now,AlarmTime) >= 0 then
    begin
        Timer1.Enabled := False;
        if CheckBox1.Checked then
            PlaySound('notify.wav',0,SND_ASYNC);

        ShowMessage(FormatDateTime(' hh:nn - ', Now) +  Edit3.Text);

        Form1.Show; // отобразить (развернуть) окно
  end;
end;

// нажатие клавиши в поле редактирования "Минут"
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    #8: ; // <BackSpace>
    '0'..'9': if Length(Edit1.Text) = 2 then Key := #0;
```

```
  else
    Key := #0;
  end;

end;

// нажатие клавиши в поле редактирования "Секунд"
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    #8: ; // <Backspace>
    '0'..'9': if Length(Edit1.Text) = 2 then Key := #0;
  else
    Key := #0;
  end;

end;
```

ProgressBar

Компонент `ProgressBar`, его значок (рис. 3.32) находится на вкладке **Win32**, представляет собой индикатор, который обычно используется для наглядного представления протекания процесса, например обработки (копирования) файлов или загрузки информации из сети. Свойства компонента приведены в табл. 3.22.

Рис. 3.32. Значок компонента `ProgressBar`

Таблица 3.22. *Свойства компонента* `ProgressBar`

Свойство	Описание
`Position`	Значение, отображаемое в поле компонента в виде прямоугольника, ширина которого пропорциональна значению свойства `Position`
`Min`	Нижняя граница диапазона допустимого значения свойства `Position`
`Max`	Верхняя граница диапазона допустимого значения свойства `Position`
`Step`	Шаг изменения значения свойства `Position`, если для изменения значения свойства `Position` используется метод `StepIt`
`Smooth`	Определяет вид индикатора. Если значение свойства равно `False`, то полоса делится на сегменты

Использование компонента `ProgressBar` демонстрирует программа "Угадай число" (ее форма приведена на рис. 3.33, а текст — в листинге 3.11). Компонент применяется в качестве индикатора времени, оставшегося на решение поставленной задачи. Следует обратить внимание, что размер компонента подобран так, чтобы в поле компонента отображалось 30 сегментов. Значения свойств компонента `ProgressBar` приведены в табл. 3.23.

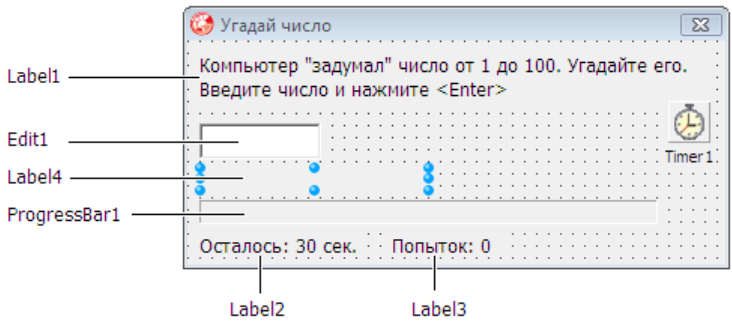


Рис. 3.33. Форма программы "Угадай число"

Таблица 3.23. *Значения свойств компонента* `ProgressBar`

Свойство	Значение
Mim	0
Max	30
Position	30
Step	-1
Smoth	False

Листинг 3.11. Угадай число

```
// начало работы программы
procedure TForm1.FormCreate(Sender: TObject);
begin
  Randomize;
  sn := Random(100) + 1;
  rem := 30;
  Label2.Caption := 'Осталось: 30 сек';
  Timer1.Interval := 1000;
  Timer1.Enabled := True;
end;
```

```
// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    rem := rem - 1;
    ProgressBar1.StepIt; // изменить значение Position
    Label2.Caption := 'Осталось: ' + IntToStr(rem) + ' сек';
    if rem = 0 then
    begin
        Timer1.Enabled := False;
        Edit1.Enabled := False;
        ShowMessage('Вы не справились с поставленной задачей.' + #10 +
                    'Компьютер "задумал" число ' + IntToStr(sn));
    end;
end;

// нажатие клавиши в поле редактирования
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
  igrok: integer; // вариант игрока
begin

  Label4.Caption := '';
  case Key of
    '0'..'9': if Length(Edit1.Text) = 3 then Key := #0; // цифры
    #8: ;   // <Backspace>
    #13:    // <Enter>
         begin
            p := p + 1;
            igrok := StrToInt(Edit1.Text);
            Label3.Caption := 'Попыток: ' + IntToStr(p);
            if igrok = sn then
              begin
                //  число угадано
                Timer1.Enabled := False;
                ShowMessage('Вы справились с поставленной задачей' +
                            #10 + 'за ' + IntToStr(30 - rem) + 'сек.');
                Edit1.Enabled := False;
              end
              else begin
                 if igrok < sn then
                      Label4.Caption := 'больше'
```

```
                else
                    Label4.Caption := 'меньше';
            end;
        end;
    else Key := #0;
  end;
end;
```

Image

Компонент `Image`, его значок (рис. 3.34) находится на вкладке **Additional**, обеспечивает отображение графики (иллюстраций, фотографий, рисунков). Свойства компонента приведены в табл. 3.24.

Рис. 3.34. Значок компонента `Image`

Таблица 3.24. *Свойства компонента* `Image`

Свойство	Описание
`Picture`	Иллюстрация, которая отображается в поле компонента
`Width`, `Height`	Размер компонента. Если размер компонента меньше размера иллюстрации, а значения свойств `AutoSize`, `Stretch` и `Proportional` равны `False`, то отображается часть иллюстрации
`AutoSize`	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
`Stretch`	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена
`Proportional`	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение данного свойства должно быть `True`, а свойства `AutoSize` — `False`
`Center`	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно `True`, то картинка располагается в центре поля компонента
`Align`	Задает границу формы, к которой "привязан" компонент. Если значение свойства равно `alClient`, то размер компонента устанавливается равным размеру "клиентской" (внутренней) области формы, причем, если во время работы программы будет изменен размер формы, автоматически будет изменен и размер компонента

Таблица 3.24 *(окончание)*

Свойство	Описание
`AlignWithMargins`	Признак того, что для вычисления положения компонента на поверхности объекта, к которому компонент "привязан", следует использовать значения, заданные свойством `Margins`. Позволяет установить отступ от границы объекта, к которой компонент привязан
`Margins`	Задает расстояния от границы компонента до соответствующей границы объекта (формы), к которой компонент привязан
`Canvas`	Поверхность компонента

Картинку, отображаемую в поле компонента `Image`, можно задать как во время разработки формы, так и загрузить из файла во время работы программы. Если картинка задана во время разработки формы, то файл, из которого она была загружена, во время работы программы не нужен (копия картинки помещается в файл ресурса программы). Загрузку картинки из файла во время работы программы обеспечивает метод `LoadFromFile` свойства `Picture`. Необходимо отметить: для того чтобы во время работы программы в поле компонента можно было загрузить иллюстрацию из JPEG-файла, в директиву `uses` модуля формы надо добавить ссылку на модуль `Jpeg`.

Использование компонента `Image` для просмотра фотографий демонстрирует программа "Просмотр иллюстраций" (ее форма приведена на рис. 3.35).

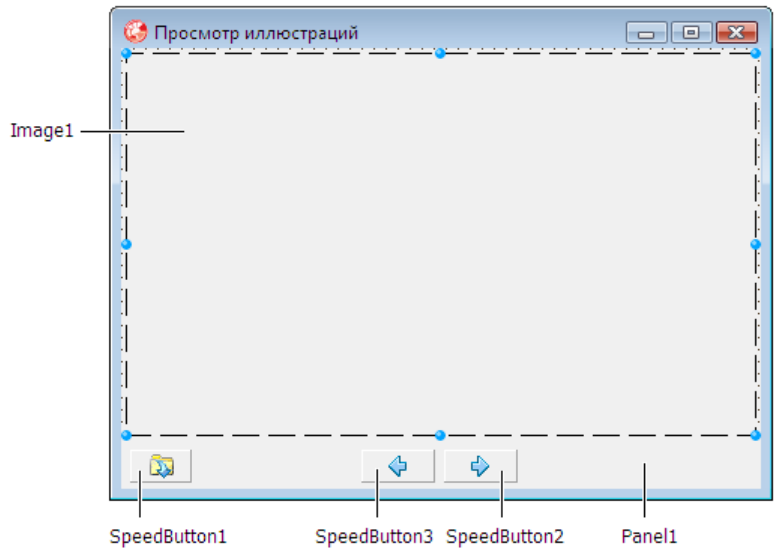


Рис. 3.35. Форма программы "Просмотр иллюстраций"

Необходимо отметить, что во время создания формы сначала нужно настроить компонент `Panel` *(см. разд. "Panel" ранее в этой главе)* (присвоить значение `alBottom` свойству `Align`), а затем компонент `Image` (значения свойств приведены в табл. 3.25).

В листинге 3.12 приведены процедуры обработки событий `Click` на кнопках `SpeedButton`. Процедура обработки события `Click` на `SpeedButton1` отображает стандартное окно **Выбор папки** и формирует список иллюстраций. Для хранения списка иллюстраций в программе используется список строк `Pictures` (объект типа `TStringList`). Его объявление надо поместить в секцию `private` объявления формы. Создает объект `Pictures` конструктор формы — процедура обработки события `Create`.

Таблица 3.25. *Значения свойств компонента* `Image`

Свойство	Значение
`Align`	`alClient`
`AlignWithMargins`	`True`
`Margins.Bottom`	`3`
`Margins.Left`	`3`
`Margins.Right`	`3`
`Margins.Top`	`3`

Листинг 3.12. Просмотр иллюстраций

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, FileCtrl, Jpeg;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    SpeedButton2: TSpeedButton;
    SpeedButton1: TSpeedButton;
    SpeedButton3: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
```

```
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    aPath: string; // каталог, который выбрал пользователь
    aSearchRec : TSearchRec; // информация о файле
    Pictures : TStringList;  // список иллюстраций
    n: integer;   // номер отображаемой иллюстрации

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  Pictures := TStringList.Create;
  SpeedButton2.Enabled := False;
  SpeedButton3.Enabled := False;
end;

// щелчок на кнопке "Папка"
procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  r: integer;
begin
  if SelectDirectory('Выберите папку', '',aPath) then
  begin
    aPath := aPath + '\';
    Form1.Caption := 'Просмотр иллюстраций - ' + aPath;

    // сформировать список иллюстраций
    r := FindFirst(aPath+'*.jpg',faAnyFile,aSearchRec);
    if r = 0 then
```

```
      begin
        // в указанном каталоге есть JPEG-файл
        Pictures.Clear; // очистить список иллюстраций
        Pictures.Add(aSearchRec.Name); // добавить имя файла
                                       // в список иллюстраций

        // получить имена остальных JPEG-файлов
        repeat
           r := FindNext(aSearchRec); // получить имя следующего файла
           if r = 0 then
              Pictures.Add(aSearchRec.Name);
        until (r <> 0);

        if Pictures.Count > 1 then
             SpeedButton2.Enabled := True;

        // отобразить иллюстрацию
        n := 0; // номер отображаемой иллюстрации
        try
           Form1.Image1.Picture.LoadFromFile(aPath + Pictures[n]);
           Form1.Caption := aPath + Pictures[n];
           except on EInvalidGraphic
               do Form1.Image1.Picture.Graphic := nil;
        end;
        if Pictures.Count = 1 then
          SpeedButton2.Enabled := False;
      end
      else begin
         // в выбранном каталоге нет JPEG-файлов
         SpeedButton2.Enabled := False;
         SpeedButton3.Enabled := False;
         Form1.Image1.Picture.Graphic := nil;
    end;
  end;
end;

// вывод следующей картинки
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
   // вывести картинку
   n := n+1;
```

```
   try
      Form1.Image1.Picture.LoadFromFile(aPath + Pictures[n]);
      Form1.Caption := aPath + Pictures[n];
      except on EInvalidGraphic do
            Form1.Image1.Picture.Graphic := nil;
   end;

   if n = Pictures.Count-1 then
      SpeedButton2.Enabled := False;

   // если кнопка "Предыдущая" не доступна, сделать ее доступной
   if (n > 0) and SpeedButton3.Enabled = False then
        SpeedButton3.Enabled := True;
end;

// вывод предыдущей картинки
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
   // вывести картинку
   n := n-1;
   try
       Form1.Image1.Picture.LoadFromFile(aPath + Pictures[n]);
       Form1.Caption := aPath + Pictures[n];
       except on EInvalidGraphic do
            Form1.Image1.Picture.Graphic := nil;
   end;

   if n = 0 then
      SpeedButton3.Enabled := False;

   // если кнопка "Следующая" не доступна,
   // сделать ее доступной
   if (n < Pictures.Count) and SpeedButton2.Enabled = False then
        SpeedButton2.Enabled := True;
end;

// изменился размер окна
procedure TForm1.FormResize(Sender: TObject);
begin
```

```
  // изменить положение командных кнопок
  // кнопка "Назад"
  SpeedButton3.Left := Round(Panel1.Width/2)-SpeedButton3.Width - 5;

  // кнопка "Вперед"
  SpeedButton2.Left := Round(Panel1.Width/2) + 5;
end;

end.
```

MainMenu

Компонент `MainMenu`, его значок (рис. 3.36) находится на вкладке **Standard**, представляет собой строку главного меню.

Рис. 3.36. Значок компонента `MainMenu`

После того как значок компонента будет помещен на форму, его нужно настроить. Сначала надо определить структуру меню. Для этого необходимо двойным щелчком на значке компонента раскрыть окно редактора меню.

В начале работы над новым меню в окне редактора меню находится один-единственный прямоугольник, который изображает новый элемент меню (свойства элемента меню отображаются в окне **Object Inspector**). Сначала в поле значения свойства `Caption` нужно ввести название меню, например `Файл`, и нажать клавишу <Enter>. В результате чего в меню будет добавлен элемент (создан объект типа `TMenuItem`), а в окне редактора меню появятся два прямоугольника: снизу и справа от выбранного элемента меню. Следует обратить внимание, что по умолчанию редактор меню присваивает каждому созданному элементу меню имя, которое состоит из буквы `N` и порядкового номера элемента. Так, первый элемент меню получает имя `N1`, второй — `N2` и т. д.

Чтобы добавить в созданное меню команду, надо выбрать прямоугольник, который находится снизу, и в поле значения свойства `Caption` ввести название команды, например `Открыть`. Чтобы добавить раздел меню, надо выбрать тот прямоугольник, который находится справа, и в поле значения свойства `Caption` ввести название раздела меню, например `Справка`.

В качестве примера на рис. 3.37 приведено окно редактора меню, в котором отображается меню программы MEdit.

После того как структура меню будет определена, можно выполнить его окончательную настройку.

Каждый элемент меню представляет собой объект типа `TMenuItem` (свойства объекта приведены в табл. 3.26).

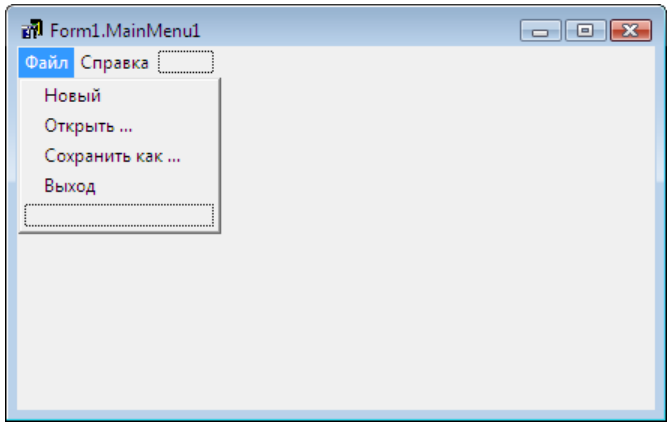


Рис. 3.37. Окно редактора меню (настройка компонента `MainMenu`)

Таблица 3.26. *Свойства объекта* `TMenuItem`

Свойство	Описание
`Name`	Идентификатор элемента меню
`Caption`	Название элемента меню или команды
`Bitmap`	Картинка, которая отображается слева от названия элемента меню
`Enabled`	Признак доступности элемента меню (`True` — элемент доступен, `False` — недоступен)
`ShortCut`	Функциональная клавиша или комбинация клавиш, например <Ctrl>+<Z>, с помощью которой можно быстро выбрать элемент меню

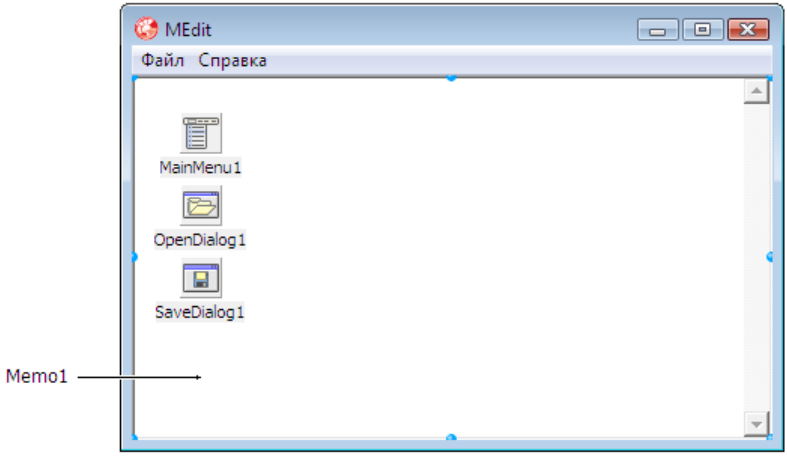


Рис. 3.38. Форма программы Medit

Использование компонента MainMenu демонстрирует программа MEdit (ее форма показана на рис. 3.38, а значения свойств элементов меню приведены в табл. 3.27). После того как меню будет создано и настроено, надо для каждого элемента меню задать действие, которое должно быть выполнено в результате выбора этого элемента меню. Чтобы это сделать, надо в окне **Object Inspector** выбрать объект MenuItem, соответствующий нужному элементу меню, и далее обычным образом создать процедуру обработки события Click. Процедуры обработки событий приведены в листинге 3.13.

Таблица 3.27. *Значения свойств компонентов*

Компонент (объект)	Свойство	Значение
N1	Name Caption	N1 Файл
N2	Name Caption Bitmap	N2 Новый 
N3	Name Caption Bitmap	N3 Открыть... 
N4	Name Caption Bitmap	N4 Сохранить как... 
N5	Name Caption	N5 Выход
N6	Name Caption	N6 Справка
N7	Name Caption Enabled	N7 О программе False

Листинг 3.13. MEdit — простой редактор текста

```
// команда Файл >> Новый
procedure TForm1.N2Click(Sender: TObject);
var
  r: integer; // идентификатор кнопки
```

```
begin
  if Memo1.Modified  then
  begin
      r := MessageDlg('Текст был изменен. Создать новый документ' +
                      #10 +'без сохранения изменений в текущем?',
                       mtWarning,[mbYes,mbNo],0,mbNo);
      if r = mrYes then
          Memo1.Clear;
  end
  else Memo1.Clear;
end;

// команда Файл >> Открыть
procedure TForm1.N3Click(Sender: TObject);
var
  r: integer; // идентификатор кнопки
begin
  if Memo1.Modified  then
  begin
      r := MessageDlg('Текст был изменен. Открыть новый файл' +
                       #10 + 'без сохранения изменений?',
                       mtWarning,[mbYes,mbNo],0,mbNo);
      if r = mrNo then
          exit; // продолжить работу
  end;

  // отобразить окно "Открыть"
  if OpenDialog1.Execute then
  begin
      // пользователь выбрал файл
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
      FileName := OpenDialog1.FileName;
      Form1.Caption := 'MEdit - ' + FileName;
  end;
end;

// команда Файл >> Сохранить как
procedure TForm1.N4Click(Sender: TObject);
begin
  SaveDialog1.FileName := FileName;
```

```
  if SaveDialog1.Execute then
      // сохранить текст в файле
      Memo1.Lines.SaveToFile(FileName);
end;

// команда Файл >> Выход
procedure TForm1.N5Click(Sender: TObject);
var
  r: integer; // идентификатор кнопки
begin
  if Memo1.Modified  then
  begin
      r := MessageDlg('Текст был изменен. Завершить работу' + #10 +
                      'программы без сохранения изменений?',
                       mtWarning,[mbYes,mbNo],0,mbNo);
      if r = mrYes then
          Close; // завершить работу программы
  end;
end;

// щелчок на кнопке "Закрыть"
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var
  r: integer; // идентификатор кнопки
begin
  if Memo1.Modified  then
  begin
      r := MessageDlg('Текст был изменен. Завершить работу' + #10 +
                      'программы без сохранения изменений?',
                       mtWarning,[mbYes,mbNo],0,mbNo);
      if r = mrNo then
          CanClose := False; // не завершать работу программы
  end;
end;
```

OpenDialog

Компонент `OpenDialog`, его значок (рис. 3.39) находится на вкладке **Dialogs**, представляет собой диалог **Открыть**. Свойства компонента приведены в табл. 3.28. Отображение диалога обеспечивает метод `Execute`, значение которого

позволяет определить, щелчком на какой кнопке, **Открыть** или **Отмена**, пользователь закрыл диалог.

Рис. 3.39. Значок компонента `OpenDialog`

***Таблица. 3.28.** Свойства компонента `OpenDialog`*

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Открыть**
`Filter`	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют выбранному (текущему) фильтру. Во время отображения диалога пользователь может выбрать фильтр в списке **Тип файлов**. Каждый фильтр задается строкой вида *`описание|маска`*, например `Текст|*.txt`
`FilterIndex`	Если в списке `Filter` несколько элементов (например, `Текст|*.txt|Все файлы|*.*`), то значение свойства задает фильтр, который используется в момент появления диалога на экране
`InitialDir`	Каталог, содержимое которого отображается при появлении диалога на экране. Если значение свойства не указано, то в окне диалога отображается содержимое папки Мои документы
`FileNane`	Имя файла, выбранного пользователем

Использование компонента `OpenDialog` для выбора файла, содержимое которого надо отобразить в поле компонента `Memo`, демонстрирует программа "Просмотр текста" (ее форма приведена на рис. 3.40). В меню **Файл** находятся два элемента: **Открыть** (идентификатор `N2`) и **Выход** (идентификатор `N3`). Отображение окна **Открыть файл** и загрузку выбранного файла обеспечивает процедура обработки события `Click` на элементе меню `N2` (листинг 3.14). Значения свойств компонентов программы приведены в табл. 3.29.

***Таблица 3.29.** Значения свойств компонентов*

Компонент	Свойство	Значение
`OpenDialog1`	`Title`	`Открыть файл`
	`Filter`	`Текст|*.txt |Все файлы|*.*`
`Memo1`	`Align`	`alClient`
	`ReadOnly`	`True`

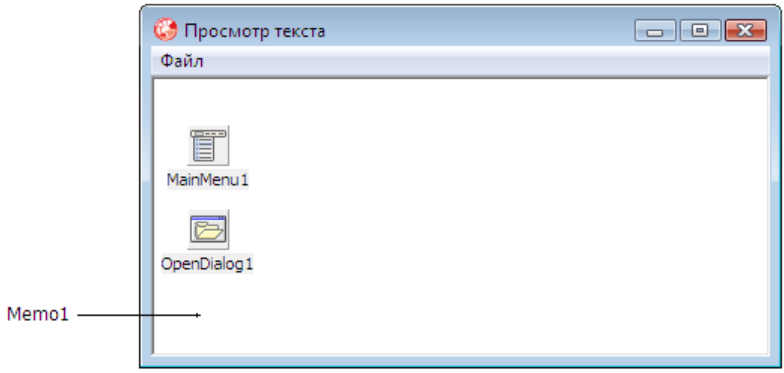


Рис. 3.40. Форма программы "Просмотр текста"

Листинг. 3.14. Просмотр текста

```
// команда Файл >> Открыть
procedure TForm1.N2Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    // пользователь выбрал файл
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;

// команда Файл >> Выход
procedure TForm1.N3Click(Sender: TObject);
begin
  Close;
end;
```

SaveDialog

Компонент `SaveDialog`, его значок (рис. 3.41) находится на вкладке **Dialogs**, представляет собой диалог **Сохранить**. Свойства компонента приведены в табл. 3.30. Отображение диалога обеспечивает метод `Execute`, значение которого позволяет определить, щелчком на какой кнопке, **Сохранить** или **Отмена**, пользователь закрыл диалог.

Рис. 3.41. Значок компонента `SaveDialog`

Таблица. 3.30. *Свойства компонента* `SaveDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Сохранить как**
`Filter`	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют выбранному (текущему) фильтру. Во время отображения диалога пользователь может выбрать фильтр в списке **Тип файлов**. Каждый фильтр задается строкой вида *описание*`|`*маска*, например `Текст|*.txt`
`FilterIndex`	Если в списке `Filter` несколько элементов (например, `Текст|*.txt|Все файлы|*.*`), то значение свойства задает фильтр, который используется в момент появления диалога на экране
`InitialDir`	Каталог, содержимое которого отображается при появлении диалога на экране. Если значение свойства не указано, то в окне диалога отображается содержимое папки Мои документы
`FileNane`	Имя файла, введенное пользователем в поле **Имя файла**
`DefaultExt`	Расширение, которое будет добавлено к имени файла, если в поле **Имя файла** пользователь не задаст расширение файла

В качестве примера использования диалога `SaveDialog` в листинге 3.15 приведена процедура обработки события `CloseQuery` главного окна простого редактора текста (форма представлена на рис. 3.42, значения свойств компонента `SaveDialog` — в табл. 3.31).

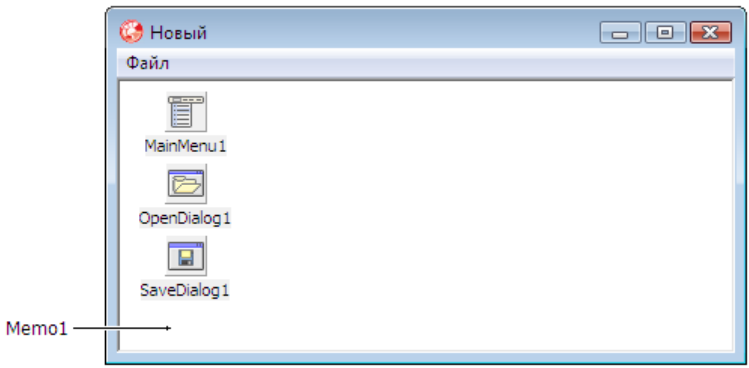


Рис. 3.42. Форма простого редактора текста

Таблица 3.31. *Значения свойств компонента* `SaveDialog`

Свойство	Значение
`DefaultExt`	`txt`
`Filter`	`Текст|*.txt`

Листинг 3.15. Обработка события `CloseQuery`

```
// запрос на завершение работы с программой (попытка закрыть окно)
procedure TForm1.FormCloseQuery(Sender: TObject;
                                var CanClose: Boolean);
var
  r: integer; // идентификатор кнопки, нажатой пользователем
              // в окне MessageDlg
begin
  if Memo1.Modified  then
  begin
     r := MessageDlg('Текст изменен. Сохранить изменения?',
                     mtWarning,[mbYes,mbNo,mbCancel],0);
     case r of
       mrYes:    // записать текст в файл
          begin
            if OpenDialog1.FileName <> '' then
                // имя файла задано (редактируется загруженный файл)
                Memo1.Lines.SaveToFile(OpenDialog1.FileName)
            else begin
                  // имя файла не задано, отобразить SaveDialog
                  if SaveDialog1.Execute then
                        // пользователь задал имя файла
                        Memo1.Lines.SaveToFile(SaveDialog1.FileName)
                  else
                       // не задано имя файла,
                       // продолжить работу с программой
                       CanClose := False;
            end;
          end;
       mrCancel: // продолжить работу с программой
                 CanClose := False;
     end;
  end;
end;
```

Компоненты Vista

Одной из отличительных черт Windows Vista и, соответственно, Windows 7 от предыдущих версий Windows является новый дизайн и существенно расширенная

функциональность диалоговых окон. Окна диалогов **Открыть** и **Сохранить**, а также окна информационных сообщений в Windows Vista выглядят несколько иначе, чем в предыдущих версиях Windows.

Для того чтобы в полной мере использовать возможности Vista, чтобы диалоги **Открыть**, **Сохранить** и окна сообщений в программе, работающей в Windows Vista, отображались в стиле Vista, вместо компонентов `OpenDialog`, `SaveDialog` и функции `MessageDlg` следует использовать компоненты `FileOpenDialog`, `FileSaveDialog` и `TaskDialogs` (рис. 3.43).

Рис. 3.43. Компоненты **Vista Dialogs**

Следует обратить внимание, что отображение диалогов обеспечивает операционная система. Поэтому при попытке отобразить Vista-диалог в программе, работающей в Windows предыдущей версии, например в Windows XP или Windows 2000, возникает ошибка. Чтобы предотвратить возникновение ошибки при попытке отобразить Vista-диалог в случае, если программа запущена не из Vista, в код необходимо добавить инструкции, которые будут проверять версию операционной системы и активизировать отображение диалога, соответствующего версии операционной системы.

TaskDialog

Компонент `TaskDialog` предназначен для отображения окон сообщений (рис. 3.44). Помимо текста и стандартных командных кнопок (**Да**, **Нет**, **Отменить** и др.) в окне диалога могут отображаться определенные программистом кнопки, переключатели и другие элементы управления. Кроме этого в тексте сообщений могут быть ссылки, например на разделы справочной информации.

Свойства компонента `TaskDialog` приведены в табл. 3.32.

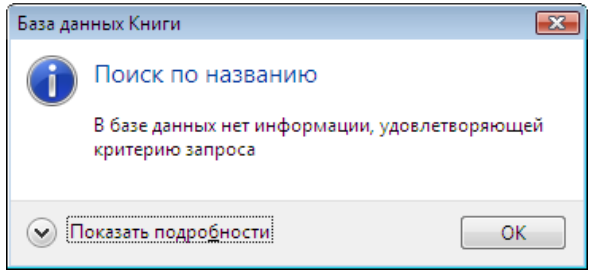


Рис. 3.44. Пример сообщения

***Таблица 3.32.** Свойства компонента* `TaskDialog`

Свойство	Описание
`Caption`	Текст в заголовке окна сообщения (обычно название программы, которая вывела окно сообщения)
`Title`	Заголовок сообщения
`MainIcon`	Значок, отображаемый слева от заголовка сообщения. Обычно это один из стандартных значков: **Information** (`tdiInformation`), **Warning** (`tdiWarning`) или **Error** (`tdiError`). Если значение свойства равно `tdiNone`, то отображается значок, заданный значением свойства `CustemMainIcon`
`CustomMainIcon`	Значок, отображаемый слева от заголовка сообщения. Появляется, если значение свойства `MainIcon` равно `tdiNone`
`Text`	Основной текст сообщения
`CommonButtons`	Стандартные кнопки, отображаемые в окне диалога. Уточняющие свойства `tcbOk`, `tcbYes`, `tcbNo`, `tcbCancel`, `tcbRetry` и `tcbClose` определяют, какие кнопки должны отображаться
`ExpandedText`	Поясняющий текст. Отображается в результате щелчка на кнопке `ExpandButton`
`ExpandButtonCaption`	Текст, отображаемый рядом с кнопкой `ExpandButton` (по умолчанию — **Дополнительно**)
`FooterText`	Дополнительный текст, отображаемый в нижней части окна диалога (нижний колонтитул). Если значение свойства задано, то перед текстом отображается компонент `CheckBox`
`FooterIcon`	Значок, который отображается в области вывода нижнего колонтитула
`Buttons`	Дополнительные, определенные программистом, командные кнопки
`RadioButtons`	Дополнительные, определенные программистом, переключатели
`ProgressBar`	Индикатор протекания процесса
`ProgressBar.Position`	Значение, отображаемое в поле индикатора в виде закрашенной области
`ProgressBar.Min`	Нижняя граница области допустимого значения свойства `Position`

Таблица 3.32 *(продолжение)*

Свойство	Описание
`ProgressBar.Max`	Верхняя граница области допустимого значения свойства `Position`
`ProgressBar.State`	Характеристика процесса, для отображения которого используется индикатор (`pbsNormal` — нормальное протекание, `pbsPaused` — процесс приостановлен, `pbsError` — процесс завершен с ошибкой)
`ProgressBar.MarqueeSpeed`	Скорость перемещения маркера внутри полосы (закрашенной области) индикатора
`Flags.tfEnableHyperlinks`	Разрешает/запрещает использование HTML-тега `<A>` в тексте сообщений (основного, пояснительного, дополнительного). Если значение свойства равно `True`, то вместо текста тега отображается соответствующая ссылка
`Flags.tfUseHiconMain`	Разрешает/запрещает отображение значка, заданного значением свойства `MainIcon`. Если значение свойства равно `True`, то перед текстом заголовка отображается значок, заданный свойством `MainIcon`, в противном случае — свойством `CustomMainIcon`
`Flags.tfExpandFooterArea`	Управляет отображением области нижнего колонтитула. Если значение свойства равно `False`, то область нижнего колонтитула не отображается
`Flags.tfUseHiconFooter`	Разрешает/запрещает отображение значка, заданного значением свойства `FooterIcon`. Если значение свойства равно `True`, то в области нижнего колонтитула, перед компонентом `CheckBox` отображается значок, заданный свойством `FooterIcon`, в противном случае значок не отображается
`Flags.tfVerificationFlagChecked`	Определяет состояние компонента `CheckBox`, отображаемого в области нижнего колонтитула
`Flags.tfShowProgressBar`	Разрешает/запрещает отображение индикатора состояния процесса. Если значение свойства равно `True`, то индикатор отображается, в противном случае — нет
`Flags.tfShowMarqueeProgressBar`	Разрешает/запрещает отображение маркера на индикаторе состояния процесса. Если значение свойства равно `True`, то маркер отображается, в противном случае — нет

Таблица 3.32 *(окончание)*

Свойство	Описание
`Flags.tfNoDefaultRadioButton`	Управляет отображением состояния переключателей. Если значение свойства равно `True`, то при появлении диалога ни один из переключателей не является выбранным (пользователь должен сделать выбор)
`ModalDesult`	Идентификатор командной кнопки, щелчком на которой пользователь закрыл окно диалога: `mrOk`, `mrYes`, `mrNo`, `mrCancel`, `mrRetry` или `mrClose`. Если окно закрыто в результате щелчка на определенной пользователем командной кнопке, то по умолчанию значение свойства для первой кнопки равно 100, для второй — 101 и т. д.
`RadioButton.ID`	Идентификатор переключателя, выбранного пользователем

Значения свойств компонента `TaskDialog` устанавливаются в окне **Object Inspector** обычным образом. Чтобы увидеть, как будет выглядеть диалог во время работы программы, надо сделать двойной щелчок левой кнопкой мыши на находящемся на форме значке компонента или выбрать из контекстного меню компонента команду **Test Dialog**.

Отображение диалога `TaskDialog` обеспечивает метод-функция `Execute`. Этот метод возвращает значение `False`, если пользователь завершил диалог щелчком на кнопке **Close** или **Закрыть**. Если окно закрыто щелчком на какой-либо другой кнопке, значение метода равно `True`. При этом свойство `ModalResult` содержит идентификатор нажатой кнопки.

Как было сказано ранее, при попытке отобразить диалог `TaskDialog` не в Windows Vista возникает исключение. Поэтому, перед тем как активизировать процесс отображения диалога, необходимо проверить версию операционной системы. Код, позволяющий проверить версии ОС:

```
// определить версию ОС
OSVersionInfo.dwOSVersionInfoSize := sizeof(TOSVersionInfo);
GetVersionEx(OSVersionInfo);
if OSVersionInfo.dwMajorVersion >= 6 then
   begin
   //  ОС — Vista

   end
else
   begin
    // не Vista
   end;
```

Использование диалога `TaskDialog` для вывода сообщения о неверных данных демонстрирует программа "Доход по вкладу" (ее окно показано на рис. 3.45, значения свойств компонента `TaskDialog` — в табл. 3.33, процедура обработки события `Click` на кнопке **Ok** — в листинге 3.16). Следует обратить внимание, что программа спроектирована так, что она будет работать как в Windows Vista, так и в Windows предыдущих версий. В Vista вывод сообщений обеспечивает диалог `TaskDialog`, в ОС предыдущих версий — функция `MessageDlg`. Пример сообщения приведен на рис. 3.46.

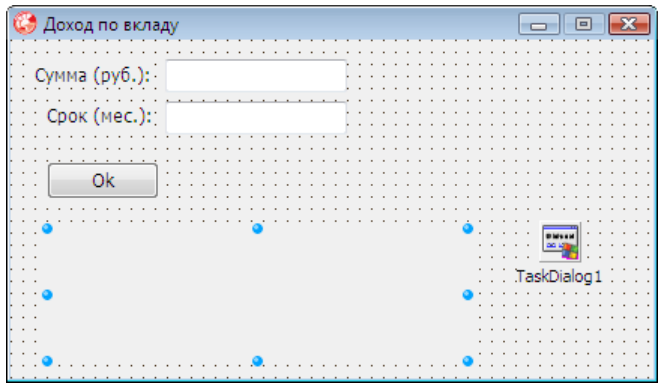


Рис. 3.45. Форма программы "Доход по вкладу"

Таблица 3.33. *Значения свойств компонента* `TaskDialog`

Свойство	Значение
Caption	Доход по вкладу
Title	Ошибка
MainIcon	tdiError
Text	Неверные исходные данные
CommonButtons	tcbOk

Листинг 3.16. Щелчок на кнопке *Ok*

```
// щелчок на кнопке Ok
procedure TForm1.Button1Click(Sender: TObject);
var
  summ: real;       // сумма вклада
  period: integer;  // срок вклада (дней)
  percent: real;    // процентная ставка (годовых)
```

```
  profit: real;     // доход
  total: real;      // сумма в конце срока вклада
  ermsg: string;    // сообщение об ошибке

  OSVersionInfo: TOSVersionInfo;  // версия ОС

begin
   summ := StrToFloat(Edit1.Text);
   period := StrToInt(Edit2.Text);

   // проверить исходные данные
   ermsg := '';
   if summ < 5000 then
      ermsg := 'Сумма вклада должна быть не менее 5000 руб.' + #10;
   if (period < 6) or (period > 24) then
      ermsg := ermsg + 'Срок вклада должен быть от 6 до 24 мес.';

   if ermsg <> '' then
   begin
        Label3.Caption := '';
        // определить версию ОС
        OSVersionInfo.dwOSVersionInfoSize := sizeof(TOSVersionInfo);
        GetVersionEx(OSVersionInfo);
        if OSVersionInfo.dwMajorVersion >= 6  then
           begin
              //  ОС — Vista
              TaskDialog1.ExpandedText := ermsg;
              TaskDialog1.Execute;
           end
        else
            // не Vista
            MessageDlg(ermsg,mtError,[mbOk],0);
   end
   else  begin
        // определить процентную ставку
        case period of
          1..3:   percent := 9.5;
          4..6:   percent := 11;
          7..12:  percent := 12.5;
          else
                  percent:=14.5;
        end;
```

```
        profit := summ * (percent/100/ 12) * period;
        total := summ + profit;
        Label3.Caption :=
             'Сумма: ' +  FloatToStrF(summ,ffCurrency,6,2) + #10 +
             'Срок: ' + IntToStr(period) + ' мес.' + #10 +
             'Процентная ставка: ' + FloatToStrF(percent,ffgeneral,6,2) +
              '%' + #10 +
             '-------------------------------------------------' + #10 +
             'Доход: ' + FloatToStrF(profit,ffCurrency,6,2) + #10 +
             'Сумма в конце срока: ' + FloatToStrF(total,ffCurrency,6,2);
    end;
end;
```

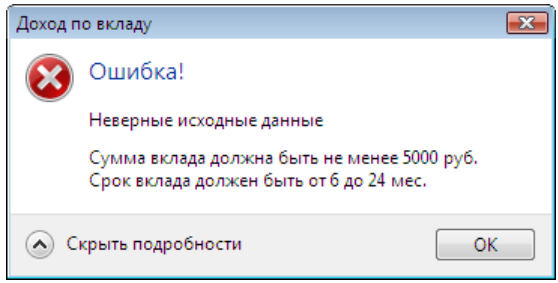


Рис. 3.46. Пример сообщения об ошибке

FileOpenDialog и FileSaveDialog

Компоненты `FileOpenDialog` и `FileSaveDialog` обеспечивают отображение диалогов **Открыть** и **Сохранить** соответственно. Свойства компонентов приведены в табл. 3.34 и 3.35.

***Таблица 3.34.** Свойства компонента `FileOpenDialog`*

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не задано, то в заголовке отображается текст **Открыть**
`DefaultFolder`	Папка, содержимое которой отображается в окне диалога при появлении его на экране. Если значение свойства не установлено, то в окне диалога отображается содержимое папки Мои документы
`DefaultExtension`	Расширение, которое добавляется к имени файла, введенному в поле **Имя файла**, в случае если пользователь явно не указал расширение

Таблица 3.34 *(окончание)*

Свойство	Описание
FileTypes	Свойство задает список фильтров имен файлов. В окне диалога отображаются только те файлы, имена которых соответствуют текущему фильтру. Список формируется путем добавления элементов в коллекцию FileTypes с последующей установкой значений свойств DisplayName и FileMask. Свойство DisplayName задает имя фильтра (например, Все файлы), а свойство FileMask — фильтр (например, *.*)
FileTypeIndex	Задает текущий фильтр
FileNane	Имя выбранного файла
FileNameLabel	Текст, который отображается перед полем ввода. Если значение свойства не задано, то перед полем ввода отображается текст **Имя файла**
OkButtonLabel	Текст, который отображается на кнопке **OK**. Если значение свойства не задано, то на кнопке отображается текст **Открыть**
FavoriteLinks	Список имен каталогов, который добавляется в список **Избранные ссылки**, отображаемый в окне диалога
Options	Свойство представляет собой совокупность уточняющих свойств и позволяет выполнить "тонкую" настройку диалога
fdoNoChangeDir	Запрещает (True) сменить текущий (открыть другой) каталог
fdoFileMustExist	Активизирует (True) режим проверки существования файла, имя которого пользователь ввел в поле редактирования
fdoPathMustExist	Активизирует (True) режим проверки существования каталога, имя которого пользователь ввел в поле редактирования
fdoNotAddToRecent	Управляет записью в список **Недавние документы** имен файлов, выбранных пользователем в окне диалога. Если значение свойства True, то имя файла в списке **Недавние документы** не фиксируется
fdoHiddePinnedPlaces	Управляет отображением избранных ссылок. Если значение свойства True, то отображаются только ссылки, определенные программистом (свойство FavoriteLinks), другие ссылки, например **Документы**, **Рабочий стол** и пр., не отображаются

Таблица 3.35. *Свойства компонента* `FileSaveDialog`

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не задано, то отображается текст **Сохранить как**

Таблица 3.35 *(продолжение)*

Свойство	Описание
FileNameLabel	Текст, который отображается перед полем ввода имени файла. Если значение свойства не задано, то перед полем ввода имени файла отображается текст **Имя файла**
FileNane	Имя файла, введенное пользователем в поле редактирования **Имя файла**
OkButtonLabel	Текст, который отображается на кнопке **OK**. Если значение свойства не задано, то на кнопке отображается текст **Сохранить**
DefaultFolder	Папка, содержимое которой отображается в окне диалога при появлении его на экране. Если значение свойства не установлено, то в окне диалога отображается содержимое папки Мои документы
DefaultExtension	Расширение, которое добавляется к имени файла, введенному в поле **Имя файла**, в случае если пользователь явно не указал расширение
FileTypes	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют текущему фильтру. Список формируется путем добавления элементов в коллекцию FileTypes с последующей установкой значений свойств DisplayName и FileMask. Свойство DisplayName задает имя фильтра (например, Все файлы), а свойство FileMask — фильтр (например, *.*)
FileTypeIndex	Задает текущий фильтр
FavoriteLinks	Список имен каталогов, который добавляется в список **Избранные ссылки**, отображаемый в окне диалога
Options	Свойство представляет собой совокупность уточняющих свойств и позволяет выполнить "тонкую" настройку диалога
fdoNoChangeDir	Запрещает (True) сменить текущий (открыть другой) каталог
fdoFileMustExist	Активизирует (True) режим проверки существования файла, имя которого пользователь ввел в поле редактирования
fdoPathMustExist	Активизирует (True) режим проверки существования каталога, имя которого пользователь ввел в поле редактирования
fdoNotAddToRecent	Управляет записью в список **Недавние документы** имен файлов, выбранных пользователем в окне диалога. Если значение свойства True, то имя файла в списке **Недавние документы** не фиксируется

Таблица 3.35 *(окончание)*

Свойство	Описание
`fdoHiddePinnedPlaces`	Управляет отображением избранных ссылок. Если значение свойства `True`, то отображаются только ссылки, определенные программистом (свойство `FavoriteLinks`), другие ссылки, например **Документы**, **Рабочий стол** и пр., не отображаются

Отображение диалогов `FileOpenDialog` и `FileSaveDialog` обеспечивает метод-функция `Execute`. Значение, возвращаемое методом, позволяет определить, щелчком на какой кнопке, **Открыть** (для диалога `FileOpenDialog`), **Сохранить** (для диалога `FileSaveDialog`) или **Отмена**, пользователь закрыл окно диалога.

Использование компонентов `FileOpenDialog` и `FileSaveDialog` демонстрирует программа NkEdit (редактор текста), ее форма приведена на рис. 3.47, а текст — в листинге 3.17. В начале своей работы программа проверяет ОС (делает это процедура обработки события `Activate`), и если операционная система не Vista, завершает работу.

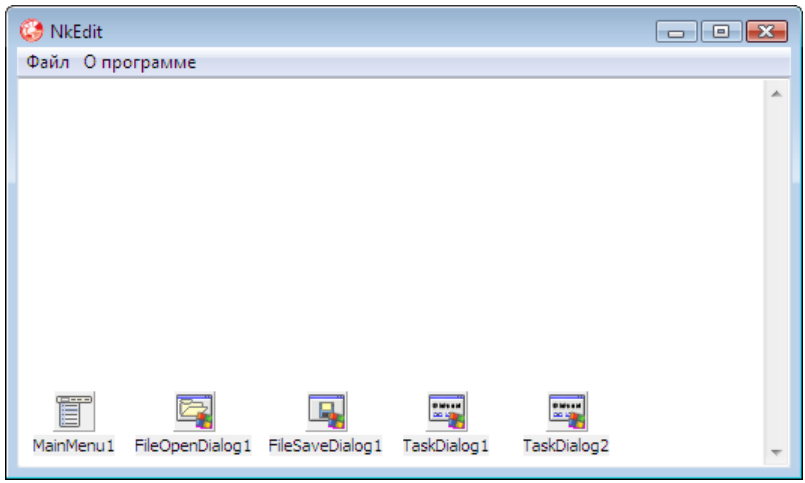


Рис. 3.47. Форма программы NkEdit

Листинг 3.17. Модуль формы NkEdit

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Menus, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    TaskDialog1: TTaskDialog;
    Memo1: TMemo;
    FileSaveDialog1: TFileSaveDialog;
    FileOpenDialog1: TFileOpenDialog;
    N5: TMenuItem;
    TaskDialog2: TTaskDialog;
    procedure FormCreate(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure N4Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N5Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
    aFile: string; // файл, в котором надо сохранить документ
    function ToFile(var aFile: string):boolean;

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
    //Form1.Width := 470;
    //Form1.Height := 520;
end;
```

```
// команда Файл >> Открыть
procedure TForm1.N2Click(Sender: TObject);
var
    r: boolean; // true — текст записан в файл
begin
    r := true;
    if Memo1.Modified then
    begin
        // сохранить текст, находящийся в поле редактирования
        TaskDialog1.Title := 'Текст изменен';
        if aFile = ''
              then
                 TaskDialog1.Text := 'Сохранить набранный текст в файле?'
              else
                 TaskDialog1.Text := 'Сохранить измененный текст в файле?';

        TaskDialog1.Execute;  // запрос о необходимости сохранения текста

        case TaskDialog1.ModalResult of
           mrYes: begin
                    r := ToFile(aFile);
                    if r then
                        Memo1.Modified := false;
                  end;
           mrNo:    r := true;
           mrCancel: r := false;
        end;
     end;

     // открыть файл
     if r and FileOpenDialog1.Execute then
        begin
            aFile := FileOpenDialog1.FileName;
            Memo1.Lines.LoadFromFile(aFile);
            Form1.Caption := 'NkEdit - ' + aFile;
        end;
end;

// команда Файл >> Сохранить
procedure TForm1.N3Click(Sender: TObject);
begin
```

```
    if ToFile(aFile) then
        Form1.Caption := 'NkEdit - ' + aFile;
end;

// команда Файл >> Выход
procedure TForm1.N4Click(Sender: TObject);
begin
  Form1.Close;
end;

// команда "О программе"
procedure TForm1.N5Click(Sender: TObject);
begin
  TaskDialog2.Execute;
end;

// попытка закрыть окно программы
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
    if Memo1.Modified // текст изменен
    then begin
        TaskDialog1.Title := 'Завершение работы';
        if aFile = ''
            then
               TaskDialog1.Text := 'Сохранить набранный текст в файле?'
             else
               TaskDialog1.Text := 'Сохранить измененный текст в файле?';

        TaskDialog1.Execute;

        // какую кнопку нажал пользователь?
        case TaskDialog1.ModalResult of
           mrYes: if ToFile(aFile) then CanClose := true;
           mrNo:     CanClose := true;
           mrCancel: CanClose := false;
        end;
    end
    else CanClose := true; // можно закрыть окно программы
end;

// записать в файл текст, находящийся в поле Memo1
// возвращает False, если пользователь отменил операцию
```

```
function TForm1.ToFile(var aFile: string): boolean;
var
    r: boolean;
begin
    r := false;
    if aFile <> '' then
    begin
        // имя файла задано, сохранить
        Memo1.Lines.SaveToFile(aFile);
        r := true;
    end
    else
         // отобразить диалог "Сохранить"
         if FileSaveDialog1.Execute then
               begin
                  // пользователь задал имя файла
                  aFile := FileSaveDialog1.FileName;
                  Memo1.Lines.SaveToFile(aFile);
                  Memo1.Modified := false;
                  r := true;
               end
          else r:= false; // пользователь не ввел имя файла
    ToFile := r;
end;


procedure TForm1.FormActivate(Sender: TObject);
var
    OSVersionInfo: TOSVersionInfo;
begin
    // определить версию ОС
    OSVersionInfo.dwOSVersionInfoSize := sizeof(TOSVersionInfo);
    GetVersionEx(OSVersionInfo);
    if OSVersionInfo.dwMajorVersion < 6 then
    begin
        MessageDlg('Программа требует Windows Vista или выше',
                    mtError, [mbOk],0);
        Form1.Close;
    end
end;

end.
```

ЧАСТЬ II

ПРАКТИКУМ ПРОГРАММИРОВАНИЯ

Эта часть книги посвящена практике — программированию графики, мультимедиа, разработке программ работы с базами данных. В ней также рассматриваются вопросы создания справочной системы и программы-установщика.

Глава 4

Графика

В этой главе рассказывается, что надо сделать, чтобы на поверхности формы появилась картинка, сформированная из графических элементов (например, график или диаграмма), иллюстрация, созданная в графическом редакторе, или фотография. Также вы познакомитесь с принципами реализации анимации, узнаете, как "оживить" картинку.

Графическая поверхность

Отображение графики обеспечивают компоненты `Image` и `PaintBox`. Компонент `Image` обычно используется для отображения иллюстраций (в том числе и фотографий), загружаемых из файла, компонент `PaintBox` — в качестве поверхности, на которой графика формируется из отдельных элементов во время работы программы.

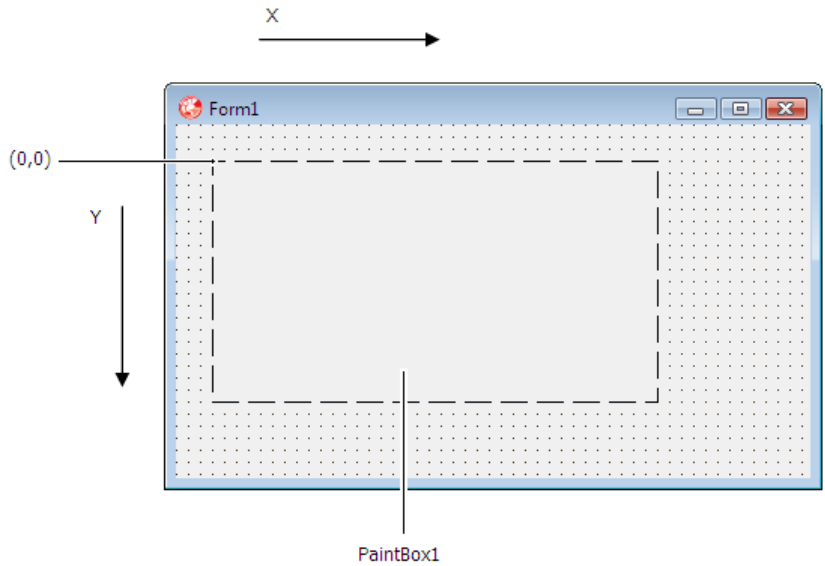


Рис. 4.1. Координаты точек графической поверхности

Графика, отображаемая в поле компонента PaintBox, формируется на его *графической поверхности*. Графическая поверхность представляет собой совокупность отдельных точек (пикселов), каждая из которых может быть окрашена каким-либо одним цветом. Положение пиксела на графической поверхности характеризуется горизонтальной (*X*) и вертикальной (*Y*) координатами. Координаты отсчитываются от левого верхнего угла и возрастают сверху вниз и слева направо (рис. 4.1). Левый верхний пиксел имеет координаты (0, 0).

Доступ к графической поверхности компонента обеспечивает свойство Canvas (*canvas* с англ. "поверхность", "холст для рисования"), представляющее собой объект типа TCanvas.

Рисование графических примитивов (линий, прямоугольников, окружностей и т. д.) обеспечивают методы класса TCanvas (табл. 4.1). Свойства класса TCanvas (табл. 4.2) определяют вид линий (цвет, толщину, стиль) и областей (цвет и стиль закраски).

Таблица 4.1. *Методы класса* `TCanvas`

Метод	Действие
`LineTo(x,y)`	Рисует линию из текущей точки в точку с указанными координатами (перемещение указателя текущей точки в нужную обеспечивает метод `MoveTo`). Цвет линии определяет свойство `Pen.Color`
`Rectangle(x1,y1,x2,y2)`	Рисует прямоугольник. Параметры `x1`, `y1` и `x2`, `y2` задают координаты находящихся на одной диагонали углов прямоугольника. Цвет границы прямоугольника определяет значение свойства `Pen.Color`, цвет закраски области — свойство `Brush.Color`
`RoundRect(x1,y1,x2,y2,x3,y3)`	Рисует прямоугольник со скругленными углами. Параметры `x1`, `y1` и `x2`, `y2` задают координаты находящихся на одной диагонали углов прямоугольника, параметры `x3`, `y3` — радиус скругления. Цвет границы прямоугольника определяет значение свойства `Pen.Color`, цвет закраски области — свойство `Brush.Color`
`Ellipse(x1,y1,x2,y2)`	Рисует эллипс (окружность). Параметры `x1`, `y1`, `x2`, `y2` задают координаты углов прямоугольника, внутри которого вычерчивается эллипс (окружность, если прямоугольник является квадратом). Цвет границы определяет значение свойства `Pen.Color`, цвет закраски области — свойство `Brush.Color`
`Arc(x1,y1,x2,y2,x3,y3,x4,y4)`	Рисует дугу. Параметры `x1`, `y1`, `x2` и `y2` задают эллипс, частью которого является дуга, параметры `x3`, `y3`, `x4` и `y4` — начальную и конечную точки дуги. Цвет дуги определяет свойство `Pen.Color`

Таблица 4.1 *(окончание)*

Метод	Действие
`Pie(x1,y1,x2,y2,x3,y3,x4,y4)`	Рисует сектор. Параметры `x1`, `y1`, `x2` и `y2` задают эллипс, частью которого является сектор, параметры `x3`, `y3`, `x4` и `y4` — границы сектора. Цвет границы сектора определяет свойство `Pen.Color`, цвет закраски сектора — свойство `Brush.Color`
`FillRect(aRect)`	Рисует закрашенный прямоугольник. Параметр `aRect` (тип `TRect`) определяет положение и размер прямоугольника. Цвет закраски области определяет свойство `Brush.Color`
`FrameRect(aRect)`	Рисует контур прямоугольника. Параметр `aRect` (тип `TRect`) определяет положение и размер прямоугольника. Цвет контура определяет свойство `Brush.Color`
`Polyline(points,n)`	Рисует ломаную линию. `Points` — массив типа `TPoint`. Каждый элемент массива представляет собой запись, поля `x` и `y` которой содержат координаты точки перегиба ломаной. `n` — количество звеньев ломаной. Метод `Polyline` вычерчивает ломаную линию, последовательно соединяя прямыми точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д.

Таблица 4.2. Свойства класса *`TCanvas`*

Свойство	Описание
`Pen`	Карандаш. Определяет цвет, стиль и толщину линии, которую рисует, например, метод `LineTo`
`PenPos`	Положение (координаты) карандаша
`Brush`	Кисть. Определяет цвет и стиль закраски области, например прямоугольника, который рисует метод `Rectangle`
`Font`	Шрифт. Определяет шрифт, который используется для вывода текста, например методом `TextOut`

Необходимо обратить внимание, что у формы также есть свойство `Canvas`, т. е. программа может вывести графику не только в поле компонента `PaintBox`, но и непосредственно на поверхность формы.

Методы вывода графических примитивов рассматривают свойство `Canvas` как поверхность, на которой они могут *рисовать*.

Следует обратить внимание на важный момент. Графику на поверхности формы или компонента PaintBox должна формировать процедура обработки события Paint. Это событие возникает в момент запуска программы, когда окно появляется на экране первый раз, а также всякий раз, когда окно (или его часть) появляется на экране снова, например после того как пользователь сдвинет другое окно, частично или полностью перекрывающее окно программы, или развернет окно программы, если оно было свернуто.

Карандаш и кисть

Вид графического элемента определяют свойства Pen (Карандаш) и Brush (Кисть) той поверхности (Canvas), на которой рисует метод.

Карандаш и кисть, являясь свойствами объекта Canvas, представляют собой объекты Pen и Brush. Свойства объекта Pen (табл. 4.3) определяют цвет, толщину и тип линии, свойства объекта Brush (табл. 4.4) — цвет и стиль закраски областей.

***Таблица 4.3.** Свойства объекта Pen (Карандаш)*

Свойство	Описание
Color	Цвет линии
Width	Толщина линии (задается в пикселах)
Style	Вид линии: psSolid — сплошная; psDash — пунктирная, длинные штрихи; psDot — пунктирная, короткие штрихи; psDashDot — пунктирная, чередование длинного и короткого штрихов; psDashDotDot — пунктирная, чередование одного длинного и двух коротких штрихов; psClear — линия не отображается (используется, если не надо изображать границу области, например прямоугольника)

***Таблица 4.4.** Свойства объекта Brush (Кисть)*

Свойство	Описание
Color	Цвет закраски замкнутой области
Style	Стиль закраски области: bsSolid — сплошная заливка; штриховка: bsHorizontal — горизонтальная; bsVertical — вертикальная; bsFDiagonal — диагональная с наклоном линий вперед; bsBDiagonal — диагональная с наклоном линий назад; bsCross — в клетку; bsDiagCross — диагональная клетка

Чтобы задать цвет карандаша или кисти, надо присвоить значение соответственно свойству Pen.Color или Brush.Color. В табл. 4.5 приведены именованные константы, которые можно использовать, чтобы задать цвет.

Таблица 4.5. *Константы* `TColor`

Цвет	Константа	Цвет	Константа
`ClAqua`	Бирюзовый	`clNavy`	Темно-синий
`clBlack`	Черный	`clOlive`	Оливковый
`ClBlue`	Синий	`clPurple`	Фиолетовый
`ClFuchsia`	Ярко-розовый	`ClRed`	Красный
`clGreen`	Зеленый	`ClSilver`	Серебристый
`ClLime`	Салатный	`clTeal`	Зелено-голубой
`clMaroon`	Каштановый	`ClWhite`	Белый

Если необходимо установить цвет, отличный от стандартного, то в свойство `Color` надо записать его RGB-код (как известно, любой цвет можно получить путем смешивания в разных пропорциях красной, зеленой и синей красок). Получить код цвета можно, обратившись к функции `RGB`, указав в качестве параметров долю красной, зеленой и синей составляющей. Например, значение `RGB(56,176,222)` — это код цвета "осеннее небо".

Графические примитивы

Картинку, чертеж или схему можно рассматривать как совокупность графических *примитивов*: точек, линий, окружностей, дуг, а также букв (текста).

Вычерчивание графических примитивов на графической поверхности, например компонента `PaintBox`, выполняют соответствующие методы класса `TCanvas`.

Инструкция, обеспечивающая вычерчивание графического элемента, в общем виде выглядит так:

`Объект``.Canvas.`*`Метод`*`(`*`Параметры`*`)`

`Объект` определяет объект, на поверхности которого нужно нарисовать графический элемент. В качестве объекта обычно указывается компонент `PaintBox`.

`Метод` — это имя метода, который обеспечивает рисование нужного графического элемента.

`Параметры`, в большинстве случаев, определяют положение графического элемента на графической поверхности и его размер.

Например, в результате выполнения инструкции

```
PaintBox1.Canvas.Rectangle(10,20,60,40)
```

в поле компонента `PaintBox1` будет нарисован прямоугольник шириной 50 и высотой 20 пикселов, левый верхний угол которого будет находиться в точке (10, 20).

При записи инструкций, обеспечивающих вывод графики, удобно использовать инструкцию `with`, которая позволяет сократить количество набираемого кода. Например, вместо:

```
// итальянский флаг
PaintBox1.Canvas.Brush.Color := clGreen;
PaintBox1.Canvas.Rectangle(20,20,46,70);
PaintBox1.Canvas.Brush.Color := clWhite;
PaintBox1.Canvas.Rectangle(45,20,71,70);
PaintBox1.Canvas.Brush.Color := clRed;
PaintBox1.Canvas.Rectangle(70,20,96,70);

PaintBox1.Canvas.Brush.Style := bsClear;
PaintBox1.Canvas.Font.Name := 'Tahoma';
PaintBox1.Canvas.Font.Size := 10;
x := 20 + (75 - PaintBox1.Canvas.TextWidth('Италия')) div 2;
PaintBox1.Canvas.TextOut(x,70 + Font.Size,'Италия');
```

можно написать:

```
with PaintBox1.Canvas do
    begin
        // итальянский флаг
        Brush.Color := clGreen;
        Rectangle(20,20,46,70);
        Brush.Color := clWhite;
        Rectangle(45,20,71,70);
        Brush.Color := clRed;
        Rectangle(70,20,96,70);

        Brush.Style := bsClear;
        Font.Name := 'Tahoma';
        Font.Size := 10;
        x := 20  + (75 - TextWidth('Италия')) div 2;
        TextOut(x,70 + Font.Size,'Италия');
    end;
```

Текст

Вывод строки текста на графическую поверхность объекта обеспечивает метод `TextOut`. Инструкция вызова метода `TextOut` в общем виде выглядит следующим образом:

Объект`.Canvas.TextOut(x,y,`*Текст*`)`

Параметры `x` и `y` определяют координаты точки графической поверхности, от которой выполняется вывод текста (рис. 4.2). Необходимо обратить внимание на то, что область вывода текста закрашивается текущим цветом кисти, а после вывода текста карандаш автоматически перемещается в правый верхний угол области вывода текста. Также следует обратить внимание на то, что размер области вывода текста зависит от количества символов (длины текста) и характеристик шрифта, который используется для отображения текста.

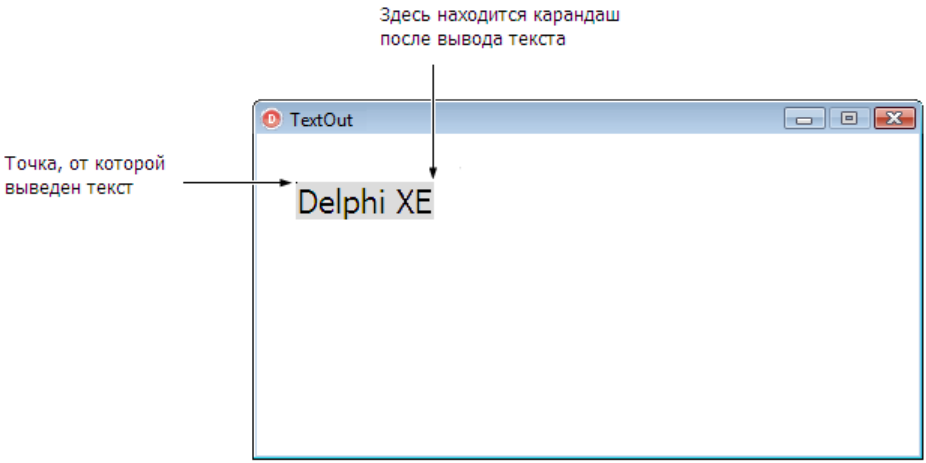


Рис. 4.2. Координаты области вывода текста

Шрифт, используемый для отображения текста, определяет свойство `Font` графической поверхности, на которую текст выводится. Свойство `Font` представляет собой объект типа `TFont`. В табл. 4.6 перечислены свойства класса `TFont`.

Таблица 4.6. *Свойства класса* `TFont`

Свойство	Определяет
`Name`	Шрифт, который используется для отображения текста. В качестве значения следует брать название шрифта, например `Arial`
`Size`	Размер шрифта
`Style`	Стиль начертания символов. Задается с помощью констант: `fsBold` (полужирный), `fsItalic` (курсив), `fsUnderline` (подчеркнутый), `fsStrikeOut` (перечеркнутый). Свойство `Style` является множеством, что позволяет комбинировать необходимые стили. Например, инструкция, которая устанавливает стиль "полужирный курсив", выглядит так: `Font.Style := [fsBold, fsItalic]`
`Color`	Цвет символов. В качестве значения можно использовать константу типа `TColor`

При выводе текста весьма полезны функции (методы класса `TCanvas`) `TextWidth` и `TextHeight`, которые позволяют определить размер (соответственно ширину и высоту) области вывода текста. Обоим этим методам в качестве параметра передается строка, которую предполагается вывести на графическую поверхность методом `TextOut`.

Использование метода `TextOut` для вывода текста на поверхность формы демонстрирует следующий фрагмент кода (листинг 4.1). Приведенная процедура обработки события `Paint` загружает из файла фоновый рисунок, выводит его и затем в центре окна выводит текст (рис. 4.3). Следует обратить внимание, что у текста есть тень. Эффект тени достигается вследствие того, что текст сначала выводится белым цветом, затем, со смещением на один пиксел влево и вниз — черным.

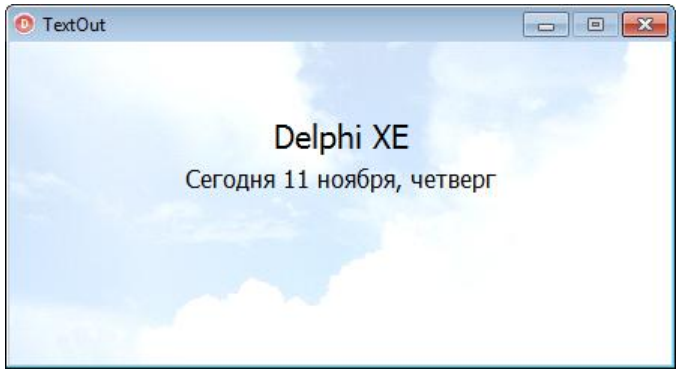


Рис. 4.3. Вывод текста на графическую поверхность

Листинг 4.1. Вывод текста на графическую поверхность

```
type
  TForm1 = class(TForm)
    PaintBox1: TPaintBox;
    procedure PaintBox1Paint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    back: TBitmap; // фоновый рисунок
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}
uses
  DateUtils; // для доступа к MonthOf
var
  month: array[1..12] of string = ('января','февраля','марта','апреля',
                                   'мая','июня','июля','августа',
                                   'сентября','октября','ноября','декабря');
// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  // загрузить фоновый рисунок
  back := TBitmap.Create;
  back.LoadFromFile('skylight.bmp');

  // установить размер формы в соответствии с размером фонового рисунка
  Form1.ClientWidth := back.Width;
  Form1.ClientHeight := back.Height;
end;

procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  st: string;    // текст
  x,y: integer;  // точка, от которой выводится текст
  w,h: integer;  // размер области отображения текста

  aRect: TRect;  // область вывода нескольких строк текста
begin
  with PaintBox1.Canvas do
  begin

    // отобразить фоновый рисунок
    Draw(0,0,back);

    // Разместим текст в центре окна (по горизонтали).
    // Размер области вывода зависит от шрифта,
    // который используется для отображения текста

    // Чтобы область вывода текста не была видна,
    // кисть должна быть "прозрачной"
    Brush.Style := bsClear;  // "прозрачная" кисть
```

```
    st := 'Delphi XE';
    Font.Name := 'Tahoma';
    Font.Size := 16;
    Font.Color := clBlack;

    w := TextWidth(st);
    h := TextHeight(st);
    x := Round((PaintBox1.Width - w) / 2);
    y := 50;

    TextOut(x,y,st);

    y := y+ h;

    Font.Size := 12;
    Font.Color := clBlack;
    //Font.Style := [fsItalic];

    st := FormatDateTime('Сегодня d ',now) + month[MonthOf(Now)]
        + FormatDateTime(', dddd',now);

    w := TextWidth(st);
    x := Round((PaintBox1.Width - w) / 2);
    y := y + Round(Font.Size / 2);

    TextOut(x,y,st);
  end;
end;
```

Часто требуется вывести какой-либо текст после сообщения, длина которого во время разработки программы неизвестна. В этом случае необходимо знать координаты правой границы области выведенного текста. Координаты правой границы текста, показанного методом `TextOut`, можно получить, обратившись к свойству `PenPos`.

Следующий фрагмент кода демонстрирует возможность вывода строки текста с помощью двух инструкций `TextOut`.

```
with PaintBox1.Canvas do
    begin
      TextOut(10,10,'Embarcadero ');
      TextOut(PenPos.x, PenPos.y, 'Delphi XE');
    end;
```

Линия

Метод `LineTo` рисует линию из точки, в которой в данный момент находится карандаш, в точку, координаты которой указаны в инструкции вызова метода. Инструкция вызова метода в общем виде выглядит так:

Объект`.Canvas.LineTo(x,y)`

Цвет, стиль и толщину линии определяют соответственно свойства `Pen.Color`, `Pen.Style` и `Pen.Width` графической поверхности, на которой метод рисует. Начальную точку линии можно задать, переместив карандаш в нужную точку. Сделать это можно с помощью метода `MoveTo` или присвоив значение свойству `PenPos`. Следует обратить внимание, что после того как линия будет нарисована, карандаш будет находиться в точке ее конца.

В качестве примера в листинге 4.2 приведен фрагмент кода, который демонстрирует различные виды линий (рис. 4.4). Необходимо обратить внимание на то, что пунктиром можно нарисовать только линию толщиной в один пиксел.

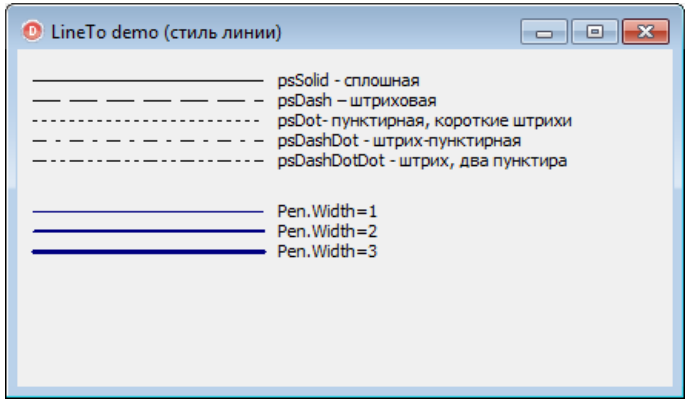


Рис. 4.4. Вид линии определяет значение свойства `Pen.Style`

Листинг 4.2. Стиль линии

```
var
  st: array[1..5] of string = (
    'psSolid - сплошная',
    'psDash - штриховая',
    'psDot - пунктирная, короткие штрихи',
    'psDashDot - штрих-пунктирная',
    'psDashDotDot - штрих, два пунктира');

procedure TForm1.PaintBox1Paint(Sender: TObject);
const
```

```
  L = 150; // длина линии
var
  x,y: integer;   // точка конца линии
  i: integer;
begin
  Canvas.MoveTo(10,20);
  Canvas.Pen.Width := 1;
  for i:=1 to 5 do
  begin
    case i of
      1: Canvas.Pen.Style := psSolid;
      2: Canvas.Pen.Style := psDash;
      3: Canvas.Pen.Style := psDot;
      4: Canvas.Pen.Style := psDashDot;
      5: Canvas.Pen.Style := psDashDotDot;
    end;
    Canvas.LineTo(Canvas.PenPos.X + L, Canvas.PenPos.Y);
    Canvas.TextOut(Canvas.PenPos.X+10, Canvas.PenPos.Y-7, st[i]);
    Canvas.MoveTo(10,Canvas.PenPos.Y + 20);
  end;

  Canvas.Pen.Style := psSolid;
  Canvas.Pen.Color := clNavy;
  for i:=1 to 3 do
  begin
    Canvas.Pen.Width := i;
    Canvas.MoveTo(10,Canvas.PenPos.Y + 20);
    Canvas.LineTo(Canvas.PenPos.X + L, Canvas.PenPos.Y);
    Canvas.TextOut(Canvas.PenPos.X+10, Canvas.PenPos.Y-7,
                   'Pen.Width='+ IntToStr(i));
  end;
end;
```

Следующая программа, ее текст приведен в листинге 4.3, строит в поле компонента `PaintBox` график изменения курса доллара (рис. 4.5). Данные загружаются из файла (листинг 4.4). Следует обратить внимание на то, что разница между минимальным и максимальным значениями ряда данных, отображаемых на графике, незначительна, поэтому график строится в отклонениях от минимального значения. Координата *Y* точек графической поверхности возрастает сверху вниз, а на графике — снизу вверх. Поэтому координата *Y* точки отсчитывается вверх от нижней границы компонента `PaintBox`.

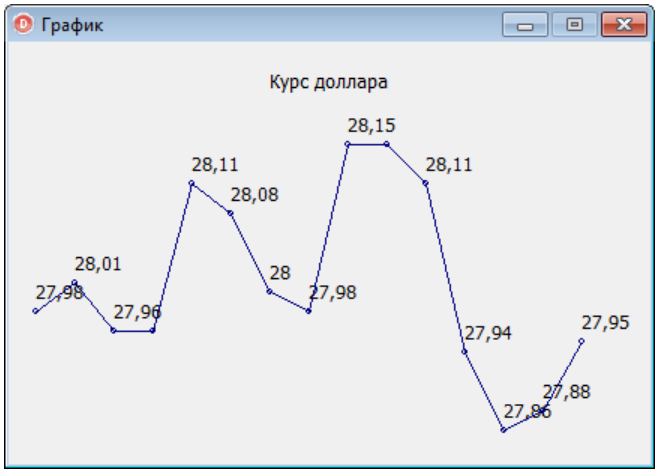


Рис. 4.5. График

Листинг 4.3. График

```
var
  kurs: array[1..15] of real; // массив данных
  nrec: integer; // количество чисел, прочитанных из файла

procedure TForm1.FormCreate(Sender: TObject);
var
  f: TextFile;
  i: integer;
  //size: integer; // размер массива (кол-во элементов) kurs
begin
  // sizeof(kurs) - кол-во байтов, занимаемых массивом
  // sizeof(real) - кол-во байтов, занимаемых числом real
  //size := Round(sizeof(kurs) / sizeof(real));
  AssignFile(f,'kurs.txt');

  try
    reset(f); // открыть для чтения
    i:=0;
    while (NOT EOF(f)) AND (i < 15) do
      begin
        i := i +1;
        readln(f,kurs[i]);
      end;
```

```
    nrec := i;

  except on EInOutError do
    //MessageDlg('Нет файла данных kurs.txt', mtError,[mbOK],0);
  end;
end;

procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    n: integer ;      // количество точек
    x,y: integer;     // координаты точки
    dx: integer;      // шаг по X

    min,max: integer; // индекс минимального и максимального элемента
    m: real;          // масштаб

    i: integer;
    st: string;
begin
    if nrec=0 then
    begin
      PaintBox1.Canvas.Font.Size := 12;
      PaintBox1.Canvas.TextOut(10,10,'Ошибка! Нет файла данных (kurs.txt)');
      // данные из файла не прочитаны
      exit;  // выход из процедуры
    end;

    // строим график
    with PaintBox1.Canvas do begin
      // заголовок
      Font.Name := 'Tahoma';
      Font.Size := 10;
      x := Round((PaintBox1.Width - TextWidth('Курс доллара')) /2);
      Brush.Style := bsClear;
      TextOut(x,10,'Курс доллара');

      // определить кол-во элементов массива
      //n := Round(sizeof(kurs) / sizeof(real));
      n:=nrec;

      // найти минимальное и максимальное значение ряда данных
      min := 1;  // пусть первый элемент минимальный
```

```
max := 1;  // пусть первый элемент максимальный
for i := 1 to n do
  begin
    if (kurs[i] < kurs[min]) then min := i;
    if (kurs[i] > kurs[max]) then max := i;
  end;

  { Если разница между минимальным и максимальным значениями
    незначительна, то график получается ненаглядным.
    В этом случае можно построить не абсолютные значения,
    а отклонения от минимального значения ряда. }

  Font.Size := 9;
  Pen.Width := 1;
  Pen.Color := clNavy;

  dx:= Round((PaintBox1.Width - 40) / (n-1));

  // Вычислим масштаб.
  // Для построения графика будем использовать не всю область
  // компонента, а ее нижнюю часть; верхняя область высотой
  // 60 пикселов используется для отображения заголовка
  m := (PaintBox1.Height - 60) / (kurs[max] - kurs[min]);

  x := 10;
  for i := 1 to n do
    begin
      y := PaintBox1.Height - Round((kurs[i] - kurs[min]) * m)-10;

      // поставить точку
      // Rectangle(x-2,y-2,x+2,y+2);
      Ellipse(x-2,y-2,x+2,y+2);
      if (i <> 1) then
          LineTo(x,y);

      // подпись данных
      if ((i = 1) or (kurs[i] <> kurs[i-1])) then
      begin
        st := FloatToStrF(kurs[i],ffGeneral,5,2);
        Brush.Style := bsClear;  // область вывода текста - прозрачная
        TextOut(x,y-20,st);
      end;
```

```
            { Так как метод TextOut меняет положение точки (карандаша),
              из которой рисует метод LineTo, то после вывода текста
              надо переместить указатель (карандаш) в точку (x,y). }
             MoveTo(x,y);
             x := x + dx;
           end;
    end;
end;
```

Листинг 4.4. Файл данных программы "График" (kurs.txt)

```
27.98
28.01
27.96
27.96
28.11
28.08
28.00
27.98
28.15
28.15
28.11
27.94
27.86
27.88
27.95
```

Ломаная линия

Метод `Polyline` чертит ломаную линию. Инструкция вызова метода в общем виде выглядит так:

Объект`.Canvas.Polyline(p)`

В качестве параметров методу передается массив типа `TPoint`, элементы которого содержат координаты узловых точек линии.

Метод `Polyline` вычерчивает ломаную линию, последовательно соединяя точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Цвет, стиль и толщину линии определяют соответственно свойства `Pen.Color`, `Pen.Style` и `Pen.Width` той поверхности, на которой метод чертит.

Следующий фрагмент кода рисует треугольный флажок (рис. 4.6), который можно рассматривать как ломаную, состоящую из трех звеньев (номера точек ломаной соответствуют индексам элементов массива).

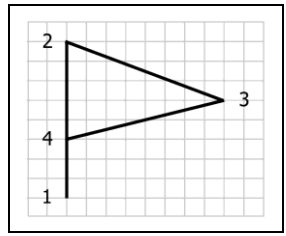


Рис. 4.6. Пример ломаной линии

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  p: array[1..4] of TPoint;
begin
  // координаты будем отсчитывать от верхней точки древка
  p[2].X := 10;
  P[2].Y := 10;
  p[3].X := p[2].X + 48;
  P[3].Y := p[2].Y + 16;
  p[4].X := p[2].X;
  P[4].Y := p[2].Y + 32;
  p[1].X := p[2].X;
  P[1].Y := p[2].Y + 52;

  PaintBox1.Canvas.Polyline(p);
end;
```

В приведенном примере базовой точкой является верхняя точка древка (от нее отсчитываются координаты остальных точек), а начальной точкой ломаной — точка, соответствующая нижней точке древка.

Метод `Polyline` можно использовать для вычерчивания замкнутых контуров. Для этого надо, чтобы первый и последний элементы массива содержали координаты одной и той же точки.

Прямоугольник

Метод `Rectangle` вычерчивает прямоугольник. Инструкция вызова метода в общем виде выглядит так:

Объект`.Canvas.Rectangle(x1,y1,x2,y2)`

Параметры `x1`, `y1` и `x2`, `y2` задают координаты углов прямоугольника. Цвет, вид и ширину линии контура определяют соответственно значения свойств `Pen.Color`, `Pen.Width` и `Pen.Style`, а цвет и стиль заливки внутренней области — значения свойств `Brush.Color` и `Brush.Style` той поверхности, на которой метод рисует.

В качестве примера в листинге 4.5 приведена процедура обработки события `Paint`, которая на поверхности формы рисует итальянский и французский флаги (рис. 4.7).

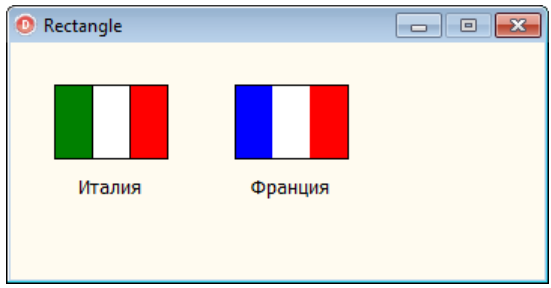


Рис. 4.7. Метод `Rectangle` рисует прямоугольник

Листинг 4.5. Французский и итальянский флаги (метод `Rectangle`)

```
procedure TForm1.FormPaint(Sender: TObject);
var
  x: integer;
begin
  with Form1.Canvas do
  begin
    // итальянский флаг
    Brush.Color := clGreen;
    Rectangle(20,20,46,70);
    Brush.Color := clWhite;
    Rectangle(45,20,71,70);
    Brush.Color := clRed;
    Rectangle(70,20,96,70);

    Brush.Style := bsClear;
    Font.Name := 'Tahoma';
    Font.Size := 10;
    x := 20  + (75 - TextWidth('Италия')) div 2;
    TextOut(x,70 + Font.Size,'Италия');

    // Французский флаг.
    // Чтобы не было контура вокруг полос, цвет контура
    // должен совпадать с цветом заливки
    Brush.Color := clBlue;
    Pen.Color := clBlue;
```

```
    Rectangle(140,20,166,70);
    Brush.Color := clWhite;
    Pen.Color := clWhite;
    Rectangle(165,20,191,70);
    Pen.Color := clRed;
    Brush.Color := clRed;
    Rectangle(190,20,216,70);

    // контур флага
    Pen.Color := clBlack;
    Brush.Style := bsClear; // "прозрачная" кисть
    Rectangle(140,20,216,70);

    Brush.Style := bsClear;
    Pen.Color := clBlack;
    x := 140 + (75 - TextWidth('Франция')) div 2;
    TextOut(x, 70+Font.Size,'Франция');
  end;
end;
```

В инструкции вызова метода `Rectangle` в качестве параметра метода можно указать структуру типа `TRect`. Поля структуры содержат координаты двух расположенных на одной диагонали углов прямоугольной области. Задать положение области можно, записав значения в поля `Left`, `Top`, `Right` и `Bottom` или в поля `TopLef` и `BottomRight`. Также для инициализации полей структуры можно использовать функцию `Rect`. Далее приведен фрагмент кода, который демонстрирует использование структуры `TRect` в качестве параметра метода `Rectangle`.

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  aRect: TRect;
  p1,p2: TPoint;

begin
  with PaintBox1.Canvas do
  begin
    aRect.Left := 10;
    aRect.Top := 20;
    aRect.Right := 30;
    aRect.Bottom := 40;
    Rectangle(aRect);

    p1.X := 50; p1.Y := 20;
    p2.X := 70; p2.Y := 40;
```

```
    aRect.TopLeft := p1;
    aRect.BottomRight := p2;
    Rectangle(aRect);
  end;
end;
```

Если нужно нарисовать только контур прямоугольника или закрашенный прямоугольник, цвет границы которого совпадает с цветом закраски, то вместо метода `Rectangle` можно применить соответственно метод `FrameRect` или `FillRect`. Необходимо обратить внимание на то, что для рисования контура метод `FrameRect` использует кисть, а не карандаш.

Использование методов `FillRect` и `FrameRect` демонстрирует следующая программа (листинг 4.6).

Листинг 4.6. Методы `FillRect` и `FrameRect`

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  r: TRect;
  x: integer;
begin
  with PaintBox1.Canvas do
  begin
    // французский флаг рисуем методами FillRect и FrameRect
    r := Rect(140,20,165,70);
    Brush.Color := clBlue;
    FillRect(r);

    r.Left := 165; r.Right := 190;
    Brush.Color := clWhite;
    FillRect(r);

    Brush.Color := clRed;
    r.Left := 190; r.Right := 215;
    FillRect(r);

    // контур
    Brush.Color := clBlack;
    r.Left := 140; r.Right := 215;
    FrameRect(r);

    Font.Name := 'Tahoma';
    Font.Size := 10;
```

```
    Brush.Style := bsClear;
    Pen.Color := clBlack;
    x := 140 + (75 - TextWidth('Франция')) div 2;
    TextOut(x, 70+Font.Size,'Франция');
  end;
end;
```

Метод `RoundRect` вычерчивает прямоугольник со скругленными углами. Инструкция вызова метода `RoundRec` в общем виде выглядит так:

```
Объект.Canvas.RoundRect(x1,y1,x2,y2,x3,y3)
```

Параметры `x1`, `y1`, `x2`, `y2` определяют положение углов прямоугольника, а параметры `x3` и `y3` — размер эллипса, одна четверть которого используется для вычерчивания скругленного угла (рис. 4.8).

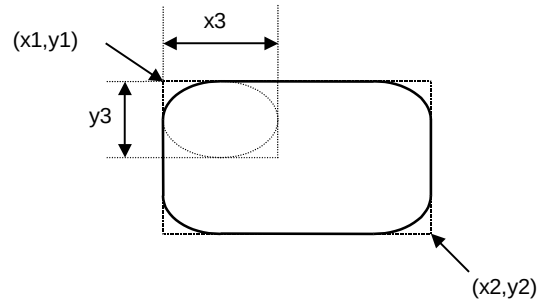


Рис. 4.8. Метод `RoundRec` вычерчивает прямоугольник со скругленными углами

Многоугольник (полигон)

Метод `Polygon` вычерчивает полигон (многоугольник). Инструкция вызова метода в общем виде выглядит так:

```
Объект.Canvas.Polygon(p)
```

где `p` — массив записей типа `TPoint`, который содержит координаты вершин многоугольника (`p[i].X` и `p[i].Y` — координаты `X` и `Y` `i`-ой вершины полигона).

Метод `Polygon` чертит полигон, соединяя прямыми линиями точки, координаты которых находятся в массиве: первую точку со второй, вторую с третьей, третью с четвертой и т. д. Цвет, вид и ширину линии контура определяют соответственно значения свойств `Pen.Color`, `Pen.Width` и `Pen.Style`, а цвет и стиль заливки внутренней области — значения свойств `Brush.Color` и `Brush.Style` той поверхности, на которой метод рисует.

В качестве примера в листинге 4.7 приведена программа, которая в виде полигона показывает динамику изменения курса доллара (рис. 4.9). Данные загружаются из файла.

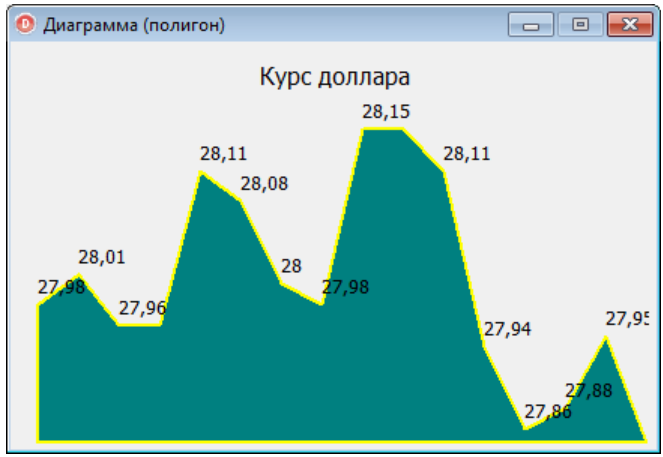


Рис. 4.9. Полигон

Листинг 4.7. Диаграмма (полигон)

```
var
  kurs: array[1..15] of real;  // массив данных
  min,max: integer; // мин. и макс. значения ряда данных
  nrec: integer;    // количество чисел, прочитанных из файла

  p: array[0..16] of TPoint; // координаты точек полигона
  // p[0] — левый нижний угол полигона,
  // p[1]..p[15] — углы, изображающие данные
  // p[16] — правый нижний угол полигона,

procedure TForm1.FormCreate(Sender: TObject);
var
  f: TextFile;
  i: integer;

begin
  AssignFile(f,'kurs.txt');

  try
    reset(f); // открыть для чтения
    i:=0;
    while (NOT EOF(f)) AND (i < 15) do
      begin
        i := i +1;
```

```
        readln(f,kurs[i]);
      end;
    CloseFile(f); // закрыть файл

    nrec := i;

    // найти минимальное и максимальное значения ряда данных
    min := 1;  // пусть первый элемент минимальный
    max := 1;  // пусть первый элемент максимальный
    for i := 1 to nrec do
        begin
          if (kurs[i] < kurs[min]) then min := i;
          if (kurs[i] > kurs[max]) then max := i;
        end;

  except on EInOutError do
    //MessageDlg('Нет файла данных kurs.txt', mtError,[mbOK],0);
  end;
end;

procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    x,y: integer;     // координаты точки
    dx: integer;      // шаг по X

    m: real;          // масштаб

    i: integer;
    st: string;

begin
    if nrec=0 then
    begin
      PaintBox1.Canvas.Font.Size := 12;
      PaintBox1.Canvas.TextOut(10,10,'Ошибка! Нет файла данных (kurs.txt)');
      // данные из файла не прочитаны
      exit;  // выход из процедуры
    end;

    // строим график
    with PaintBox1.Canvas do begin
```

```
// заголовок
Font.Name := 'Tahoma';
Font.Size := 12;
x := Round((PaintBox1.Width - TextWidth('Курс доллара')) /2);
Brush.Style := bsClear;
TextOut(x,5,'Курс доллара');

// *** вычислим координаты углов полигона ***

dx:= Round((PaintBox1.Width - 40) / (nrec-1));

// Вычислим масштаб.
// Для построения графика будем использовать не всю область
// компонента PaintBox, а ее нижнюю часть. Верхняя область
// высотой 60 пикселов используется для отображения заголовка
m := (PaintBox1.Height - 60) / (kurs[max] - kurs[min]);

x := 10;

// левый нижний угол полигона
p[0].X := x;
p[0].Y := PaintBox1.Height-1;

for i := 1 to nrec do
    begin
      // вычислить координату Y
      y := PaintBox1.Height - Round((kurs[i] - kurs[min]) * m)-10;
      // записать координаты точки в массив p
      p[i].X := x;
      p[i].Y := y;

      x := x + dx;
    end;

// правый нижний угол полигона
p[16].X := x;
p[16].Y := PaintBox1.Height-1;

// *** рисуем полигон ***
// цвет закраски
PaintBox1.Canvas.Brush.Color := clTeal;
```

```
      // цвет и ширина контура
      PaintBox1.Canvas.Pen.Color := clYellow;
      PaintBox1.Canvas.Pen.Width := 2;
      // полигон
      PaintBox1.Canvas.Polygon(p);

      // подписи данных
      Font.Size := 10;
      Brush.Style := bsClear;  // область вывода текста — прозрачная
      for i := 1 to nrec do
          begin
            if ((i = 1) or (kurs[i] <> kurs[i-1])) then
              begin
                st := FloatToStrF(kurs[i],ffGeneral,5,2);
                TextOut(p[i].X,p[i].Y-20,st);
              end;
          end;
  end;
end;
```

Окружность и эллипс

Нарисовать эллипс или окружность можно с помощью метода `Ellipse`. Инструкция вызова метода в общем виде выглядит следующим образом:

Объект`.Canvas.Ellipse(x1,y1,x2,y2)`

Параметры `x1`, `y1`, `x2`, `y2` определяют координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, окружность (рис. 4.10). Цвет, вид и ширину линии эллипса определяют соответственно значения свойств `Pen.Color`, `Pen.Width` и `Pen.Style`, а цвет и стиль заливки внутренней области — значения свойств `Brush.Color` и `Brush.Style` той поверхности, на которой метод рисует.

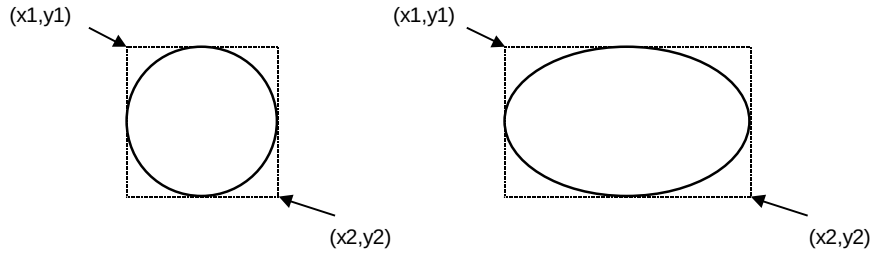


Рис. 4.10. Значения параметров метода `Ellipse` определяют вид геометрической фигуры

Вместо четырех параметров — координат диагональных углов прямоугольника, методу `Ellipse` можно передать один — объект типа `TRect`.

Дуга

Метод `Arc` рисует дугу — часть эллипса. Инструкция вызова метода в общем виде выглядит так:

```
Объект.Canvas.Arc(x1,y1,x2,y2,x3,y3,x4,y4)
```

Параметры `x1`, `y1`, `x2`, `y2` определяют эллипс, частью которого является дуга; `x3` и `y3` — начальную, а `x4` и `y4` — конечную точку дуги. Начальная (конечная) точка дуги — это точка пересечения границы эллипса и прямой, проведенной из центра эллипса в точку с координатами `x3` и `y3` (`x4`, `y4`). Метод `Arc` вычерчивает дугу против часовой стрелки от начальной точки к конечной (рис. 4.11).

Цвет, вид и ширину дуги определяют соответственно значения свойств `Pen.Color`, `Pen.Width` и `Pen.Style` той поверхности, на которой метод рисует.

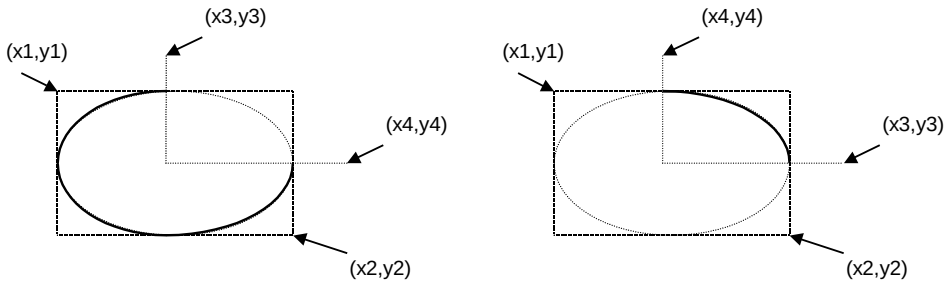


Рис. 4.11. Значения параметров метода `Arc` определяют дугу как часть эллипса

Сектор

Метод `Pie` рисует сектор эллипса. Инструкция вызова метода в общем виде выглядит следующим образом:

```
Объект.Canvas.Pie(x1,y1,x2,y2,x3,y3,x4,y4)
```

Параметры `x1`, `y1`, `x2`, `y2` определяют эллипс, частью которого является сектор; `x3`, `y3`, `x4` и `y4` — прямые границы сектора. Начальная точка границ совпадает с центром эллипса. Сектор вырезается против часовой стрелки от прямой, заданной точкой с координатами (`x3`, `y3`), к прямой, заданной точкой с координатами (`x4`, `y4`) (рис. 4.12).

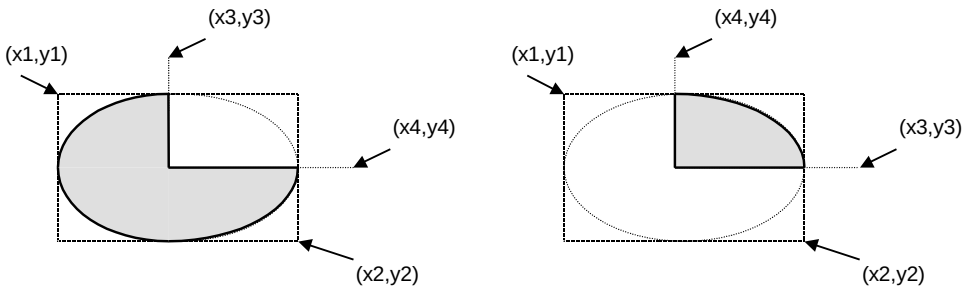


Рис. 4.12. Значения параметров метода `Pie` определяют сектор как часть эллипса (окружности)

Использование метода `Pie` демонстрирует программа "Круговая диаграмма" (ее окно приведено на рис. 4.13). В окне программы отображается круговая диаграмма. Текст процедуры приведен в листинге 4.8. Строит диаграмму процедура обработки события `Paint`. Процедура `preparate` выполняет предварительную обработку данных: сортирует данные по возрастанию и вычисляет долю каждой категории в общей сумме. Загрузку данных из файла (листинг 4.9) выполняет процедура обработки события `Create` формы.

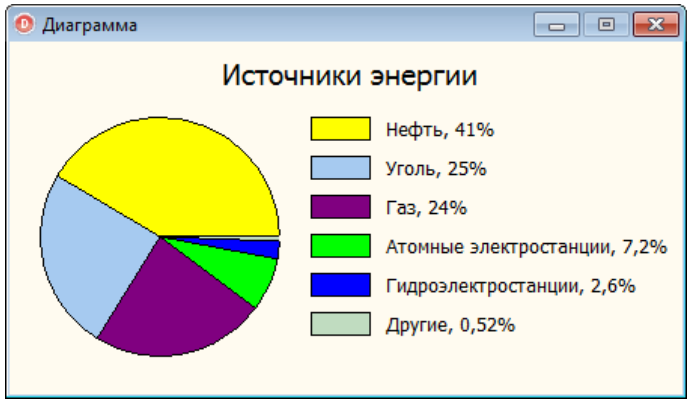


Рис. 4.13. Круговая диаграмма

Листинг 4.8. Круговая диаграмма

```
const
  R = 80;  // радиус диаграммы
  HB = 6;  // количество категорий данных

var
  // исходные данные
  Title: string;
  data: array[1..HB] of real;
  percent: array[1..HB] of real;  // доля категории в общей сумме (процент)

  // подписи данных
  dTitle: array[1..HB] of string;

  // цвет для каждой категории
  cl: array[1..HB] of TColor = (clLime, clBlue, clPurple, clSkyBlue,
                                clYellow, clMoneyGreen);

  dataok: boolean; // True — данные загружены успешно
```

```
// обработка исходных данных: сортировка по возрастанию
// и вычисление процента
procedure preparation;
var
  i,j: integer;

  // буферные переменные, используемые в процессе сортировки
  // массивов data, dTitle и cl
  bd: real;
  bt: string;
  bc: TColor;

  sum: real;
begin
    // сортировка исходных данных методом "пузырька"
    for i := 1 to HB-1 do
        for j := 1 to HB-1 do
            if (data[j+1] < data[j]) then
            begin
                // обменять местами j-й и j+1-й элементы массива
                bd := data[j];
                data[j] := data[j+1];
                data[j+1] := bd;

                bt := dTitle[j];
                dTitle[j] := dTitle[j+1];
                dTitle[j+1] := bt;

                bc := cl[j];
                cl[j] := cl[j+1];
                cl[j+1] := bc;
            end;

    // обработка данных — вычисление доли каждой категории в общей сумме
    sum := 0;
    for i := 1 to HB do
        sum := sum + data[i];

    for i := 1 to HB do
        percent[i] := (data[i] / sum) * 100;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  f: TextFile;
  i: integer;
begin
  dataok := False;
  try
    AssignFile(f,'data.txt');
    reset(f);

    readln(f,Title); // заголовок диаграммы
    i:=0;
    // читать данные из файла
    while NOT EOF(f) do
      begin
        i:=i+1;
        readln(f,dTitle[i]); // категория
        readln(f,data[i]);   // значение
      end;
    CloseFile(f);
    dataok := True;
    preparation;

  except on e: EInOutError do
      //dataok:= False;
  end;
end;

// процедура обработки события Paint рисует диаграмму
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    x0,y0: integer; // центр сектора (круга)
    x1,y1,x2,y2: integer; // координаты области, в которую вписан круг,
                          // из которого вырезается сектор
    x3,y3: integer; // координата точки начала дуги
    x4,y4: integer; // координата точки конца дуги
    a1,a2: integer; // угол между OX и прямыми, ограничивающими сектор

    x,y: integer;
    dy: integer;
    i: integer;
```

```
begin
  if NOT dataok then
    begin
      // данные не загружены
      PaintBox1.Canvas.TextOut(10,10,'Ошибка! Нет файла данных (data.txt)');
      exit;
    end;

  with PaintBox1.Canvas do begin

    // заголовок
    Font.Name := 'Tahoma';
    Font.Size := 14;
    x := (Width - TextWidth(Title)) div 2;
    Brush.Style := bsClear;
    TextOut(x,10,Title);

    // круговая диаграмма
    x1:= 20 ;
    y1 := 50;

    x2 := x1 + 2*R;
    y2 := y1 + 2*R;

    x0 := x1 + R;
    y0 := y1 + R;

    a1 := 0; // первый сектор откладываем от оси OX

    for i := 1 to HB do
    begin
        { Из-за ошибок округления возможна ситуация, когда между первым
          и последним секторами будет небольшой промежуток или последний
          сектор перекроет первый. Чтобы этого не было, зададим,
          что граница последнего сектора совпадает с прямой OX. }
        if (i <> HB) then
            // 100% - 360 градусов; 1% - 3,6 градуса
            a2 := Round(a1 + 3.6 * percent[i])
        else
            a2 := 359;
```

```
        // координата точки начала дуги
        x3 := x0 + Round(R * cos(a2 * PI / 180));
        y3 := y0 + Round(R * sin(a2 * PI / 180));

        // координата точки конца дуги
        x4 := x0 + Round(R * cos(a1 * PI / 180));
        y4 := y0 + Round(R * sin(a1 * PI / 180));

        // если сектор меньше 6 градусов, границу сектора не рисуем
        if (abs(a1-a2) <= 6) then
            Pen.Style := psClear
        else
            Pen.Style := psSolid;

        Brush.Color := cl[i];
        Pie(x1,y1,x2,y2,x3,y3,x4,y4);

        a1 := a2; // следующий сектор рисуем от начала текущего
    end;

    // легенда
    Font.Size := 10;
    dy := TextHeight('a');
    x := x2 + 20;
    y := y1;
    for i := HB downto 1 do
    begin
        Brush.Color := cl[i];
        Rectangle(x,y,x+40,y+dy);
        Brush.Style := bsClear;
        TextOut(x+50, y,
                dTitle[i]+', '+FloatToStrF(percent[i],ffGeneral,2,2)+'%');
        y := y + dy + 10;
    end;
  end;
end;
```

Листинг 4.9. Файл данных (data.txt)

```
Источники энергии
Другие
0.5
```

```
Гидроэлектростанции
2.5
Атомные электростанции
7
Газ
23
Уголь
24
Нефть
40
```

Точка

Свойство `Pixels` объекта `Canvas` представляет собой двумерный массив типа `TColor`, который содержит информацию о цвете точек графической поверхности. Значением элемента массива `Pixels` является код цвета точки (значение типа `TColor`). Первый индекс элемента массива определяет горизонтальную координату точки, второй — вертикальную. Размер массива `Pixels` соответствует размеру графической поверхности.

Элемент `Pixels[0,0]` соответствует левой верхней точке графической поверхности, элемент `Pixels[Canvas.Width-1,Canvas.Height-1]` — правой нижней.

Присвоив значение элементу массива `Pixels`, можно изменить цвет точки графической поверхности, т. е. "нарисовать" или "стереть" точку. Например, инструкция

```
PaintBox1.Canvas.Pixels[10,10] := clRed;
```

окрашивает точку поверхности в красный цвет.

Битовые образы

Для формирования сложных изображений используют *битовые образы*. Битовый образ — это небольшая картинка, которая находится в памяти компьютера.

Битовый образ можно загрузить из bmp-файла или из *ресурса*, а также сформировать путем копирования из другого битового образа или с графической поверхности.

Картинку битового образа (иногда говорят просто "битовый образ") можно подготовить в графическом редакторе или, если предполагается, что битовый образ будет загружен из ресурса программы, с помощью редактора ресурсов, например Borland Resource Workshop. Файл ресурсов можно создать и с помощью утилиты Image Editor.

Битовый образ — это объект типа `TBitmap`. Некоторые свойства класса `TBitmap` приведены в табл. 4.7.

Таблица 4.7. *Свойства класса* `TBitmap`

Свойство	Описание
`Height`, `Width`	Размер (ширина, высота) битового образа. Значения свойств соответствуют размеру загруженной из файла (метод `LoadFromFile`) или ресурса (метод `LoadFromResourceID` или `LoadFromResourceName`) картинки
`Empty`	Признак того, что картинка в битовый образ не загружена (`True`)
`Transparent`	Устанавливает (`True`) режим использования "прозрачного" цвета. При выводе битового образа методом `Draw` элементы картинки, цвет которых совпадает с цветом `TransparentColor`, не выводятся. По умолчанию значение `TransparentColor` определяет цвет левого нижнего пиксела
`TransparentColor`	Задает прозрачный цвет. Элементы картинки, окрашенные этим цветом, методом `Draw` не выводятся
`Canvas`	Поверхность битового образа, на которой можно рисовать точно так же, как на поверхности формы или компонента `Image`

Загрузку битового образа из файла обеспечивает метод `LoadFromFile`, которому в качестве параметра передается имя bmp-файла. Например, следующий фрагмент кода обеспечивает создание и загрузку битового образа из файла plane.bmp.

```
bm := TBitmap.Create;          // создать объект TBitmap
bm.LoadFromFile('plane.bmp'); // загрузить картинку
```

В результате выполнения приведенного фрагмента битовый образ `bm` представляет собой изображение самолета (предполагается, что в файле plane.bmp находится изображение самолета, а не что-либо другое).

После того как битовый образ сформирован (загружен из файла или из ресурса), его можно вывести, например, на графическую поверхность формы или компонента `PaintBox`. Отображение битового образа обеспечивает метод `Draw` объекта `Canvas`. В качестве параметров метода указывают координаты точки, от которой надо вывести битовый образ, и сам битовый образ. Например, инструкция

```
Form1.Canvas.Draw(10,20,bm);
```

выводит на поверхность формы битовый образ `bm` — изображение самолета.

Если свойству `Transparent` битового образа присвоить значение `True`, то фрагменты битового образа, цвет которых совпадает с цветом его левой нижней точки, не будут выведены. Такой прием используется для создания эффекта прозрачного фона. "Прозрачный" цвет можно задать напрямую, присвоив значение свойству `Transparent.Color`.

Загрузку и отображение битовых образов демонстрирует следующая программа (ее окно приведено на рис. 4.14).

Рис. 4.14. Изображения неба и самолета — битовые образы, загруженные из файлов

После запуска программы в ее окне появляется изображение самолета, летящего в облаках. И небо, и самолет — это битовые образы, загруженные из файлов. Загрузку битовых образов выполняет процедура обработки события `Create` формы, отображение — процедура обработки события `Paint`. Объявление битовых образов следует поместить в секцию `private` объявления типа формы. Процедуры обработки событий и объявление типа формы приведены в листинге 4.10.

Листинг 4.10. Загрузка и отображение битовых образов

```
type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    // битовые образы
    Sky: TBitmap;     // небо
    Plane: TBitmap;   // самолет
  public
    { Public declarations }
  end;

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  // создать два объекта TBitmap и загрузить в них картинки
  Sky := TBitmap.Create;
  Sky.LoadFromFile('sky.bmp');
  Plane := TBitmap.Create;
  Plane.LoadFromFile('plane.bmp');
```

```
  Plane.Transparent := True; // прозрачный фон
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Draw(0,0,Sky);       // фон — небо
  Canvas.Draw(120,50,Plane); // объект — самолет
end;
```

Битовый образ можно использовать для формирования фонового рисунка по принципу кафельной плитки (рис. 4.15).

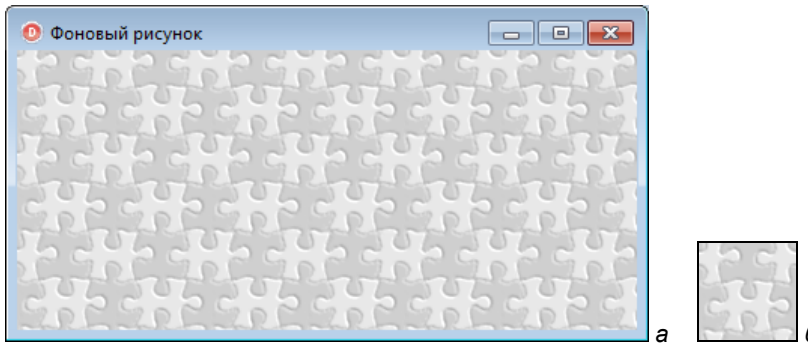


Рис. 4.15. Фоновый рисунок (*а*) и битовый образ (*б*), из которого он составлен

Как сформировать фоновый рисунок путем многократного вывода битового образа на поверхность формы, показывает программа "Фоновый рисунок" (листинг 4.11). Формирование рисунка (многократный вывод образа на поверхность формы) выполняет процедура обработки события `Paint` формы, загрузку битовых образов — конструктор формы.

Листинг 4.11. Фоновый рисунок

```
type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    bm: TBitmap; // "плитка"
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  bm := TBitmap.Create; // создать битовый образ
  try
    // загрузить картинку
    bm.LoadFromFile('back.bmp');
  finally

  end;
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
var
  x,y: integer;  // координаты точки вывода битового образа
begin
  if bm.Empty
      then exit; // битовый образ не загружен

  x:=0; y :=0;
  repeat
     // горизонтальный ряд
     repeat
          Canvas.Draw(x,y,bm);
          x := x + bm.Width;
     until (x > ClientWidth);
     // к следующему ряду
     x := 0;
     y := y + bm.Height;
  until (y > ClientHeight);
end;
```

Мультипликация

Под мультипликацией обычно понимается рисованное изображение, элементы которого движутся.

Изображение объекта можно формировать из графических примитивов во время работы программы или использовать заранее подготовленные битовые образы. Первый подход требует значительных вычислительных ресурсов. Второй подход менее ресурсоемок, поэтому именно он широко используется разработчиками компьютерных игр.

Движение

Реализовать эффект движения объекта не сложно: сначала нужно вывести изображение объекта на графическую поверхность (например, компонента `PaintBox`), затем через некоторое время стереть и снова вывести, но уже на небольшом расстоянии от его первоначального положения. Подбором времени между выводом и удалением изображения, а также расстояния между новым и предыдущим положением объекта, можно добиться того, что у наблюдателя будет складываться впечатление, что объект равномерно движется.

Описанный метод реализации анимации демонстрирует программа "Движение объекта", в ее окне "плывет" корабль (рис. 4.16).

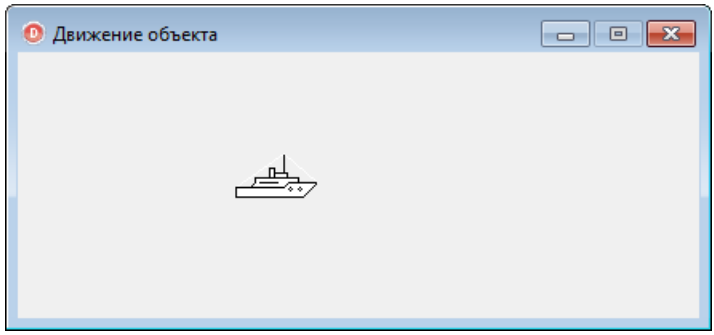


Рис. 4.16. В окне программы "плывет" кораблик

Объект (кораблик) на поверхности формы рисует процедура `Ship`. В качестве параметров она получает координаты *базовой точки* объекта. Базовая точка (x_0, y_0) определяет положение графического объекта в целом, от нее отсчитываются координаты графических примитивов, образующих объект (рис. 4.17). Следует обратить внимание на то, что координаты графических примитивов лучше отсчитывать не в пикселах, а в приращениях. Такой подход позволяет легко масштабировать изображение — чтобы изменить размер объекта, достаточно соответствующим образом изменить шаг сетки.

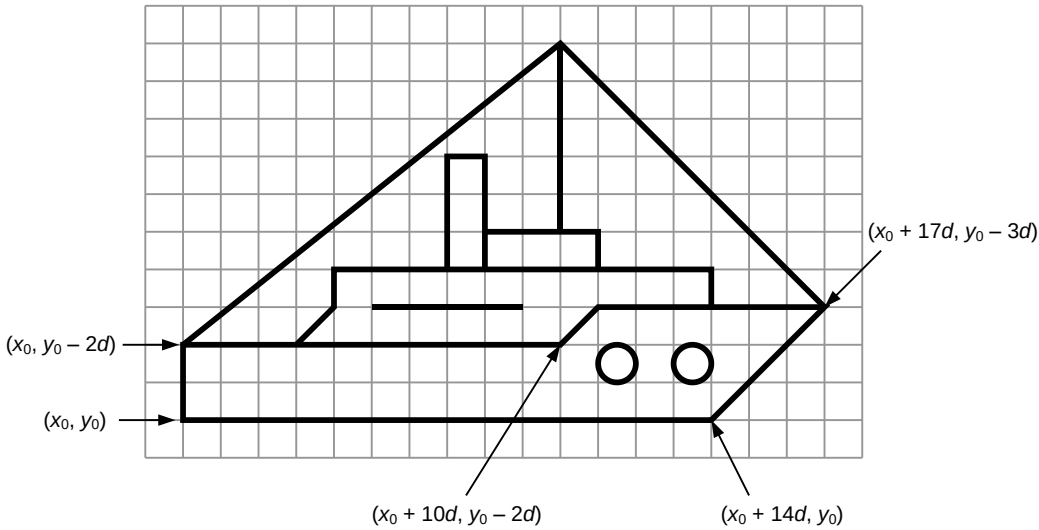


Рис. 4.17. Базовая точка (x_0, y_0) определяет положение объекта

На рис. 4.18 приведена форма программы. Компонент `Timer` обеспечивает генерацию события `Timer`, процедура обработки которого выполняет основную работу: стирает изображение объекта и рисует его на новом месте. Настройку компонента `Timer` обеспечивает процедура обработки события `Create` формы. Эта же процедура задает размер и исходное положение корабля, а также скорость его движения (скорость определяется периодом возникновения события `Timer` и величиной приращения координаты `X`).

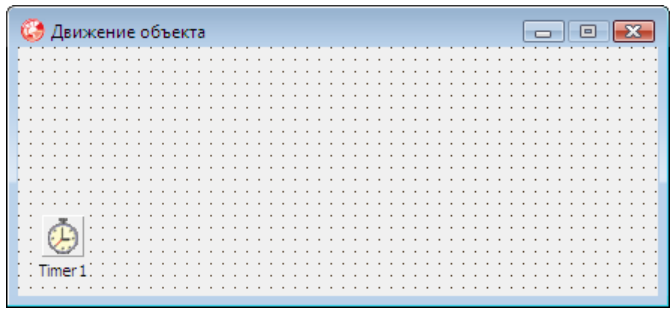


Рис. 4.18. Форма программы

В листинге 4.12 приведена процедура `Ship` и представлены процедуры обработки события `Timer` и события `Create`.

Листинг 4.12. Движение объекта

```
var
  d: integer;    // размер объекта (коэф. масштабирования)
```

```
  x,y: integer; // координаты объекта (базовой точки)
  dx: integer;  // приращение координаты X

procedure Ship(x,y: integer; d: integer);
var
  // корпус и надстройку будем рисовать с помощью метода Polygon
  p1: array[1..7] of TPoint;  // координаты точек корпуса
  p2: array[1..8] of TPoint;  // координаты точек надстройки

  pc,bc: TColor; // текущий цвет карандаша и кисти
begin
  with Form1.Canvas do
  begin

  // сохраним текущий цвет карандаша и кисти
  pc := Pen.Color;
  bc := Brush.Color;

  // установим нужный цвет карандаша и кисти
  Pen.Color := clBlack;
  Brush.Color := clWhite;

  // корпус
  p1[1].x := x;       p1[1].y := y;
  p1[2].x := x;       p1[2].y := y-2*d;
  p1[3].x := x+10*d;  p1[3].y := y-2*d;
  p1[4].x := x+11*d;  p1[4].y := y-3*d;
  p1[5].x := x+17*d;  p1[5].y :=y-3*d;
  p1[6].x := x+14*d;  p1[6].y :=y;
  p1[7].x := x;       p1[7].y :=y;
  Polygon(p1);

  // надстройка
  p2[1].x := x+3*d;   p2[1].y := y-2*d;
  p2[2].x := x+4*d;   p2[2].y := y-3*d;
  p2[3].x := x+4*d;   p2[3].y := y-4*d;
  p2[4].x := x+13*d;  p2[4].y := y-4*d;
  p2[5].x := x+13*d;  p2[5].y := y-3*d;
  p2[6].x := x+11*d;  p2[6].y := y-3*d;
  p2[7].x := x+10*d;  p2[7].y := y-2*d;
  p2[8].x := x+3*d;   p2[8].y := y-2*d;
  Polygon(p2);
```

```
  MoveTo(x+5*d,y-3*d);
  LineTo(x+9*d,y-3*d);

  // капитанский мостик
  Rectangle(x+8*d,y-4*d+1,x+11*d,y-5*d);

  // труба
  Rectangle(x+7*d,y-4*d+1,x+8*d+1,y-6*d);

  // иллюминаторы
  Ellipse(x+11*d,y-2*d,x+12*d,y-1*d);
  Ellipse(x+13*d,y-2*d,x+14*d,y-1*d);

  // мачта
  MoveTo(x+10*d,y-5*d);
  LineTo(x+10*d,y-9*d);

  // оснастка
  Pen.Color := clWhite;
  MoveTo(x+17*d,y-3*d);
  LineTo(x+10*d,y-9*d);
  LineTo(x,y-2*d);

  // восстановим цвет карандаша и кисти
  Pen.Color := pc;
  Brush.Color := bc;
  end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // стереть объект
  Form1.Canvas.Brush.Color := Form1.Color;
  Form1.Canvas.FillRect(rect(x-1,y+1,x+17*d,y-9*d));

  // вычислить координаты
  if x < Form1.ClientWidth then
      x := x +1
    else
      x := -70;

  // нарисовать объект на новом месте
  Ship(x,y,d);

end;
```

```
// конструктор
procedure TForm1.FormCreate(Sender: TObject);
begin
  // размер объекта
  d := 3;

  // начальное положение объекта
  x:= -50;
  y:= 90;

  // значения Timer1.Interval и dx определяют скорость движения объекта
  Timer1.Interval := 20;
  dx := 3;
end;
```

Взаимодействие с пользователем

Программист может позволить пользователю управлять движением объектов в окне программы. Следующая программа показывает, как это сделать.

Программа "ПВО" представляет собой игру, цель которой — уничтожить самолеты противника (рис. 4.19). В начале игры в окне программы отображаются два объекта: летящий самолет и пусковая установка. Игрок, нажав клавишу <Пробел>, может запустить ракету, которая, если момент запуска выбран правильно, собьет самолет. Также, с помощью клавиш перемещения курсора, игрок может сместить пусковую установку влево или вправо. Текст программы приведен в листинге 4.13.

Рис. 4.19. Игра "ПВО"

Основную работу выполняет процедура обработки события `Timer`, которая рисует цель (самолет), пушку (пусковую установку) и снаряд. Сначала она сравнивает

координаты самолета и снаряда. Если координаты самолета и снаряда совпадают, процедура стирает самолет, увеличивает счетчик попаданий и завершает работу. Если снаряд не долетел до самолета, процедура перерисовывает самолет на новом месте. Если ракета запущена (в этом случае значение переменной `dy` равно –1), то процедура рисует ее. Далее процедура проверяет, надо ли перерисовать на новом месте пусковую установку. Если игрок удерживает клавишу перемещения курсора, то процедура перерисовывает пусковую установку со смещением относительно ее текущего положения. Запуск ракеты обеспечивает процедура обработки события `KeyPress`. Она, если нажата клавиша <Пробел>, и если ракета не запущена, записывает в переменную `dy` минус единицу (в результате процедура обработки события `Timer` рисует ракету). Нажатие клавиш перемещения курсора обрабатывает процедура события `KeyDown` (это событие генерируется до тех пор, пока пользователь удерживает клавишу). Процедура, в зависимости от того, какую клавишу удерживает игрок, задает направление перемещения установки (если значение `dx` не равно нулю, процедура обработки события `Paint` рисует установку на новом месте). Если пользователь отпустит клавишу, то возникает событие `KeyUp`, процедура обработки которого записывает в переменную `dx` ноль, в результате установка перестает двигаться.

Листинг 4.13. Игра "ПВО" (управление движением объекта)

```
 var
  // координаты объектов
  us: TPoint;   // установка (пушка)
  sn: TPoint;   // снаряд
  pl: TPoint;   // самолет

  dy: integer; // приращение координаты Y снаряда
  dx: integer; // приращение координаты X установки

  n: integer; // количество выстрелов
  m: integer; // количество попаданий

{$R *.dfm}

// информация
procedure info;
var
  st: string;
begin
  Form1.Canvas.Brush.Color := Form1.Color;
  st := 'Выстрелов: ' + IntToStr(n);
```

```
  Form1.Canvas.TextOut(10,10,st);
  st := 'Попаданий: ' + IntToStr(m);
  Form1.Canvas.TextOut(10,25,st);
end;

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  // исходное положение установки
  us.X := ClientWidth div 2 - 25;
  us.Y := ClientHeight - 20;

  // исходное положение самолета
  pl.X := 0;
  pl.Y := 50;

  Canvas.Pen.Color := Form1.Color;

end;

// клавиша нажата
procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  // следующую ракету можно пустить, если предыдущая улетела
  if (key = ' ') and (dy = 0) then
    begin
        // пуск ракеты
        sn.X := us.X + 25;
        sn.Y := ClientHeight - 30;
        dy := -1;
        n:= n+1;
        Info;
    end;
end;

// клавиша удерживается
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
    case key of
      VK_LEFT : dx := -1;
```

```
      VK_RIGHT: dx :=  1;
    end;
end;

// клавиша отпущена
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
                           Shift: TShiftState);
begin
  if (key = VK_LEFT) or (key = VK_RIGHT) then
      dx := 0;
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin

  // сравнить координаты самолета и снаряда
  if (pl.X > sn.X -5) and (pl.X < sn.X + 5) and
     (pl.Y < sn.Y +5) and (pl.Y > sn.Y -5)  then
  begin
    // попадание, стереть самолет
    Canvas.Brush.Color := Form1.Color;
    Canvas.Rectangle(pl.X-5, pl.Y-5, pl.X +10, pl.Y+10);
    pl.X := -20;
    sn.X := us.X + 25;
    dy:=0;
    m := m +1; // количество попаданий
    info;      // отобразить информацию

    pl.Y := 50 + random(20);
    exit;
  end;

  // стереть самолет
  Canvas.Brush.Color := Form1.Color;
  Canvas.Rectangle(pl.X, pl.Y, pl.X +7, pl.Y+5);

  if pl.X  <  ClientWidth then
      pl.X := pl.X + 1
    else
      begin
          pl.X := - 20;
```

```
          pl.Y := 50 + random(20);
      end;

  // нарисовать самолет на новом месте
  Canvas.Brush.Color := clNavy;
  Canvas.Rectangle(pl.X, pl.Y, pl.X +7, pl.Y+5);

  // снаряд
  if dy < 0 then
    // снаряд летит
    begin
      // стереть снаряд
      Canvas.Brush.Color := Form1.Color;
      Canvas.Rectangle(sn.X, sn.Y, sn.X +4, sn.Y+7);
      if sn.y  > 0 then
      begin
        sn.Y := sn.Y - 1;
        // нарисовать снаряд на новом месте
        Canvas.Brush.Color := clBlack;
        Canvas.Rectangle(sn.X, sn.Y, sn.X +4, sn.Y+7);
      end
      else
        // снаряд долетел до верхней границы окна
        dy := 0;
    end;

   //  установка
   if ((dx < 0) and (us.X > 0)) or ((dx > 0) and (us.X < ClientWidth -
50))then
   begin
        // dx <> 0 — игрок удерживает клавишу
        // "курсор вправо" или "курсор влево"
        Canvas.Brush.Color := Form1.Color;
        Canvas.Rectangle(us.X, us.Y, us.X +50, us.Y+7);
        us.X := us.x + dx;
        // нарисовать установку на новом месте
        Canvas.Brush.Color := clBlack;
        Canvas.Rectangle(us.X, us.Y, us.X +50, us.Y+7);
   end;
end;
```

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    info;
    Canvas.Brush.Color := clBlack;
    Canvas.Rectangle(us.X, us.Y, us.X +50, us.Y+7);
end;
```

Использование битовых образов

В предыдущих примерах изображение объектов формировалось из графических примитивов. Недостаток такого способа очевиден: чтобы сформировать более или менее реалистичную картинку, необходимо обеспечить отображение большого количества графических примитивов, что существенно увеличивает размер кода, снижает скорость работы программы (именно поэтому разработчики компьютерных игр используют специальные библиотеки). Теперь на примере программы "Полет в облаках" рассмотрим, как можно существенно улучшить графику благодаря использованию битовых образов.

Как и в предыдущих программах, эффект движения (полет самолета) достигается вследствие периодической перерисовки объекта с некоторым смещением относительно его прежнего положения. Перед выводом картинки в новой точке предыдущее изображение должно быть удалено. Удалить изображение объекта можно путем перерисовки всей фоновой картинки или только той ее части, которая перекрыта объектом. В рассматриваемой программе используется второй способ.

Форма программы "Полет в облаках" приведена на рис. 4.20, текст — в листинге 4.14.

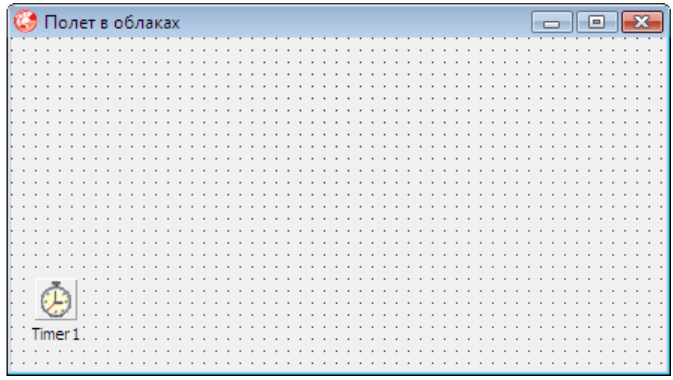


Рис. 4.20. Форма программы "Полет в облаках"

Листинг 4.14. Полет в облаках

```
type
  TForm1 = class(TForm)
```

```
    Timer1: TTimer;
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    back: TBitmap;  // фон
    plane: TBitmap; // объект
    x,y: integer;   // координаты объекта
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// конструктор
procedure TForm1.FormCreate(Sender: TObject);
begin
  try
    back := TBitmap.Create;
    back.LoadFromFile('sky.bmp');
    // установить размер формы в соответствии с размером фонового рисунка
    Form1.ClientWidth := back.Width;
    Form1.ClientHeight := back.Height;

    plane := TBitmap.Create;
    plane.LoadFromFile('plane.bmp');
    plane.Transparent := True;

    // исходное положение объекта
    x :=  -30;
    y :=  70;

    Timer1.Interval := 25;
  finally

  end;
end;
```

```
// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  r: TRect; // область, в которой находится объект
begin
  r := Rect(x,y,x+plane.Width,y+plane.Height);
  Canvas.CopyRect(r,back.Canvas,r); // стереть объект (восстановить фон)
  x := x + 2;
  Canvas.Draw(x,y,plane);

  if x > Form1.Width + plane.Width + 10 then
    x := -20;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Draw(0,0,back);
  Canvas.Draw(x,y,plane);
end;
```

Конструктор (процедура обработки события `Create`) загружает битовые образы (фон и изображение объекта), устанавливает размер формы в соответствии с размером фонового рисунка и задает начальное положение объекта. Следует обратить внимание на то, что начальное значение переменной `x`, которая определяет положение левой верхней точки области вывода изображения объекта — отрицательное число, по модулю больше ширины битового образа объекта. Поэтому в начале работы программы самолет не виден. С каждым сигналом таймера значение координаты `x` увеличивается, и 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. Основную работу (перерисовку объекта) выполняет процедура обработки сигнала таймера (события `Timer`). Сначала она стирает изображение объекта (восстанавливает "испорченную" часть фона), затем выводит изображение объекта на новом месте. Восстановление фона выполняется путем копирования фрагмента битового образа фона в ту область графической поверхности, в которой в данный момент находится объект (рис. 4.21). Копирование фрагмента обеспечивает метод `CopyRect`. Процедура обработки события `Paint` обеспечивает отображение фона в начале работы программы, а также всякий раз после того, как окно программы появляется на экране.

Запустив программу "Полет в облаках", можно заметить, что изображение самолета мерцает. Это объясняется тем, что глаз успевает заметить, как самолет исчез и появился снова. Чтобы устранить мерцание, надо чтобы самолет не исчезал, а смещался. Добиться этого можно, если формировать изображение не на поверхно-

сти формы, а на невидимой для пользователя графической поверхности, и затем выводить готовое изображение на поверхность формы. Рисунок 4.22 поясняет процесс формирования изображения. Сначала фрагмент фона копируется на рабочую поверхность (шаг 1), затем накладывается изображение объекта (шаг 2), после чего сформированное изображение выводится в нужную точку видимой графической поверхности (шаг 3). Приведенная в листинге 4.15 программа демонстрирует реализацию описанного метода формирования изображения.

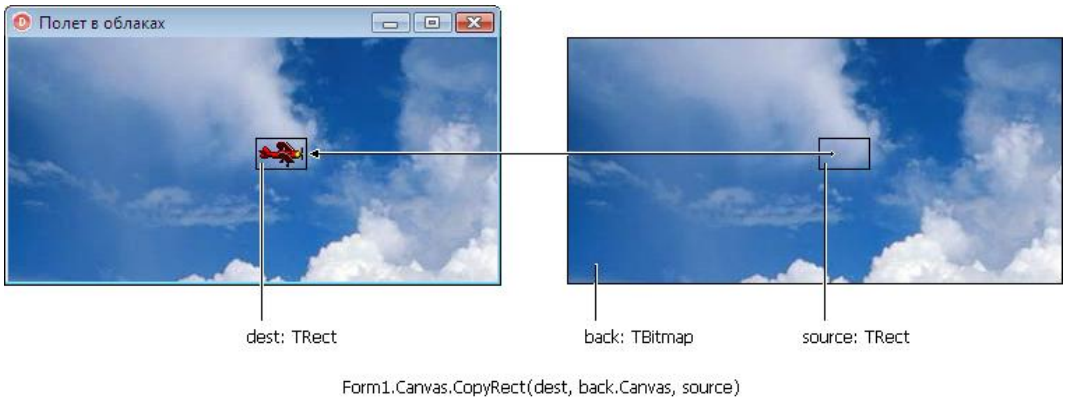


Рис. 4.21. Восстановление фона перед отрисовкой объекта на новом месте обеспечивает метод `CopyRect`

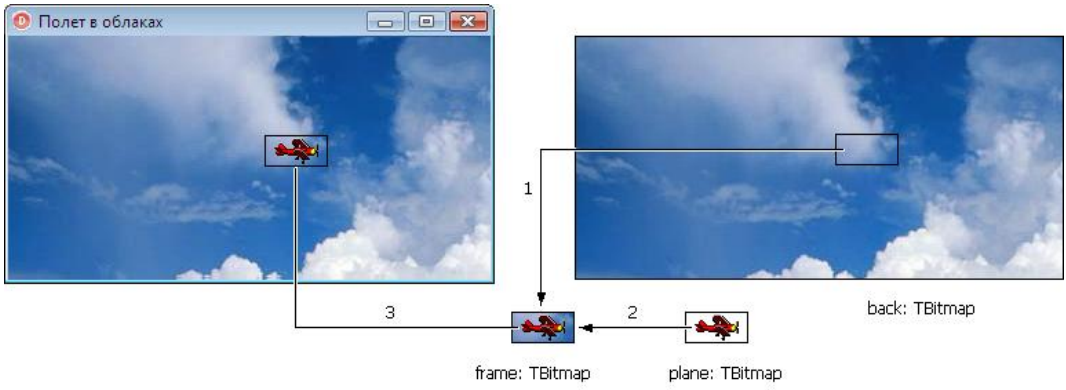


Рис. 4.22. Формирование и отображение кадра

Листинг 4.15. Формирование изображения на невидимой поверхности

```
type
  TForm1 = class(TForm)
    Timer1: TTimer;
```

```
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);

  private
    back: TBitmap;  // фон
    plane: TBitmap; // объект (самолет)
    frame: TBitmap; // кадр = область фона + объект
    x,y: integer;   // координаты объекта

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// конструктор
procedure TForm1.FormCreate(Sender: TObject);
begin
  try
    back := TBitmap.Create;
    back.LoadFromFile('sky.bmp');
    // установить размер формы в соответствии с размером фонового рисунка
    Form1.ClientWidth := back.Width;
    Form1.ClientHeight := back.Height;

    plane := TBitmap.Create;
    plane.LoadFromFile('plane.bmp');
    plane.Transparent := True;

    frame := TBitmap.Create;
    frame.LoadFromFile('plane.bmp');

    // исходное положение объекта
    x :=  20;
    y :=  70;
```

```
    Timer1.Enabled := True;
  finally

  end;
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  source :TRect; // область, откуда надо скопировать фон
  dest: Trect;   // область рабочей поверхности, куда надо скопировать фон
begin
    x := x + 2;

   source := Rect(x,y, x+frame.Width, y+frame.Height);
   dest := Rect(0,0,frame.Width,frame.Height);

  // копировать фрагмент фона
  frame.Canvas.CopyRect(dest, back.Canvas, source);

  // наложить объект
  frame.Canvas.Draw(0,0,plane);

  // отобразить кадр
  Canvas.Draw(x,y,frame);

  if x > Form1.Width + plane.Width + 10 then
    x := -20;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  Canvas.Draw(0,0,back);
  Canvas.Draw(x,y,plane);
end;
```

Глава 5

Мультимедиа

Большинство современных программ являются *мультимедийными*, что подразумевает использование возможности компьютера отображать графику, воспроизводить видео, анимацию, музыку. Типичными примерами мультимедийных программ являются игры и обучающие программы.

Функция *PlaySound*

Для реализации звуковых эффектов, например в играх, весьма удобна функция `PlaySound`. Она позволяет проиграть звуковой фрагмент, находящийся в wav-файле.

Инструкция вызова функции `PlaySound` в общем виде выглядит так:

`PlaySound(`*`WAV-файл`*`, 0, `*`Режим`*`)`

Параметр *`WAV-файл`* задает звуковой файл, параметр *`Режим`* — режим воспроизведения: синхронный или асинхронный. Если задан синхронный режим воспроизведения, то функция `PlaySound` возвращает управление программе сразу после того, как будет инициирован процесс воспроизведения звука. Если задан асинхронный режим, то программа, вызвавшая функцию `PlaySound`, продолжит работу только после того, как завершится воспроизведение звукового файла. В качестве значения параметра *`Режим`* можно указать именованную константу `SND_SYNC` (синхронный режим) или `SND_ASYNC` (асинхронный режим).

Например, инструкция

```
PlaySound('ringin.wav',0,SND_ASYNC);
```

активизирует процесс воспроизведения файла ringin.wav.

Для того чтобы функция `PlaySound` стала доступной, в директиву `uses` надо включить ссылку на модуль `mmsystem`.

В качестве примера использования функции `PlaySound` в листинге 5.1 приведен фрагмент программы "Будильник" — процедура обработки сигнала таймера. Когда наступает время, на которое установлен будильник, на экране появляется окно с сообщением. Появление окна сопровождается звуком ringin.wav. Так как задан асинхронный режим воспроизведения, то окно появляется сразу после начала воспроизведения звукового файла.

Листинг 5.1. Использование функции `PlaySound`

```
// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if CompareTime(Now,AlarmTime) >= 0 then
    begin
        Timer1.Enabled := False;
        if CheckBox1.Checked then
            // звуковой сигнал
            PlaySound('ringin.wav',0,SND_ASYNC);
        ShowMessage(FormatDateTime(' hh:nn - ', Now) +  Edit3.Text);
        Form1.Show; // отобразить (развернуть) окно
    end;
end;
```

Следует обратить внимание на то, что если в имени wav-файла путь не указан, то функция `PlaySound` сначала будет искать звуковой файл в текущем каталоге (в каталоге, из которого запущена программа), затем в каталоге C:\Windows\Media. Если ни в одном из этих каталогов нужного файла нет, то будет воспроизведен так называемый стандартный звук (задается в настройках Windows). Программист может запретить воспроизведение стандартного звука. Для этого в качестве параметра *Режим* надо указать константу `SND_NODEFAULT`.

Компонент *MediaPlayer*

Компонент `MediaPlayer` обеспечивает воспроизведение звуковых файлов различных форматов (WAV, MIDI, MP3), компакт-дисков, видеороликов (AVI) и сопровождаемой звуком анимации.

Значок компонента `MediaPlayer` (рис. 5.1) находится на вкладке **System**.

Рис. 5.1. Значок компонента `MediaPlayer`

Внешне компонент `MediaPlayer` представляет собой группу кнопок (рис. 5.2), подобных тем, которые можно видеть на аудио- или видеоплеере. Назначение этих кнопок пояснено в табл. 5.1. Свойства компонента `MediaPlayer`, доступные во время разработки формы, приведены в табл. 5.2.

Рис. 5.2. Компонент `MediaPlayer`

Таблица 5.1. *Кнопки компонента* `MediaPlayer`

Кнопка		Обозначение	Действие
	**Воспроизведение**	`btPlay`	Воспроизведение звука или видео
	**Пауза**	`btPause`	Приостановка воспроизведения
	**Стоп**	`btStop`	Остановка воспроизведения
	**Следующий**	`btNext`	Переход к следующему кадру
	**Предыдущий**	`btPrev`	Переход к предыдущему кадру
	**Шаг**	`btStep`	Переход к следующему звуковому фрагменту, например к следующему треку (композиции) на CD
	**Назад**	`btBack`	Переход к предыдущему звуковому фрагменту, например к предыдущей песне на CD
	**Запись**	`btRecord`	Активизирует процесс записи
	**Открыть**	`btEject`	Открывает CD-дисковод компьютера

Таблица 5.2. *Свойства компонента* `MediaPlayer`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента и управления работой плеера
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент `MediaPlayer`. Тип устройства задается именованной константой: `dtAutoSelect` — тип устройства определяется автоматически по расширению файла; `dtWaveAudio` — проигрыватель звука; `dtAVIVideo` — видеопроигрыватель; `dtCDAudio` — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
`AutoOpen`	Признак автоматической загрузки сразу после запуска программы, файла видеоролика или звукового фрагмента
`Display`	Определяет компонент, поверхность которого используется в качестве экрана для воспроизведения видеоролика (обычно в качестве экрана для отображения видео используют компонент `Panel`)
`VisibleButtons`	Составное свойство. Определяет видимые кнопки компонента

Помимо свойств, доступных в процессе разработки формы, компонент `MediaPlayer` предоставляет свойства, доступные во время работы программы (табл. 5.3), которые позволяют получить информацию о состоянии медиаплеера, воспроизводимом файле или треке Audio CD. Следует обратить внимание, что значения свойств, содержащих информацию о длительности, могут быть представлены в различных форматах. Наиболее универсальным форматом является формат `tfMilliseconds`, в котором длительность выражается в миллисекундах. Некоторые устройства поддерживают несколько форматов. Например, если `MediaPlayer` используется для воспроизведения CD, то информация о воспроизводимом треке может быть представлена в формате tfTMSF (Track, Minute, Second, Frame — трек, минута, секунда, кадр). Для преобразования миллисекунд в минуты и секунды надо воспользоваться известными соотношениями. Если значение свойства представлено в формате tfTMSF, то для преобразования можно использовать функции `MCI_TMSF_TRACK`, `MCI_TMSF_SECOND` и `MCI_TMSF_MINUTE`.

Таблица 5.3. *Свойства компонента* `MediaPlayer`*, доступные во время работы программы*

Свойство	Описание
`Length`	Длина (время, необходимое для воспроизведения) открытого файла (например, WAV или AVI) или всех треков Audio CD
`Tracks`	Количество треков на открытом устройстве (количество композиций на Audio CD)
`TrackLength`	Длина (длительность) треков. Свойство представляет собой массив, каждый элемент которого содержит информацию о длине трека (времени воспроизведения)
`Position`	Позиция (время от начала) в процессе воспроизведения трека
`TimeFormat`	Формат представления значений свойств `Length`, `TrackLength` и `Position`. Наиболее универсальным является формат `tfMilliseconds`. Если медиаплеер представляет собой CD-проигрыватель, то удобно использовать формат tfTMSF
`Mode`	Состояние устройства воспроизведения. Устройство может быть в состоянии воспроизведения (`mpPlaying`). Процесс воспроизведения может быть остановлен (`mpStopped`) или приостановлен (`mpPaused`). Устройство может быть не готово к работе (`mpNotReady`) или в устройстве (CD-дисководе) может отсутствовать носитель (`mpOpen`)
`Display`	Экран — поверхность, на которой отображается клип. Если значение свойства не задано, то клип отображается в отдельном, создаваемом во время работы программы окне
`DisplayRect`	Размер и положение области отображения клипа на поверхности экрана

Компонент `MediaPlayer` предоставляет методы (табл. 5.4), используя которые можно управлять работой медиаплеера из программы так, как будто это делает пользователь.

Таблица 5.4. *Методы компонента* `MediaPlayer`

Метод	Действие
`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 Audio CD
`Previous`	Переход к предыдущему треку, например к следующей композиции на Audio CD
`Step`	Переход к следующему кадру
`Back`	Переход к предыдущему кадру

Использование компонента `MediaPlayer` для воспроизведения MP3-файлов демонстрирует программа "Простой MP3-плеер" (ее форма приведена на рис. 5.3). Значения свойств компонента `MediaPlayer` приведены в табл. 5.5, модуль формы — в листинге 5.2.

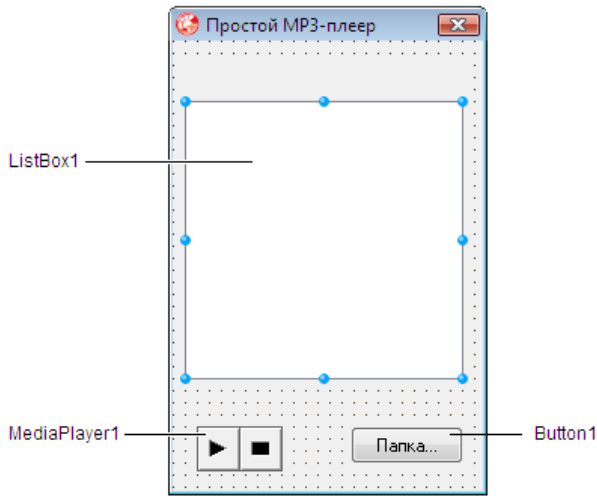


Рис. 5.3. Окно программы "Простой MP3-плеер"

Таблица 5.5. *Значения свойств компонента* `MediaPlayer`

Свойство	Значение
DeviceType	dtAutoSelect
VisibleButtons.btPlay	True
VisibleButtons.btPause	False
VisibleButtons.btStop	True
VisibleButtons.btNext	False
VisibleButtons.btPrev	False
VisibleButtons.btStep	False
VisibleButtons.btBack	False
VisibleButtons.btRecord	False
VisibleButtons.btEject	False

Листинг 5.2. Простой MP3-плееер

```
unit smp3m;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, MPlayer, ComCtrls,
  MMSYSTEM, FileCtrl; // эти ссылки вставлены вручную

type
  TForm1 = class(TForm)

    ListBox1: TListBox;
    MediaPlayer1: TMediaPlayer;
    Label1: TLabel;
    Button1: TButton;

    procedure FormCreate(Sender: TObject);
    procedure ListBox1Click(Sender: TObject);

    procedure MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
                                var DoDefault: Boolean);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
```

```
  private
    { Private declarations }

    procedure PlayList(Path: string);     // формирует список MP3-файлов
    procedure SelectFolder(Root:String); // выбор папки
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

var
  SoundPath: string[255]; // путь к MP3-файлам

// формирует список MP3-файлов
procedure TForm1.PlayList(Path: string);
var
  lpBuf: PChar;          // указатель на nul-terminated строку
  sWinDir: string[128];  // обычная Паскаль-строка

  SearchRec: TSearchRec; // структура SearchRec содержит информацию
                         // о файле, удовлетворяющем условию поиска
begin
  ListBox1.Clear;
  Label1.Caption := '';

  // сформировать список MP3-файлов
  if FindFirst(Path + '*.mp3', faAnyFile, SearchRec) =0 then
     begin
          // В каталоге есть файл с расширением mp3.
          // Добавим имя этого файла в список
          ListBox1.Items.Add(SearchRec.Name);
          // пока в каталоге есть другие файлы с расширением mp3
          while (FindNext(SearchRec) = 0) do
            begin
              ListBox1.Items.Add(SearchRec.Name);
            end;
```

```
          ListBox1.ItemIndex := 0;
     end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   PlayList(''); // список файлов, находящихся в каталоге программы
   ListBox1.ItemIndex := 0;
   Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];
   MediaPlayer1.FileName := SoundPath + ListBox1.Items[ListBox1.itemIndex];
   MediaPlayer1.Open;
end;

// щелчок на кнопке медиаплеера
procedure TForm1.MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
                                   var DoDefault: Boolean);
begin
  case Button of
    btPlay:
    // нажата кнопка Play
    begin
        if  ListBox1.Items.Count <> 0   then
        begin
          MediaPlayer1.FileName :=
              SoundPath + ListBox1.Items[ListBox1.itemIndex];
          MediaPlayer1.Open;
          MediaPlayer1.Play;
        end;
        Dodefault:=False;
    end;
  end;
end;

// щелчок на названии композиции произведения
procedure TForm1.ListBox1Click(Sender: TObject);
begin
   if MediaPlayer1.Mode = mpPlaying then
      begin
        // остановить воспроизведение
        MediaPlayer1.Stop;
      end;
```

```
    // вывести в поле Label1 имя выбранного файла
    Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];
end;

// выбрать папку и сформировать список mp3-файлов
procedure Tform1.SelectFolder(Root:String);
  var
    pwRoot : PWideChar;
    Dir: string;
begin
  // Root — корневой каталог
  GetMem(pwRoot, (Length(Root)+1) * 2);
  pwRoot := StringToWideChar(Root,pwRoot,MAX_PATH*2);
  if SelectDirectory('Укажите папку, в которой находятся MP3-файлы',
                     pwRoot, Dir)
     then
        begin
          // пользователь выбрал папку
          Dir := Dir+'\';
          // каталог, в котором находятся MP3-файлы, выбран
          SoundPath := Dir;
          // сформировать список MP3-файлов,
          // находящихся в папке, указанной пользователем
          PlayList(SoundPath);
        end
     else
       // в окне "Выбор папки" пользователь нажал кнопку "Отмена"
       Dir :='';
end;

// щелчок на кнопке "Папка"
procedure TForm1.Button1Click(Sender: TObject);
begin
   begin
      if MediaPlayer1.Mode = mpPlaying then
        begin
          MediaPlayer1.Stop;
        end;
      SelectFolder(SoundPath);
    end;
end;
```

```
// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  try
    MediaPlayer1.Close;
  finally

  end;
end;

end.
```

Приведенная программа показывает, что работой медиаплеера можно управлять не только с помощью командных кнопок, но и программно. Так, например, щелчок на названии композиции в списке `ListBox` (обработку этого события выполняет процедура `ListBox1Click`) путем вызова метода `Stop` останавливает воспроизведение текущей композиции (в режиме воспроизведения значения свойства `Mode` равно `mpPlaying`). Следует обратить внимание на то, что в конце работы программы, использующей компонент `MediaPlayer`, медиаплеер надо "выключить" — вызвать метод `Close` (см. процедуру `FormClose`).

Воспроизведение MIDI

В компьютерных играх, в обучающих и других программах, а также в качестве звуковых сигналов мобильных телефонов широко используется "электронная", или MIDI-, музыка. MIDI — это сокращение названия интерфейса (Musical Instrument Digital Interface), способа подключения электронных музыкальных инструментов к компьютеру для записи звука. В компьютере MIDI-музыка хранится в MIDI-файлах.

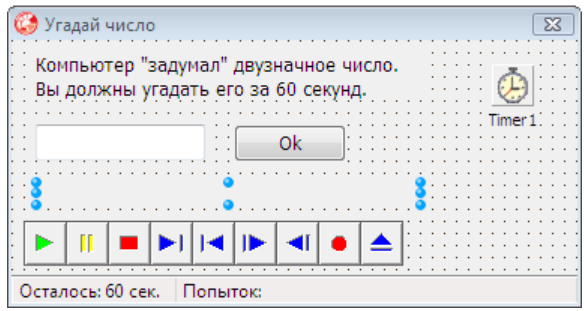


Рис. 5.4. Форма программы "Угадай число" (воспроизведение MIDI)

Компонент `MediaPlayer` позволяет проигрывать MIDI-файлы. В качестве примера рассмотрим программу "Угадай число", в которой компонент `MediaPlayer`

обеспечивает воспроизведение фоновой мелодии. MIDI-музыка начинает звучать сразу после запуска программы. Форма программы "Угадай число" приведена на рис. 5.4, текст — в листинге 5.3. Настройка компонента `MediaPlayer` выполняется программно.

Листинг 5.3. Угадай число (воспроизведение MIDI)

```
var
  comp: integer;       // секретное число
  rem: integer = 60;   // остаток времени на выполнение задания
  n: integer = 0;      // сделано попыток


// начало работы программы
procedure TForm1.FormCreate(Sender: TObject);
var
  k: integer; // номер мелодии
begin
    // управление плеером осуществляет программа,
    // поэтому сделаем его кнопки невидимыми
    MediaPlayer1.Visible := False;

    Randomize;
    k := Random(3) +1;
    case k of
      1: MediaPlayer1.FileName := 'pinkpanter.mid';
      2: MediaPlayer1.FileName := 'pulp_fiction.mid';
      3: MediaPlayer1.FileName := 'mission_impossible.mid';
    end;

    // Если имя файла указано неверно, то при попытке загрузить
    // файл и активизировать процесс воспроизведения
    // возникает исключение EMCIDeviceError
    try
      MediaPlayer1.Open;
      MediaPlayer1.Play;
    except
        on e: exception do
        // ShowMessage(e.Message);
        // обработка исключения заключается в игнорировании ошибки
    end;
```

```
    // сгенерировать число
    Randomize;
    comp := (Random(8)+1) * 10 + Random(9);
end;

// сигнал от медиаплеера
procedure TForm1.MediaPlayer1Notify(Sender: TObject);
begin
  { Если плеер воспроизводит файл и значение свойства Notify равно True
    (метод Play присваивает свойству Notify значение True), то в момент
    окончания воспроизведения возникает событие Notyfy. }

    if Timer1.Enabled  then
        // Длительность мелодии меньше времени, отведенного
        // на решение задачи. Проиграть еще раз
        MediaPlayer1.Play;
end;

// нажатие клавиши в поле редактирования
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    '0'..'9': if Length(Edit1.Text) = 2 then Key := #0;
    #13: ItsOk; // проверить, угадал ли игрок число
    #8: ;
    else Key := #0;
  end;
end;

// проверяет, угадал ли игрок число
procedure TForm1.ItsOk;
var
  igrok : integer;
begin
  n := n +1;
  igrok := StrToInt(Edit1.Text);
  if  igrok = comp then
    begin
      // игрок угадал число
      Edit1.Enabled := False;
      Button1.Enabled := False;
```

```
      Timer1.Enabled := False;
      MediaPlayer1.Stop;
      Label2.Caption := 'ПРАВИЛЬНО!';
      ShowMessage('Поздравляю! Вы справились с поставленной задачей за '+
                  IntToStr(60 - rem) + ' сек.');
    end
  else
    begin
      if igrok < comp then
         Label2.Caption := 'БОЛЬШЕ'
      else
         Label2.Caption := 'МЕНЬШЕ';
      StatusBar1.Panels[1].Text := 'Попыток: '+ IntToStr(n);
      Edit1.SetFocus; // установить курсор в поле ввода
    end;
end;

// щелчок на кнопке Ok
procedure TForm1.Button1Click(Sender: TObject);
begin
  ItsOk;  // проверить, угадал ли игрок число
end;

// изменился текст в поле ввода
procedure TForm1.Edit1Change(Sender: TObject);
begin
  if Length(Edit1.Text) = 2 then
    Button1.Enabled := True
  else
    Button1.Enabled := False;
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  rem := rem -1;
  StatusBar1.Panels[0].Text := 'Осталось: ' + IntToStr(rem) + ' сек.';
  if rem = 0 then
  begin
    // время, отведенное на решение задачи, истекло
    Edit1.Enabled := False;
```

```
    Button1.Enabled := False;
    Timer1.Enabled := False;
    MediaPlayer1.Stop;
    Label2.Caption := 'Вы не справились с задачей!';
  end;
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Timer1.Enabled := False;

  if MediaPlayer1.Mode = mpPlaying then
  begin
    MediaPlayer1.Stop;
    MediaPlayer1.Close;
  end;
end;
```

Настройку плеера выполняет процедура обработки события `Create` формы. Она же активизирует процесс воспроизведения музыки. Следует обратить внимание, что мелодия воспроизводится "по кругу" до тех пор, пока игрок не угадает число или не истечет время, отведенное на решение задачи. Процесс повторного воспроизведения музыки активизирует процедура обработки события `Notify` медиаплеера. Это событие возникает всякий раз, когда по какой-либо причине состояние плеера меняется, например, в момент завершения воспроизведения файла. Процедура обработки события `Click` на кнопке **Ok** также управляет работой плеера. Она, если игрок угадал число, останавливает процесс воспроизведения музыки. Также необходимо обратить внимание на процедуру обработки события `Close` формы. Она проверяет состояние плеера и, если плеер воспроизводит музыку, останавливает его.

Проигрыватель Audio CD

Следующий пример показывает, как на основе компонента `MediaPlayer` можно создать вполне приличный проигрыватель компакт-дисков. Форма программы приведена на рис. 5.5, значения свойств компонетов — в табл. 5.6. Помимо компонентов, которые показаны на рисунке, на форме есть компонент `MediaPlayer` и еще две кнопки `SpeedButton`. Кнопки `SpeedButton1`—`SpeedButton3` используются для управления работой плеера, а кнопки `SpeedButton4` и `SpeedButton5` хранят картинки (битовые образы) **Play** и **Stop**. Во время работы программы, в момент активизации процесса воспроизведения, битовый образ кнопки `SpeedButton5` копируется в битовый образ кнопки `SpeedButton2` (в результате на кнопке появ-

ляется значок **Stop**). Если процесс воспроизведения активен, то в момент щелчка на кнопке `SpeedButton2` битовый образ кнопки `SpeedButton4` копируется в битовый образ кнопки `SpeedButton2` (в результате на кнопке появляется значок **Play**).

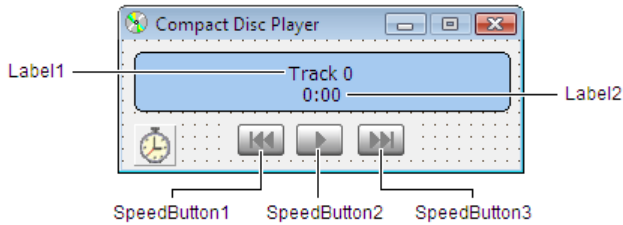


Рис. 5.5. Форма программы Compact Disc Player

Таблица 5.6. *Значение свойств компонентов*

Компонент	Свойство	Значение
`MediaPlayer`	`DeviceType`	`dtCDAudio`

`SpeedButton1`	`NumGlyphs`
	`Glyph`
	`Flat`
	`Enabled`
		`2`
		 
		`True`
		`False`

`SpeedButton2`	`NumGlyphs`
	`Glyph`
	`Flat`
	`Enabled`
		`2`
		 
		`True`
		`False`

`SpeedButton3`	`NumGlyphs`
	`Glyph`
	`Flat`
	`Enabled`
		`2`
		 
		`True`
		`False`

`SpeedButton4`	`NumGlyphs` `Glyph`	`2` 
`SpeedButton5`	`NumGlyphs` `Glyph`	`2` 
`Timer1`	`Interval`	`500`

Компонент `Timer` используется для организации цикла опроса состояния медиаплеера. Во время воспроизведения Audio CD функция обработки события `Timer`, которое генерирует таймер, выводит на индикатор (в поля компонентов `Label1` и `Label2`) номер воспроизводимого трека, его длительность и время воспроизведения.

Вид окна программы сразу после ее запуска, когда в CD-дисководе находится Audio CD, приведен на рис. 5.6. Если в дисководе диска нет или если диск не звуковой, то на индикаторе отображается сообщение **Вставьте Audio CD**. Щелчок на кнопке **Play** (`SpeedButton2`) активизирует процесс воспроизведения. Во время воспроизведения на индикаторе отражается информация о воспроизводимом треке (рис. 5.7).

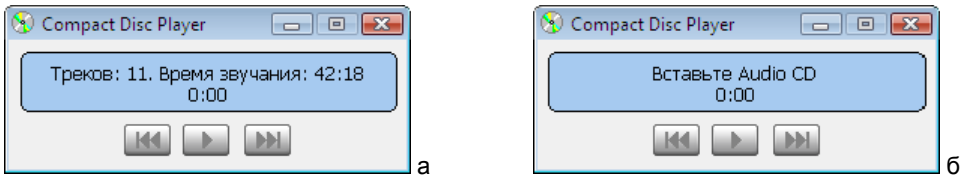


Рис. 5.6. В начале работы на индикаторе выводится информация о времени воспроизведения CD (*а*) или сообщение о необходимости вставить в дисковод Audio CD (*б*)

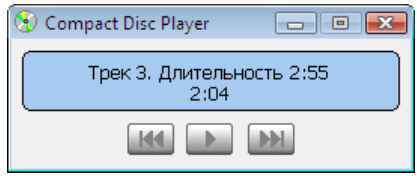


Рис. 5.7. Во время воспроизведения на индикаторе отображается информация о воспроизводимом треке

Текст программы приведен в листинге 5.4. Следует обратить внимание на событие `Notify`, которое может генерировать `MediaPlayer`. Это событие возникает в момент изменения состояния плеера (например, в момент активизации процесса воспроизведения) при условии, что значение свойства `Notify` равно `True`. В рассматриваемой программе событие `Notify` используется для обнаружения факта открытия CD-дисковода пользователем.

Листинг 5.4. Проигрыватель компакт-дисков

```
unit CDp_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
```

```
  Forms, Dialogs, StdCtrls, Buttons, MPlayer, ExtCtrls,
  MMSYSTEM; // обеспечивает управление работой медиаплеера

type
  TForm1 = class(TForm)
    MediaPlayer: TMediaPlayer;
    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    Shape1: TShape;
    SpeedButton1: TSpeedButton;  // кнопка "Предыдущий трек"
    SpeedButton3: TSpeedButton;  // кнопка Play/Stop
    SpeedButton2: TSpeedButton;  // кнопка "Следующий трек"
    SpeedButton4: TSpeedButton;  // хранит картинку Play
    SpeedButton5: TSpeedButton;  // хранит картинку Stop

    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure MediaPlayerNotify(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    procedure TrackInfo; // выводит информацию о треке
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

var
  Track: integer =0; // воспроизводимый трек

// выводит информацию о треке
procedure  TForm1.TrackInfo;
```

```
var
  ms: longint;   // длительность трека в миллисекундах
  min,sec: byte; // длительность трека: минут, секунд
begin
   MediaPlayer.TimeFormat := tfMilliseconds;
   ms := MediaPlayer.TrackLength[Track];
   MediaPlayer.TimeFormat := tfTMSF;

   ms := ms div 1000;
   min := ms div 60;
   sec := ms mod 60;

   Label1.Caption :=
        Format('Трек %d. Длительность %d:%.2d',[Track, min,sec]);
end;

// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  MediaPlayer.Open;
  { Проверить, есть ли в дисководе диск Audio CD.
    Если в дисководе есть диск, то свойство Mode = mpStopped.
    Если диска нет или дисковод открыт, то Mode = mpOpen.
    Тип диска можно определить по количеству треков.
    Если диск — CD-ROM, то на нем один трек,
    на аудиодиске количество треков больше одного. }
  if (MediaPlayer.Mode = mpStopped) and
     (MediaPlayer.Tracks > 1)
  then
    MediaPlayer.Notify := True
  else Timer1.Enabled := True;
end;

// сигнал от MediaPlayer
procedure TForm1.MediaPlayerNotify(Sender: TObject);
begin
  case MediaPlayer.Mode of
    mpOpen: // пользователь открыл дисковод
       begin
        SpeedButton2.Tag := 0;
        SpeedButton2.Enabled := False;
```

```
        SpeedButton2.Enabled := False;
        Timer1.Enabled := True;
        Label2.Caption := '00:00';
      end;
  end;
  MediaPlayer.Notify := True;
end;

// щелчок на кнопке "Предыдущий трек"
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    MediaPlayer.Previous; // в начало текущего трека
    MediaPlayer.Previous; // в начало предыдущего трека
    if MCI_TMSF_TRACK(MediaPlayer.Position) = 1 then
      SpeedButton1.Enabled := False;
    if not SpeedButton3.Enabled
      then SpeedButton3.Enabled := True;
end;

// щелчок на кнопке Play/Stop
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
  trk: integer; // номер трека
begin
  if SpeedButton2.Tag = 0 then
  begin
    // щелчок на кнопке Play
    MediaPlayer.Play;
    MediaPlayer.TimeFormat := tfTMSF;
    trk := MCI_TMSF_TRACK(MediaPlayer.Position);
    if trk > 1 then
        SpeedButton1.Enabled := True;
    if trk < MediaPlayer.Tracks then
        SpeedButton3.Enabled := True;

    MediaPlayer.Notify := False;

    Timer1.Enabled :=True;
    SpeedButton2.Tag := 1;
    // на кнопке SpeedButton5 картинка Stop
    SpeedButton2.Glyph := SpeedButton5.Glyph;
  end
```

```
  else
  begin // щелчок на кнопке Stop
    SpeedButton1.Enabled := False;
    SpeedButton3.Enabled := False;
    MediaPlayer.Notify := True;
    MediaPlayer.Stop;
    Timer1.Enabled := False;
    SpeedButton2.Tag := 0;
    // на кнопке SpeedButton4 картинка Play
    SpeedButton2.Glyph := SpeedButton4.Glyph;
   end;
end;

// щелчок на кнопке "Следующий трек"
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  MediaPlayer.Next;
  // если перешли к последнему треку, то кнопку
  // Next сделать недоступной
  if MCI_TMSF_TRACK(MediaPlayer.Position) = MediaPlayer.Tracks
    then SpeedButton3.Enabled := False;
  if not SpeedButton1.Enabled
    then SpeedButton1.Enabled := True;
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  trk,            // трек
  ms: longint;    // время звучания в миллисекундах
  min, sec: byte; // время
begin
  case MediaPlayer.Mode of

    mpPlaying: // режим воспроизведения
      begin
        // вывести номер трека и время воспроизведения
        trk := MCI_TMSF_TRACK(MediaPlayer.Position);
        min := MCI_TMSF_MINUTE(MediaPlayer.Position);
        sec := MCI_TMSF_SECOND(MediaPlayer.Position);
```

```
      Label2.Caption := Format('%d:%.2d',[min,sec]);
      if trk <> Track then
         begin
           // начало воспроизведения нового трека
           Track := trk;  TrackInfo;
           if Track = 2
              then SpeedButton2.Enabled := True;
           if Track = MediaPlayer.Tracks
               then SpeedButton3.Enabled := False;
         end;
   end;

mpStopped:
   begin
      // Если дисковод открыт или в нем нет
      // Audio CD, то Mode = mpOpen
      if (MediaPlayer.Tracks > 1) then
         begin
           // В дисководе Audio CD. Вывести информацию о диске
           Timer1.Enabled := False;
           MediaPlayer.Open;
           MediaPlayer.Notify := True;

           trk := MCI_TMSF_TRACK(MediaPlayer.Tracks);

           MediaPlayer.TimeFormat := tfMilliseconds;
           ms := MediaPlayer.Length;
           ms := ms div 1000;

           min := ms div 60;
           sec := ms mod 60;

           SpeedButton2.Enabled := True; // кнопка Play

          Label1.Caption := Format('Треков: %d. Время звучания: %d:%.2d',
                                   [trk,min,sec]);
           Label1.Visible := True;
         end;
   end;
```

```
      mpOpen: // дисковод открыт или в дисководе нет аудиодиска
       begin
           Label1.Caption := 'Вставьте Audio CD';
           Label1.Visible := not Label1.Visible;
       end;
  end;
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  MediaPlayer.Stop;
end;

end.
```

Просмотр видеороликов

Компонент `MediaPlayer` позволяет просматривать видеоролики и сопровождаемую звуком анимацию. В качестве примера использования компонента для решения этой задачи рассмотрим программу Video Player (рис. 5.8), с помощью которой можно просмотреть небольшой ролик или анимацию. Форма программы приведена на рис. 5.9, значения свойств компонентов — в табл. 5.7.

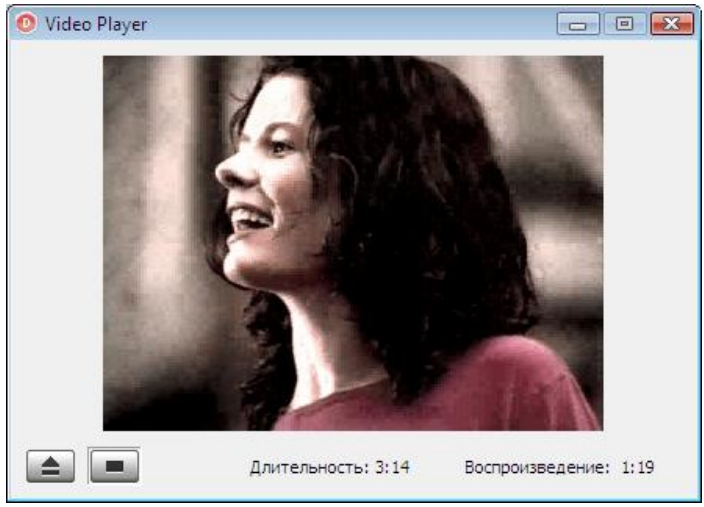


Рис. 5.8. Окно программы Video Player

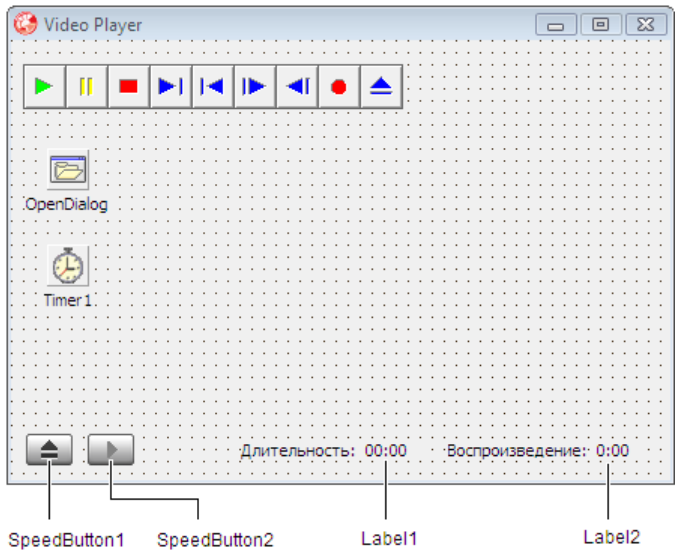


Рис. 5.9. Форма программы Video Player

Таблица 5.7. *Значения свойств компонентов*

Компонент	Свойство	Значение
MediaPlayer	DeviceType	dtAutoSelect
	Visible	False
SpeedButton1	NumGlyphs	4
	Glyph	
	Flat	True
	Enabled	True
SpeedButton2	NumGlyphs	4
	Glyph	
	Flat	True
	Enabled	False
	GroupIndex	1
Timer1	Interval	100
	Enabled	False

Компонент `OpenDialog1` обеспечивает отображение стандартного диалогового окна **Открыть файл** для выбора файла. Окно **Открыть файл** становится доступ-

ным во время работы программы в результате щелчка на кнопке **Eject** (`SpeedButton1`). Следует обратить внимание, что для управления процессом воспроизведения кнопки компонента `MediaPlayer1` не используются, поэтому свойству `Visible` компонента `MediaPlayer` присвоено значение `False`. Также необходимо обратить внимание на свойство `GroupIndex` кнопки `SpeedButon1`. Его значение равно единице, поэтому после щелчка кнопка остается в зафиксированном, нажатом, состоянии и на ее поверхности появляется значок **Stop** (значение свойства `NumGlyhts` равно четырем, это значит, что в битовом образе есть картинка для нажатого состояния). Компонент `Timer` обеспечивает обновление информации на индикаторе: процедура обработки сигнала таймера (события `Timer`) выводит в поле `Label2` информацию о времени воспроизведения клипа.

Тест программы приведен в листинге 5.5.

Листинг 5.5. Video Player

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   MediaPlayer.Display := Form1;

   // Это можно сделать во время создания формы. Но на всякий случай...
   SpeedButton1.GroupIndex := 1;
   SpeedButton1.AllowAllUp := True;
end;

// возвращает размер изображения AVI-файла
procedure GetFrameSize(f: string; var w,h: integer);
var
    fst: TFileStream;

     // структура заголовка AVI-файла
    header: record
        RIFF: array[1..4] of uchar;   // 'RIFF'
        nu1:  array[1..5] of LongInt; // не используется (в данном случае)
        AVIH: array[1..4] of uchar;   // 'avih'
        nu2:  array[1..9] of LongInt; // не используется (в данном случае)
        // размер кадра
        Width:  LongInt;
        Height: LongInt;
    end;

begin
    fst := TFileStream.Create(f,fmOpenRead);
```

```
    fst.Read(header,sizeof(header));
    w := header.Width;
    h := header.Height;
    fst.Destroy;
end;


// щелчок на кнопке Eject — выбор файла
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
    top,left: integer;     // левый верхний угол "экрана"
    width,height: integer; // размер экрана
    mw,mh: integer;        // максимально возможный размер экрана
    kh,kw: real;           // коэф-ты масштабирования по h и w
    k: real;               // коэф-т масштабирования

    ms: longint;      // время воспроизведения (миллисекунд)
    min,sec: integer; // время воспроизведения (минут, секунд)
begin
    OpenDialog.Title := 'Выбор клипа';
    if not OpenDialog.Execute
      then exit;


    if MediaPlayer.FileName = OpenDialog.FileName
       then
         // при попытке открыть файл, который уже открыт, возникает ошибка
         exit;

    // Пользователь выбрал файл.
    // Определим размер и положение "экрана" (области на
    // поверхности формы), на котором будет выведен клип
    GetFrameSize(OpenDialog.FileName,width,height);

    mh:=SpeedButton1.Top - 10;
    mw:=Form1.ClientWidth;

    if mh > height
        then kh :=1
        else kh := mh/height;
```

```
    if mw > width
        then kw :=1
        else kw := mw/width;

    if kw < kh
        then k := kw
        else k := kh;

    // здесь масштаб определен

    // определим размер экрана и его положение
    width  := Round(width * k);
    height := Round(height * k);
    left := (Form1.ClientWidth - width) div 2;
    top := 10;

    // открыть файл
    MediaPlayer.FileName := OpenDialog.FileName;
    MediaPlayer.Open;

    MediaPlayer.DisplayRect := Rect(left,top,width,height);
    SpeedButton1.Enabled := True; // кнопка Play

    // вывести информацию о времени воспроизведения
    MediaPlayer.TimeFormat := tfMilliseconds;
    ms := MediaPlayer.Length;
    min := ms div 1000 div 60;
    sec := ms div 1000 mod 60;

    Label1.Caption := Format('Длительность: %d:%.2d',[min,sec]);
    Label1.Font.Color := clWindowText;
    Label2.Font.Color := clWindowText;
    Label3.Font.Color := clWindowText;

    // активизировать процесс воспроизведения
    SpeedButton1.Down := True;  // "нажать" Play
    MediaPlayer.Play;

    Timer1.Enabled := True;
end;
```

```
// щелчок на кнопке Play/Stop
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    if SpeedButton1.Down  then
    begin
      // начать воспроизведение
      MediaPlayer.Play;
      SpeedButton1.Hint := 'Stop';
      SpeedButton2.Enabled := False;
    end
    else begin
      // остановить воспроизведение
      MediaPlayer.Stop;
      SpeedButton1.Hint := 'Play';
      SpeedButton2.Enabled := True;
      Timer1.Enabled := False;
    end;
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  ms: longint;
  min,sec: integer; // время воспроизведения
begin
  ms := MediaPlayer.Position;
  min := ms div 1000 div 60;
  sec := ms div 1000 mod 60;
  Label2.Caption := Format('%d:%.2d',[min,sec]);
end;

// сигнал от плеера
procedure TForm1.MediaPlayerNotify(Sender: TObject);
begin
  if (MediaPlayer.Mode = mpStopped) and SpeedButton1.Down then
  begin
     SpeedButton1.Down := False;   // "отжать" кнопку Play
     SpeedButton2.Enabled := True; // кнопка Eject
     Timer1.Enabled := False;
  end;
end;
```

В качестве экрана, на котором осуществляется воспроизведение видеороликов, используется поверхность формы. Поэтому установить значение свойства `Display` компонента `MediaPlayer1` во время разработки формы нельзя. Кроме того, размер экрана должен быть равен или пропорционален размеру кадров ролика. Значение свойства `Display` устанавливает функция обработки события `Create` для формы, а размер и положение экрана на форме — функция обработки события `Click` на кнопке **Eject** (`SpeedButton1`). Размер экрана устанавливается максимально возможным и таким, чтобы ролик воспроизводился без искажения (высота и ширина экрана пропорциональны высоте и ширине кадров). Размер кадров ролика возвращает функция `GetFrameSize`, которая извлекает нужную информацию из заголовка файла.

Компонент *Animate*

Компонент `Animate`, его значок (рис. 5.10) находится на вкладке **Win32**, позволяет воспроизвести простую, *не сопровождаемую звуком* AVI-анимацию.

Свойства компонента `Animate` перечислены в табл. 5.8.

Рис. 5.10. Значок компонента `Animate`

Таблица 5.8. *Свойства компонента* `Animate`

Свойство	Описание
`FileName`	Имя AVI-файла, в котором находится анимация
`FrameWidth`	Ширина кадра
`FrameHeight`	Высота кадра
`FrameCount`	Количество кадров анимации
`AutoSize`	Признак автоматического изменения размера компонента в соответствии с размером кадров анимации
`Center`	Признак центрирования кадров анимации в поле компонента. Если значение свойства равно `True` и размер компонента больше размера кадров (`AutoSize=False`), кадры анимации располагаются в центре поля компонента
`StartFrame`	Номер кадра, с которого начинается отображение анимации
`StopFrame`	Номер кадра, на котором заканчивается отображение анимации
`Active`	Признак активности процесса отображения анимации
`Color`	Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация

Таблица 5.8 *(окончание)*

Свойство	Описание
`Transparent`	Режим использования "прозрачного" цвета при отображении анимации
`Repetitions`	Количество повторов отображения анимации. Если значение свойства равно нулю, анимация воспроизводится непрерывно
`CommonAVI`	Определяет стандартную анимацию, которая отображается в поле компонента (`aviCopy` — копирование файла; `aviDeleteFile` — удаление файла; `aviRecicleFile` — перемещение файла в корзину)

Нужно еще раз обратить внимание, что компонент `Animate` предназначен для воспроизведения AVI-файлов, которые содержат *только* анимацию. При попытке открыть файл, в котором находится сопровождаемая звуком анимация, возникает исключение.

Использование компонента `Animate` для отображения анимации демонстрирует следующая программа (ее окно приведено на рис. 5.11, а текст — в листинге 5.6). В момент появления окна на экране в поле компонента `Animate` отображается последний кадр анимации — изображение Дельфийского храма. Щелчок на кнопке **Show** активизирует процесс отображения анимации. Непосредственно воспроизведение анимации инициирует метод `Play`, параметры которого задают начальный и конечный кадры фрагмента анимации, который надо воспроизвести, и число повторов отображения анимации. Начальный и конечный кадры, а также количество повторов можно задать, присвоив значения свойствам `StartFrame`, `StopFrame` и `Repetitions` соответственно. В этом случае, чтобы активизировать процесс воспроизведения анимации, необходимо свойству `Activate` присвоить значение `True`.

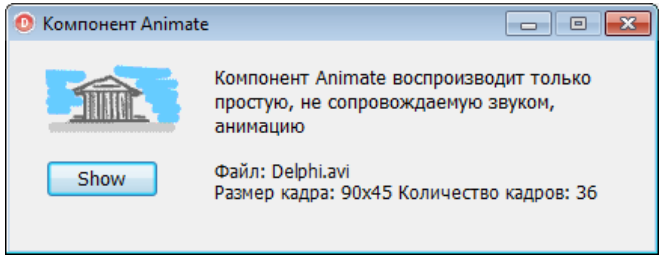


Рис. 5.11. Отображение анимации в поле компонента `Animate`

Листинг 5.6. Отображение AVI-анимации

```
// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
```

```
const
  fn = 'Delphi_2.avi';
var
  st: string;
begin
  try
      Animate1.FileName := fn;
      st := 'Файл: ' + fn + #13+
            Format('Размер кадра: %dx%d Количество кадров: %d',
                   [Animate1.Width, Animate1.Height, Animate1.FrameCount]);
      Label2.Caption := st;
    except on e: Exception do
    begin
      Label1.Caption := 'Ошибка загрузки файла ' +
                        fn + #13 +
                        'Файл недоступен или содержит анимацию, ' +
                        'которая сопровождается звуком.';
      Button1.Enabled := False;
    end;
  end;
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
    // отобразить последний кадр анимации
    Animate1.StartFrame := Animate1.FrameCount;
end;

// щелчок на кнопке Show
procedure TForm1.Button1Click(Sender: TObject);
begin
  // активизировать воспроизведение анимации
  // с первого по последний кадр
  Animate1.Play(1, Animate1.FrameCount,1);
end;
```

Глава 6

Базы данных

Delphi предоставляет программисту набор компонентов, используя которые он может создать программу работы практически с любой базой данных: от Microsoft Access до Microsoft SQL Server и Oracle.

База данных и СУБД

База данных — это файл или совокупность файлов определенной структуры, в которых находится информация. Программная система, обеспечивающая работу с базой данных, называется *системой управления базой данных (СУБД)*. СУБД позволяет создать базу данных, наполнить ее информацией, решить задачи просмотра (отображения), поиска, архивирования и др. Типичным примером СУБД является Microsoft Access.

Локальные и удаленные базы данных

В зависимости от расположения данных и приложения, обеспечивающего работу (доступ) с ними, различают *локальные* и *удаленные* базы данных.

В локальной базе данных файлы данных, как правило, находятся на диске того компьютера, на котором работает программа манипулирования данными. Локальные базы данных не обеспечивают одновременный доступ к информации нескольким пользователям. Несомненным достоинством локальной базы данных является высокая скорость доступа к информации. Microsoft Access — это типичная локальная база данных.

В удаленных базах данных файлы данных размещают на отдельном, доступном по сети, компьютере (сервере). Программы, обеспечивающие работу с удаленными базами данных, строят по технологии "клиент-сервер". Программа-клиент, работающая на компьютере пользователя, обеспечивает доступ к данным (прием команд от пользователя, передачу их серверу, получение и отображение данных). Серверная часть (сервер), работающая на удаленном компьютере, принимает запросы (команды) от клиента, выполняет их и пересылает данные клиенту. Программа, работающая на удаленном компьютере, проектируется так, чтобы обеспе-

чить одновременный доступ к базе данных многим пользователям. В большинстве случаев в качестве серверной части используется стандартный сервер баз данных, например Borland InterBase, Microsoft SQL Server, MySQL, Oracle и т. п. Таким образом, разработка программы работы с удаленной базой данных в большинстве случаев сводится к разработке программы-клиента.

Структура базы данных

База данных (в широком смысле) — это набор однородной, как правило, упорядоченной по некоторому критерию информации.

На практике наиболее широко используются так называемые *реляционные* базы данных (от англ. *relation* — отношение, таблица). Реляционная база данных — это совокупность связанных таблиц данных. Так, например, базу данных Projects (Проекты) можно представить как совокупность таблиц `Projects` (Проекты), `Tasks` (Задачи) и `Resources` (Ресурсы), а базу данных Contacts (Контакты) — одной-единственной таблицей `Contacts` (Контакты). Доступ к таблице осуществляется по имени.

Строки таблиц данных называются *записями*. Они содержат информацию об объектах базы данных. Например, строка таблицы `Tasks` (Задачи) базы данных Projects (Проекты) может содержать название задачи, дату, когда должна быть начата работа, и идентификатор ресурса, который назначен на выполнение задачи. Доступ к записям осуществляется по номеру.

Записи состоят из полей (поле — ячейка в строке таблицы). Поля содержат информацию о характеристиках объекта. Доступ к полю осуществляется по имени. Например, поля записей таблицы `Tasks` могут содержать: идентификатор задачи (поле `TaskID`), название задачи (поле `Title`), идентификатор проекта, частью которого является задача (поле `ProjID`), дату, когда работа по выполнению задачи должна быть начата (поле `Start`), информацию о состоянии задачи (поле `Status`) и идентификатор ресурса, который назначен на выполнение задачи (поле `ResID`). При представлении данных в табличной форме имена полей указывают в заголовке (в первой строке) таблицы.

Физически база данных представляет собой файл или совокупность файлов, в которых находятся таблицы. Например, в Microsoft Access все таблицы, образующие базу данных, хранятся в одном файле с расширением mdb.

Механизмы доступа к данным

Существует довольно много механизмов (технологий) доступа к данным (BDE, ADO, dbExpress и др.).

- Технология BDE, основой которой является процессор баз данных Borland Database Engine (BDE), представляющий собой набор библиотек, драйверов и утилит, обеспечивающих работу практически с любой из существующих баз

данных. Однако существенным ее недостатком является трудоемкость процесса развертывания приложений, созданных на ее основе: помимо самого приложения на компьютер пользователя необходимо установить BDE.

- Технология ADO (ActiveX Data Object) разработана Microsoft как универсальный механизм доступа к базам данных. Ее несомненное достоинство — гибкость, возможность доступа к различным источникам данных.
- Технология dbExpress разработана Borland как эффективный механизм доступа к удаленным базам данных.

Компоненты доступа к данным

Компоненты, обеспечивающие работу с базами данных, находятся на вкладках **dbGo**, **dbExpress**, **InterBase** и **BDE**.

Компоненты вкладки **dbGo** для доступа к данным используют технологию ADO. Компоненты **dbExpress** обеспечивают так называемый однонаправленный (unidirectional) доступ удаленным базам данных на основе разработанной Borland технологии dbExpress. Вкладка **InterBase** содержит компоненты работы с базами данных InterBase. Компоненты вкладки **BDE** для доступа к данным используют процессор баз данных Borland Database Engine.

Следует обратить внимание на то, что компоненты доступа к данным напрямую с базами данных не взаимодействуют, доступ к базе данных (серверу) обеспечивают соответствующие драйверы. На компьютер разработчика драйверы доступа к базам данных устанавливаются в процессе установки Delphi.

На вкладках **DataControls** и **DataAccess** находятся компоненты, обеспечивающие хранение данных во время работы программы (`ClientDataSet`) и их отображение (`DBGrid`, `DBText`, `DBEdit`, `DBMemo` и др.).

Создание базы данных

Программы работы с базами данных обычно работают с существующими файлами данных и, как правило, не предоставляют пользователю возможность создать базу данных. Поэтому, перед тем как приступить к разработке программы работы с базой данных, необходимо как минимум создать базу данных с помощью соответствующей СУБД.

База данных Microsoft Access

Процесс разработки программы работы с базой данных Microsoft Access рассмотрим на примере. Создадим программу, обеспечивающую работу с базой данных "Контакты". Для доступа к базе данных будем использовать технологию ADO.

Перед тем как приступить непосредственно к работе над программой, необходимо с помощью Microsoft Access создать базу данных "Контакты" (файл contacts.mdb), состоящую из одной-единственной таблицы `contacts` (табл. 6.1). Файл базы данных следует поместить, например, в папку D:\Database. Также в папке Database надо создать папку Images (в ней будем хранить иллюстрации).

Таблица 6.1. *Таблица* `contacts` *базы данных "Контакты"*

Поле	Тип	Размер	Описание
`cid`	Автоувеличение	—	Уникальный идентификатор
`name`	Текстовый	50	Имя
`phone`	Текстовый	30	Телефон
`email`	Текстовый	30	Адрес эл. почты
`img`	Текстовый	30	Файл иллюстрации, например фотографии

Здесь надо обратить внимание на следующее. Microsoft Access — это СУБД, *приложение*, обеспечивающее работу с базами данных. Для доступа к данным Microsoft Access использует так называемое ядро баз данных Microsoft Jet.

Доступ к данным

Доступ к данным (источнику данных) при использовании технологии ADO обеспечивают компоненты `ADOConnection`, `ADODataSet`, `ADOTable` и `ADOQuery`, значки которых находятся на вкладке **dbGo** (рис. 6.1).

Рис. 6.1. Компоненты вкладки **dbGo** обеспечивают доступ к данным

Компонент `ADOConnection` обеспечивает соединение с базой данных (источником данных). Компонент `ADODataSet` представляет собой данные, полученные от источника данных, в результате выполнения SQL-запроса. Компонент `ADOTable` также представляет собой данные, полученные из базы данных, но в отличие от компонента `ADODataSet`, который может быть заполнен информацией из разных таблиц, компонент `ADOTable` представляет данные, полученные из одной таблицы. Компонент `ADOQuery` представляет собой данные, полученные из базы данных в результате выполнения SQL-команды.

Для связи между данными, в качестве которых может выступать компонент `ADODataSet`, `ADOTable` или `ADOQuery`, и компонентом, обеспечивающим отображение данных, например `DBGrid`, используется компонент `DataSource`.

Механизм взаимодействия компонентов, обеспечивающих доступ к данным и их отображение, показан на рис. 6.2.

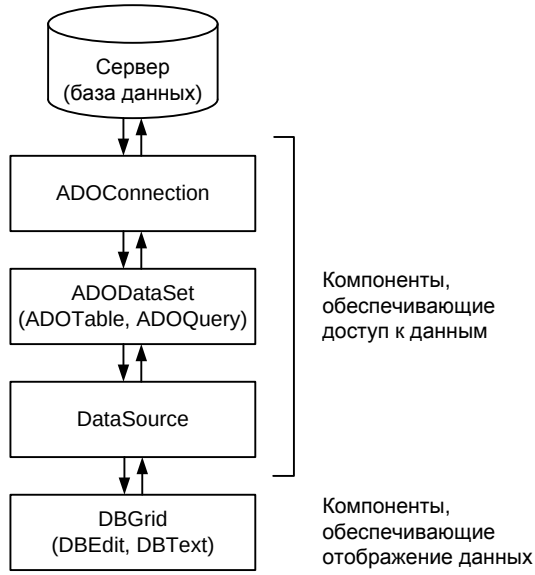


Рис. 6.2. Взаимодействие компонентов, обеспечивающих доступ к данным и их отображение

Форма программы работы с базой данных "Контакты" приведена на рис. 6.3.

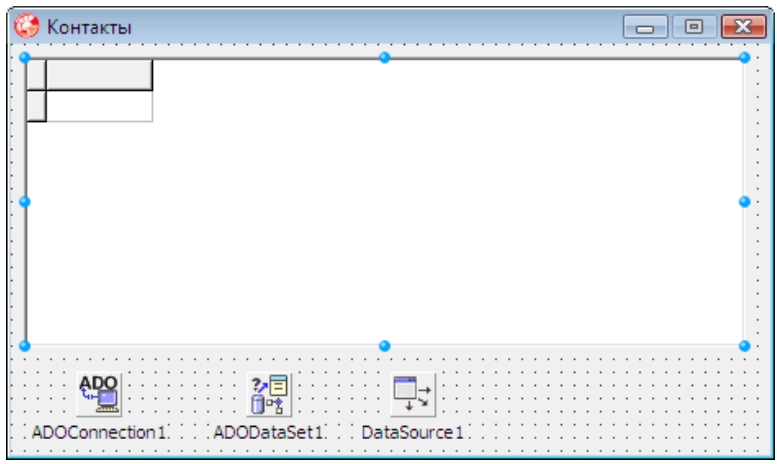


Рис. 6.3. Форма программы работы с базой данных "Контакты"

Сначала на форму надо поместить компонент `ADOConnection`, затем — `ADODataSet`, `DataSource` и `DBGrid`. Компоненты рекомендуется добавлять в том порядке, в котором они перечислены, и сразу настраивать *(см. далее)*. Следует отметить, что компоненты `ADOConnection`, `ADODataSet`, `DataSource` во время работы программы на форме не отображаются (такие компоненты называют невизуальными, или невидимыми), поэтому их можно поместить в любое место формы.

Компонент `ADOConnection`, его свойства приведены в табл. 6.2, обеспечивает *соединение* с базой данных.

***Таблица 6.2.** Свойства компонента `ADOConnection`*

Свойство	Описание
`ConnectionString`	Строка соединения. Содержит информацию, необходимую для подключения к базе данных
`LoginPrompt`	Признак необходимости в момент подключения к базе данных запросить у пользователя имя и пароль. Если значение свойства равно `False`, то окно **Login** в момент подключения к базе данных не отображается
`Mode`	Режим соединения. Соединение с базой данных может быть открыто для чтения (`cmRead`), записи (`cmWrite`), чтения/записи (`cmReadWrite`)
`Connected`	Признак того, что соединение установлено

Настраивается компонент `ADOConnection` следующим образом. Сначала надо сделать щелчок на кнопке с тремя точками, которая находится в строке свойства `ConnectionString`, затем в появившемся окне нажать кнопку **Build**. В результате откроется окно **Свойства связи с данными**, на вкладке **Поставщик данных** которого нужно выбрать тип источника данных (для базы данных Microsoft Access — это **Microsoft Jet OLE DB Provider**) и щелкнуть на кнопке **Далее** (рис. 6.4). Затем на вкладке **Подключение** (рис. 6.5) надо задать базу данных — щелкнуть на кнопке просмотра (**...**) и в открывшемся окне выбрать файл базы данных. Если для доступа к базе данных необходимы пароль и идентификатор пользователя, то их надо указать (по умолчанию к базе данных, созданной в Microsoft Access, доступ есть у пользователя Admin, но пароль для доступа не нужен). После этого можно сделать щелчок на кнопке **Проверить подключение**, убедиться, что соединение с базой данных настроено правильно, и щелчком на кнопке **OK** закрыть окно **Свойства связи с данными**. После этого, если для доступа к базе данных пароль не нужен, необходимо присвоить значение `False` свойству `LoginPrompt`. Значения свойств компонента `ADOConnection1` приведены в табл. 6.3.

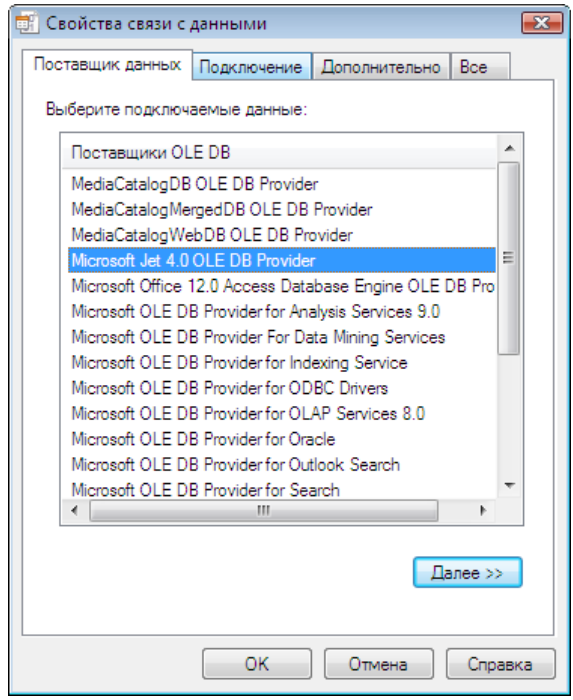


Рис. 6.4. Настройка соединения с базой данных (шаг 1)

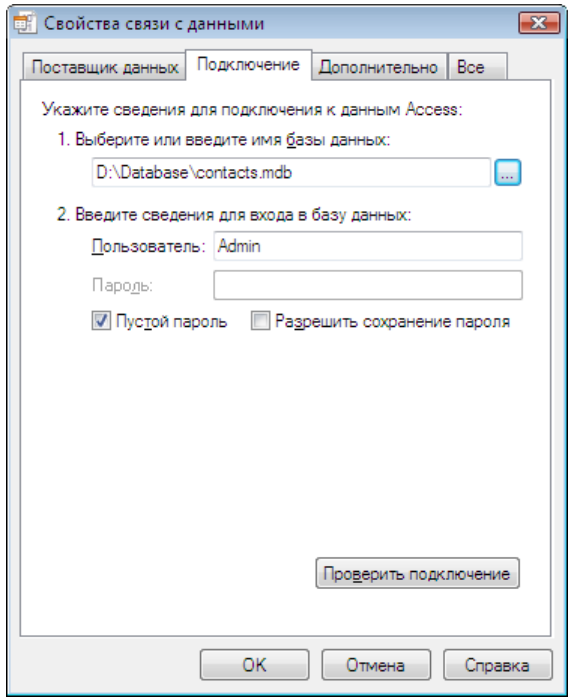


Рис. 6.5. Настройка соединения с базой данных (шаг 2)

Таблица 6.3. *Значения свойств компонента* `ADOConnection1`

Свойство	Значение

`ConnectionString`	`Provider=Microsoft.Jet.OLEDB.4.0;`
	`Data Source=D:\Database\contacts.mdb;`
	`Persist Security Info=False`

`LoginPrompt`	`False`
`Connected`	`False`

После того как будет настроен компонент `ADOConnection`, можно приступить к настройке компонента `ADODataSet`.

Компонент `ADODataSet` (набор данных) хранит данные, полученные из базы данных. Свойства компонента `ADODataSet` приведены в табл. 6.4.

Таблица 6.4. *Свойства компонента* `ADODataSet`

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`CommandText`	Команда, которая направляется серверу
`Parameters`	Параметры команды
`Filter`	Фильтр. Позволяет отобрать записи, удовлетворяющие критерию отбора
`Filtered`	Признак использования фильтра
`Activate`	Открывает или делает недоступным набор данных

В базе данных PhonesBook.mdb информация хранится в таблице `Phones`. Для того чтобы информация из этой таблицы попала в компонент `ADODataSet`, в свойство `CommandText` нужно записать SQL-команду, обеспечивающую выбор необходимой информации. Выбор информации из таблицы базы данных обеспечивает команда `SELECT`. В простейшем случае, когда надо получить всю информацию, которая находится в таблице, в качестве параметров команды `SELECT` нужно указать таблицу, имена полей и, возможно, поле, по содержимому которого данные должны быть упорядочены. Например, SQL-команда, обеспечивающая чтение данных из таблицы `contacts`, выглядит так:

```
SELECT * FROM contacts ORDER BY name
```

Значения свойств компонента `ADODataSet` приведены в табл. 6.5.

Таблица 6.5. *Значения свойств компонента* `ADODataSet`

Свойство	Значение
`Connection`	`ADOConnection1`
`CommandText`	`SELECT * FROM contacts ORDER BY name`
`Activate`	`False`

Завершив настройку компонента `ADODataSet`, можно приступить к настройке компонента `DataSource` — задать значение свойства `DataSet`, определяющего набор данных, связь с которым обеспечивает компонент (табл. 6.6).

Таблица 6.6. *Значения свойств компонента* `DataSource1`

Свойство	Значение
`DataSet`	`ADODataSet1`

Отображение данных

Пользователь может работать с базой данных в режиме таблицы или в режиме формы. В режиме таблицы информация отображается в виде таблицы, что позволяет видеть одновременно несколько записей. Этот режим обычно используется для просмотра информации. В режиме формы отображается одна запись. Обычно данный режим применяется для ввода и редактирования информации. Часто эти два режима комбинируют. Краткая информация (содержимое некоторых ключевых полей) выводится в табличной форме, а при необходимости видеть содержимое всех полей выполняется переключение в режим формы.

Отображение данных в форме таблицы обеспечивает компонент `DBGrid` (рис. 6.6). Свойства компонента (табл. 6.7) определяют вид таблицы и действия, которые могут быть выполнены над данными во время работы программы.

Рис. 6.6. Значок компонента `DBGrid`

Таблица 6.7. *Свойства компонента* `DBGrid`

Свойство	Описание
`DataSource`	Ссылка на источник данных (например, `ADODataSet`)
`Columns`	Отображаемая информация (столбцы)
`BorderStyle`	Вид границы вокруг компонента

***Таблица 6.7** (окончание)*

Свойство	Описание
`Options.dgEditing`	Разрешает (`True`) изменение, добавление и удаление данных. Чтобы активизировать режим редактирования записи, надо нажать клавишу <F2>; чтобы добавить запись — <Insert>; чтобы удалить запись — <Ctrl>+<Del> или <Del>, если значение свойства `Options.dgConfirmDelete` равно `False`
`Options.dgConfirmDelete`	Необходимость подтверждения удаления записи. Если значение свойства равно `True`, то, чтобы удалить запись, пользователь должен нажать комбинацию клавиш <Ctrl>+<Del> и подтвердить выполнение операции удаления щелчком на кнопке **OK** в появившемся окне **Confirm**. Если значение свойства равно `False`, то текущая запись будет удалена в результате нажатия клавиши <Del>
`Options.dgTitles`	Разрешает вывод строки заголовка столбцов
`Options.dgIndicator`	Разрешает (`True`) отображение колонки индикатора. Во время работы с базой данных текущая запись помечается в колонке индикатора треугольником, новая запись — звездочкой, редактируемая — специальным значком
`Options.dgColumnResize`	Разрешает (`True`) менять во время работы программы ширину колонок таблицы
`Options.dgColLines`	Разрешает (`True`) выводить линии, разделяющие колонки таблицы
`Options.dgRowLines`	Разрешает (`True`) выводить линии, разделяющие строки таблицы

Свойство `Columns` компонента `DBGrid` представляет собой коллекцию (массив) объектов типа `TColumn`. Свойства объекта `TColumn` (табл. 6.8) определяют информацию, которая отображается в колонке.

***Таблица 6.8.** Свойства объекта `TColumn`*

Свойство	Описание
`FieldName`	Поле, содержимое которого отображается в колонке
`Width`	Ширина колонки в пикселах
`Font`	Шрифт, используемый для отображения текста в ячейках колонки
`Color`	Цвет фона
`Alignment`	Способ выравнивания текста в ячейках колонки. Текст может быть выровнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)

Таблица 6.8 *(окончание)*

Свойство	Описание
`Title.Caption`	Заголовок колонки. По умолчанию в заголовке отображается имя поля
`Title.Alignment`	Способ выравнивания заголовка. Заголовок может быть выровнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Title.Color`	Цвет фона заголовка колонки
`Title.Font`	Шрифт заголовка колонки

Настройка компонента `DBGrid` выполняется следующим образом. Сначала в коллекцию `Columns` надо добавить столько элементов, сколько столбцов данных необходимо отобразить в поле компонента `DBGrid`. Для этого следует раскрыть окно редактора коллекции — щелкнуть на кнопке с тремя точками, которая находится в поле значения свойства `Columns`, или из контекстного меню, которое появляется в результате щелчка правой кнопкой мыши в поле компонента, выбрать команду **Columns Editor**. В окне редактора коллекции (рис. 6.7) надо сделать щелчок на кнопке **Add New**. В результате в коллекцию `Columns` будет добавлен новый элемент — объект `TColumns`. Добавив нужное количество элементов в коллекцию `Columns`, можно приступить к их настройке. В простейшем случае для каждой колонки достаточно установить значение свойств `FieldName` и `Title.Caption`.

В табл. 6.9 приведены значения свойств компонента `DBGrid1`, а на рис. 6.8 — вид формы после его настройки.

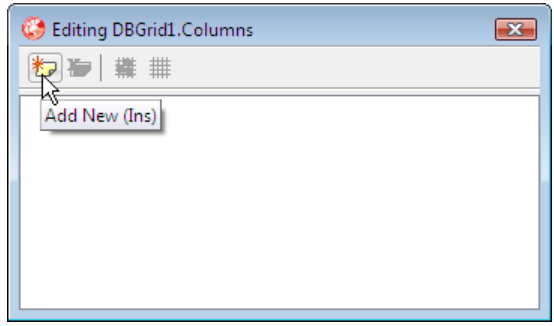


Рис. 6.7. Чтобы добавить элемент в коллекцию `Columns`, надо сделать щелчок на кнопке **Add New**

Таблица 6.9. *Значения свойств компонента* `DBGrid1`

Свойство	Значение
`Font.Name`	`Tahoma`
`Font.Size`	`9`

Таблица 6.9. *Значения свойств компонента* `DBGrid1`

Свойство	Значение
`Columns[0].FieldName`	`Name`
`Columns[0].Width`	`190`
`Columns[0].Title.Caption`	`Имя`
`Columns[0].Title.Font.Style`	`fsBold`
`Columns[1].FieldName`	`phone`
`Columns[1].Width`	`100`
`Columns[1].Title.Caption`	`Телефон`
`Columns[1].Title.Font.Style`	`fsBold`
`Columns[2].FieldName`	`email`
`Columns[2].Title.Caption`	`E-Mail`
`Columns[2].Width`	`100`
`Columns[2].Title.Font.Style`	`fsBold`

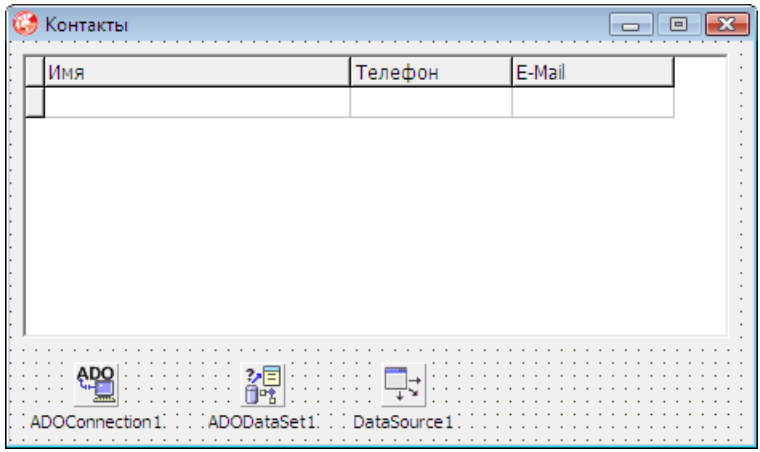


Рис. 6.8. Вид формы после настройки компонента `DBGrid`

Следующее, что надо сделать, — создать процедуры обработки событий `Activate` и `Close` формы (листинг 6.1). Процедура обработки события `Activate` должна открыть базу данных, события `Close` — сохранить изменения, сделанные пользователем. Здесь нужно обратить внимание, что все изменения, сделанные пользователем, автоматически фиксируются в базе данных (в файле) в момент перехода к следующей записи. Однако если пользователь, не завершив ввод или редактирование данных, закроет окно программы, данные последней редактируемой записи не бу-

дут записаны в файл. Поэтому, перед тем как завершить работу программы, надо проверить, не редактирует ли пользователь запись, и если редактирует (в этом случае значение свойства `EditorMode` компонента `DBGrid` равно `True`), то сохранить редактируемую запись в базе данных.

Листинг 6.1. База данных "Контакты"

```
// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  try
    ADOConnection1.Open;
    ADODataSet1.Active := True;
  except
    on e:Exception do begin
        DBGrid1.Enabled := False;
        MessageDlg('Нет файла D:\Database\contacts.mdb',
                   mtError,[mbOk],0);
    end;
  end;
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   if DBGrid1.EditorMode then  // пользователь не завершил редактирование
    begin
      // записать редактируемую запись
      ADODataset1.UpdateBatch(arCurrent);
    end;
end;
```

Выбор информации из базы данных

При работе с базой данных пользователя, как правило, интересует не все ее содержимое, а некоторая конкретная информация. В простейшем случае найти нужные сведения можно, просмотрев таблицу. Однако такой способ поиска неудобен и малоэффективен.

Выбрать нужную информацию из базы данных можно, направив серверу SQL-команду `SELECT` или, если информация уже загружена из базы данных, активизировав фильтр.

SQL-запрос

Чтобы выбрать из базы данных только нужные записи, надо направить серверу SQL-команду `SELECT`, указав в качестве параметра критерий отбора записей.

В общем виде SQL-команда `SELECT`, обеспечивающая выборку данных из базы данных (таблицы), выглядит так:

```
SELECT СписокПолей FROM Таблица WHERE (Критерий) ORDER BY СписокПолей
```

Параметр *`Таблица`* задает таблицу базы данных, из которой надо выбрать (получить) данные. Параметр *`СписокПолей`*, указанный после слова `SELECT`, задает поля, содержимое которых надо получить (если необходимы данные из всех полей, то вместо списка полей можно указать "звездочку"). Параметр *`Критерий`* задает критерий (условие) отбора записей. Параметр *`СписокПолей`*, указанный после `ORDER BY`, задает поля, по содержимому которых будут упорядочены записи таблицы, сформированной в результате выполнения команды.

Например, команда

```
SELECT name, phone FROM contacts WHERE name = 'Культин Н.Б.'
```

обеспечивает выборку из таблицы `contacts` записи, у которой в поле `name` находится текст `Культин Н.Б.`.

В критерии запроса (при сравнении строк) вместо конкретного значения можно указать шаблон. Например, шаблон `Ива%` обозначает все строки, которые начинаются с `Ива`, а шаблон `%Ива%` — все строки, в которых есть подстрока `Ива`. При использовании шаблонов вместо оператора `=` надо использовать оператор `LIKE`. Например, запрос

```
SELECT * FROM contacts WHERE name LIKE 'Ку%'
```

выберет из таблицы `contacts` только те записи, в поле `name` которых находится текст, начинающийся с `Ку`. Вместо оператора `LIKE` можно использовать оператор `CONTAINING` (Содержит). Например, приведенный ранее запрос, целью которого является вывод списка людей, фамилии которых начинаются с `Ку`, при использовании оператора `CONTAINING` будет выглядеть так:

```
SELECT * FROM contacts WHERE name CONTAINING 'Ку'
```

Использование SQL-запроса для поиска информации в базе данных демонстрирует следующая программа (ее форма приведена на рис. 6.9).

Рассматриваемая программа является многооконным приложением (структура проекта, в том числе и список форм, отображается в окне **Project Manager**). В главном окне отображается список абонентов (весь или результат поиска). Окно **Найти** (рис. 6.10), которое становится доступным в результате щелчка в главном окне на соответствующей кнопке, используется для ввода имени абонента, телефон которого нужно найти в базе данных.

Главная форма создается автоматически в момент начала работы над новым проектом. Чтобы создать форму **Найти**, необходимо в меню **File** выбрать команду **New ▸ Form - Delphi**. После того как форма **Запрос** будет настроена (рис. 6.11, табл. 6.10 и 6.11), ее надо сохранить в каталоге проекта — выбрать в меню **File** команду **Save**.

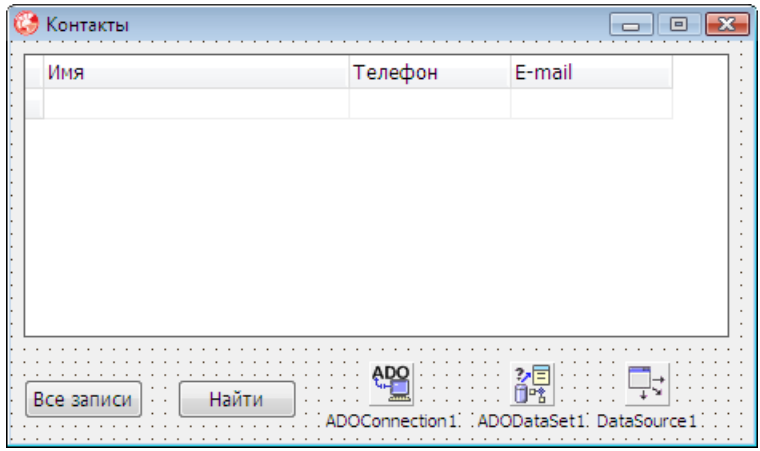


Рис. 6.9. Форма программы работы с базой данных "Контакты"

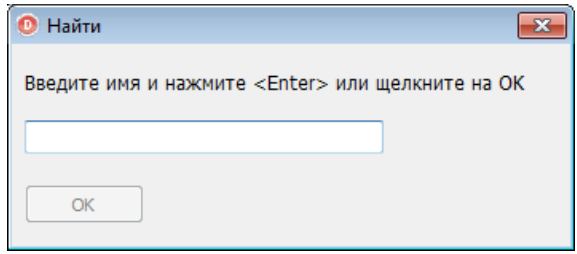


Рис. 6.10. Окно **Найти**

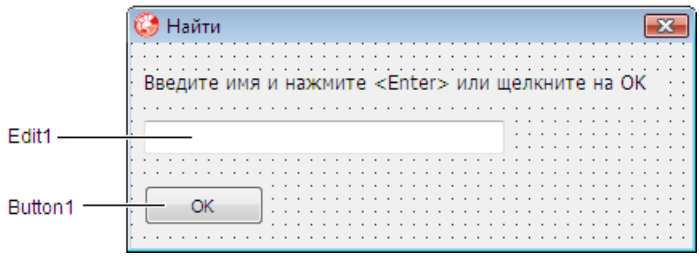


Рис. 6.11. Форма **Найти**

Здесь следует обратить внимание на свойство `ModalResult` кнопки `Button1`. По умолчанию его значение равно `mrNone`. В данном случае значение свойства `ModalResult` равно `mrOk`, поэтому во время работы программы в результате щелчка на кнопке **OK** окно будет закрыто, причем процедура, которая активизирует процесс отображения окна, получит информацию о том, щелчком на какой кнопке пользователь закрыл окно (на кнопке **OK** или на системной кнопке **Закрыть**).

Таблица 6.10. *Значения свойств формы* ***Найти*** *(`FindForm`)*

Свойство	Значение
Name	Form2
BorderStyle	bsSingle
BorderIcons.Minimize	False
BorderIcons.Maximize	False
Position	poMainFormCenter

Таблица 6.11. *Значения свойств кнопки* ***OK*** *(`Button1`) формы* ***Найти***

Свойство	Значение
Enabled	False
ModalResult	mrOk

Завершив настройку главной формы и формы **Найти**, можно приступить к созданию процедур обработки событий. Процедуры обработки событий главной формы приведены в листинге 6.2, формы **Найти** — в листинге 6.3. Следует обратить внимание, что в модуль главной формы надо добавить ссылку на модуль формы **Найти** — директиву `uses FindForm`. Необходимо также обратить внимание, что в директиве `uses` указывается имя модуля формы, которое совпадает с именем файла формы (но без расширения).

Листинг 6.2. Модуль главной формы (MainForm.pas)

```
uses FindForm; // ссылка на модуль формы "Найти"

{$R *.dfm}

// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  try
    ADOConnection1.Open;
    ADODataSet1.Open;
  except
    on e:Exception do begin
        DBGrid1.Enabled := False;
```

```
        MessageDlg('Нет файла D:\Database\contacts.mdb', mtError,[mbOk],0);
    end;
  end;
end;

// щелчок на кнопке "Запрос"
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.ShowModal; // отобразить форму Найти
  if Form2.ModalResult = mrOk then
    // пользователь ввел критерий запроса и нажал кнопку OK
    begin
      ADODataset1.Close;
      ADODataset1.CommandText := 'SELECT * FROM contacts WHERE ' +
                                 'name Like ''%' + Form2.Edit1.Text + '%''';
      ADODataset1.Open;
    end;
end;

// щелчок на кнопке "Все записи"
procedure TForm1.Button2Click(Sender: TObject);
begin
  ADODataset1.Close;
  ADODataset1.CommandText := 'SELECT * FROM contacts ORDER BY name';
  ADODataset1.Open;
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if DBGrid1.EditorMode then  // пользователь не завершил редактирование
    begin
      // записать редактируемую запись
      ADODataset1.UpdateBatch(arCurrent);
    end;
end;
```

Листинг 6.3. Модуль формы *Найти* (FindForn.pas)

```
{ Это окно отображается как модальный диалог.
  Чтобы в результате щелчка на кнопке OK окно закрылось,
```

```
  и вновь стало доступным главное окно, свойству ModalResult
  кнопки Button1 надо присвоить значение mrOK. }

// форма появилась на экране
procedure TForm2.FormActivate(Sender: TObject);
begin
  Edit1.Clear;
  Edit1.SetFocus; // установить курсор в поле Edit1
end;

// изменился текст в поле редактирования
procedure TForm2.Edit1Change(Sender: TObject);
begin
  if Length(Edit1.Text) > 0 then
    Button1.Enabled := True
  else
    Button1.Enabled := False;
end;

// нажатие клавиши в поле редактирования
procedure TForm2.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if (key = #13) and (Length(Edit1.Text) > 0) then
    Button1.SetFocus;
end;
```

Фильтр

Часто нужная информация уже есть в загруженной таблице. В этом случае, для того чтобы ее найти (скрыть ненужную в данный момент информацию), следует воспользоваться механизмом фильтрации записей.

Фильтр — это условие отбора записей. Возможностью фильтрации обладают компоненты `ADODataset`, `ADOQuery` и `ADOTable`. Для того чтобы фильтрация была выполнена, в свойство `Filter` надо записать условие отбора записей и активизировать процесс фильтрации — присвоить значение `True` свойству `Filtered` (чтобы отменить действие фильтра, свойству `Filtered` надо присвоить значение `False`). Следует обратить внимание, что фильтр воздействует на набор данных, сформированный в результате выполнения команды `SELECT`. Принципиальное отличие механизма фильтрации от выборки записей командой `SELECT` состоит в том, что фильтр воздействует на записи, загруженные из базы данных, и скрывает записи, не удовлетворяющие критерию запроса, в то время как команда `SELECT` загружает из базы данных записи, удовлетворяющие критерию запроса.

В качестве примера использования фильтра в листинге 6.4 приведены процедуры обработки событий `Click` для кнопок **Найти** и **Все записи** программы работы с базой данных "Записная книжка".

Листинг 6.4. Щелчок на кнопке *Найти* (использование фильтра)

```
// щелчок на кнопке "Найти"
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.ShowModal; // отобразить форму "Найти"
  if Form2.ModalResult = mrOk then
    // пользователь ввел критерий запроса и нажал кнопку OK
    begin
      // фильтр
      ADODataSet1.Filtered := False;
      ADODataSet1.Filter :='name Like ''%' + Form2.Edit1.Text + '%''';
      ADODataSet1.Filtered := True;
      if ADODataSet1.RecordCount < 0 then
        begin
          // в базе данных нет записей, удовлетворяющих критерию запроса
          ADODataSet1.Filtered := False;
          ShowMessage('В БД нет записей, удовлетворяющих критерию запроса.');
        end;
    end;
end;

// щелчок на кнопке "Все записи"
procedure TForm1.Button2Click(Sender: TObject);
begin
  ADODataSet1.Filtered := False;
end;
```

Работа с базой данных в режиме формы

На практике используются два режима отображения данных: таблица и форма.

В режиме таблицы в окне программы отображается таблица, что позволяет видеть несколько записей одновременно. Обычно этот режим используется для просмотра записей. Отображение данных в режиме таблицы обеспечивает компонент `DBGrid`. Если в таблице, содержимое которой отображается в поле компонента `DBGrid`, много колонок, то пользователь, как правило, не может видеть все столбцы одновременно, и, для того чтобы увидеть нужную информацию, он вынужден ме-

нять ширину столбцов или прокручивать содержимое поля компонента по горизонтали, что не совсем удобно.

В режиме формы в окне программы отображается только одна запись, что позволяет одновременно видеть содержимое *всех* полей записи. Обычно режим формы используется для ввода информации в базу данных, а также для просмотра записей, состоящих из большого количества полей. Часто режим формы и режим таблицы комбинируют.

Компоненты, обеспечивающие просмотр и редактирование полей (рис. 6.12), находятся на вкладке **Data Controls**. На практике чаще всего используются компоненты `DBEdit` и `DBMemo`. Они являются аналогами компонентов `Edit` и `Memo`, ориентированными на работу с базами данных. Свойства компонентов `DBEdit` и `DBMemo`, обеспечивающие работу с базой данных, приведены в табл. 6.12.

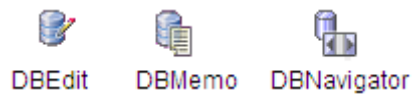


Рис. 6.12. Компоненты `DBEdit` и `DBMemo` обеспечивают редактирование полей записей базы данных, компонент `DBNavigator` — навигацию по БД

Таблица 6.12. *Свойства компонентов* `DBEdit` *и* `DBMemo`

Свойство	**Описание**
`DataSource`	Источник данных
`DataField`	Поле записи БД, содержимое которого отображается в поле компонента

В качестве примера рассмотрим программу, которая обеспечивает работу с базой "Контакты", но уже в режиме формы.

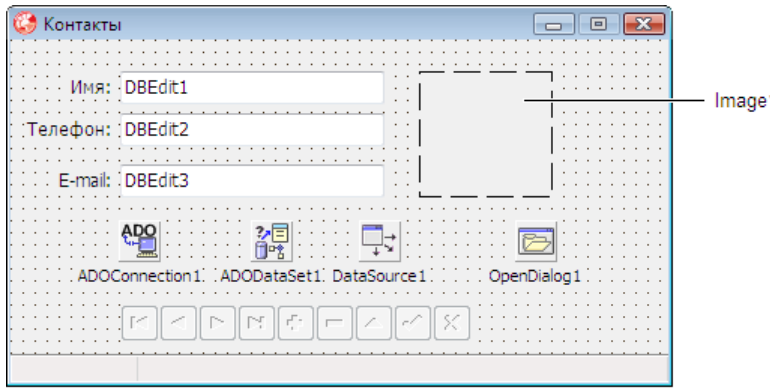


Рис. 6.13. Форма программы работы с базой данных "Контакты" (режим формы)

Форма программы работы с БД "Контакты" приведена на рис. 6.13. Компоненты DBEdit обеспечивают отображение полей *текущей* записи, компонент Image1 — отображение иллюстрации, имя файла которой находится в поле Image. Соединение с базой данных осуществляет компонент ADOConnection1, а доступ к данным, находящимся в таблице contacts, — компонент ADODataSet1. Значения свойств этих компонентов приведены в табл. 6.13.

Таблица 6.13. *Значения свойств компонентов*

Компонент	Свойство	Значение
ADOConnection1	ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\Database\contacts.mdb; Persist Security Info=False
	LoginPrompt	False
ADODataSet1	Connection	ADOConnection1
	CommandText	SELECT * FROM contacts
DataSource1	DataSet	ADOTable1
DBEdit1	DataSource	DataSource1
	DataField	name
	AutoSelect	False
	ReadOnly	True
DBEdit2	DataSource	DataSource1
	DataField	phone
	AutoSelect	False
	ReadOnly	True
DBEdit3	DataSource	DataSource1
	DataField	email
	AutoSelect	False
	ReadOnly	True
Image1	Proportional	True
	Enabled	False

В форме программы отображается одна запись базы данных (эта запись называется текущей). Компонент DBNavigator, его свойства приведены в табл. 6.14, обеспечивает перемещение указателя текущей записи к следующей, предыдущей, первой или последней записи, а также выполнение других операций в результате

щелчка на соответствующей кнопке (табл. 6.15). Следует обратить внимание на свойство `VisibleButtons`. Оно позволяет скрыть некоторые кнопки компонента `DBNavigator` и тем самым запретить выполнение соответствующих операций над файлом данных. Например, присвоив значение `False` свойству `VisibleButtons.nbDelete`, можно скрыть кнопку `nbDelete` и тем самым запретить удаление записей.

Значения свойств компонента `DBNavigator1` приведены в табл. 6.16.

Таблица 6.14. *Свойства компонента* `DBNavigator`

Свойство	Определяет
`DataSource`	Источник данных. В качестве источника данных может выступать, например, компонент `ADODataSet`, `ADOTable` или `ADOQuery`
`VisibleButtons`	Кнопки, которые отображаются в поле компонента. Скрыв некоторые кнопки, можно запретить выполнение соответствующих действий

Таблица 6.15. *Кнопки компонента* `DBNavigator`

Кнопка		Обозначение	Действие
	**К первой**	`nbFirst`	Указатель текущей записи перемещается к первой записи файла данных
	**К предыдущей**	`nbPrior`	Указатель текущей записи перемещается к предыдущей записи файла данных
	**К следующей**	`nbNext`	Указатель текущей записи перемещается к следующей записи файла данных
	**К последней**	`nbLast`	Указатель текущей записи перемещается к последней записи файла данных
	**Добавить**	`nbInsert`	В файл данных добавляется новая запись
	**Удалить**	`nbDelete`	Удаляется текущая запись файла данных
	**Редактирование**	`nbEdit`	Устанавливает режим редактирования текущей записи
	**Сохранить**	`nbPost`	Изменения, внесенные в текущую запись, записываются в файл данных
	**Отменить**	`Cancel`	Отменяет внесенные в текущую запись изменения
	**Обновить**	`nbRefresh`	Записывает внесенные изменения в файл

Таблица 6.16. *Значения свойств компонента* `DBNavigator1`

Свойство	Значение
DataSource	DataSource1
VisibleButtons.bnRefresh	False

Модуль формы программы работы с базой данных приведен в листинге 6.5. Процедура обработки события `AfterScroll` для компонента `ADODataSet1`, которое возникает после того, как указатель текущей записи будет перемещен к другой записи (следующей или предыдущей, в зависимости от того, какую кнопку компонента `DBNavigator` нажал пользователь) инициирует процесс отображения иллюстрации. Отображение иллюстрации обеспечивает процедура `ShowImage`, которой в качестве параметра передается содержимое поля `Image` (или пустая строка, если поле пустое). Процедура `ShowImage` выводит иллюстрацию или, если поле `img` текущей записи пустое, картинку nobody.jpg.

Информация в поля записи `name`, `phone` и `email` вводится обычным образом — путем заполнения полей **Имя**, **Телефон** и **E-mail**. Чтобы ввести информацию в поле img (задать имя файла иллюстрации), надо сделать щелчок в поле компонента `Image1`. В результате открывается диалог **Выбор изображения** (компонент `OpenDialog`), в котором пользователь может выбрать иллюстрацию. Если иллюстрация выбрана, то имя файла иллюстрации записывается в поле `Image` текущей записи БД, а сам файл копируется в каталог Images.

Листинг 6.5. Программа работы с базой данных "Контакты" (режим формы)

```
unit MainForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, DB, ADODB, Grids, DBGrids, ExtCtrls, DBCtrls, StdCtrls,
  ComCtrls, Mask, ExtDlgs;

type
  TForm1 = class(TForm)
    ADOConnection1: TADOConnection;
    ADODataSet1: TADODataSet;
    DataSource1: TDataSource;
    Label1: TLabel;
    Label2: TLabel;
```

```
    DBNavigator1: TDBNavigator;
    StatusBar1: TStatusBar;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    Image1: TImage;
    OpenDialog1: TOpenDialog;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormActivate(Sender: TObject);
    procedure ADODataSet1AfterScroll(DataSet: TDataSet);
    procedure Image1Click(Sender: TObject);
    procedure DBNavigator1Click(Sender: TObject; Button: TNavigateBtn);
  private
    { Private declarations }
    aPath: string;
    procedure ShowImage(img: string);// отображает картинку в поле Image1
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses Jpeg, IniFiles, StrUtils;

// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  aPath :='d:\database\';
  try
    ADOConnection1.Open;
    ADODataSet1.Open;
    StatusBar1.Panels[0].Text := ' Запись: 1';
  except
    on e:Exception do begin
        DBEdit1.Enabled := False;
```

```
        DBEdit2.Enabled := False;
        DBNavigator1.Enabled := False;
        MessageDlg('Ошибка доступа к файлу БД: '+
                   aPath + 'NoteBook.mdb', mtError, [mbOk], 0);
    end;
  end;
end;

// событие возникает после перехода к другой записи
procedure TForm1.ADODataSet1AfterScroll(DataSet: TDataSet);
var
    img: string;
begin
    if ADODataSet1.RecNo <> -1 then
    begin
       StatusBar1.Panels[0].Text := ' Запись: ' + IntToStr(ADODataSet1.RecNo);

       if ADODataSet1.FieldValues['img'] <> Null then
            img := ADODataSet1.FieldValues['img']
       else
            img := '';
       ShowImage(img);
    end
    else
       StatusBar1.Panels[0].Text := ' Новая запись'
end;

// отображает иллюстрацию
procedure TForm1.ShowImage(img: string);
begin
  if img = '' then
    img := 'nobody.jpg';
  try
     Image1.Picture.LoadFromFile(aPath+'images\'+img);
  finally
  end;
end;

// щелчок на кнопке компонента DBNavigator
procedure TForm1.DBNavigator1Click(Sender: TObject; Button: TNavigateBtn);
```

```
begin
  case Button of
    nbInsert, nbDelete, nbEdit:
          begin
             DBEdit1.ReadOnly := False;
             DBEdit2.ReadOnly := False;
             DBEdit3.ReadOnly := False;
             Image1.Enabled := True;
             if Button = nbInsert then
                ShowImage('nobody.jpg');
          end ;
    nbPost, nbCancel:
          begin
             DBEdit1.ReadOnly := True;
             DBEdit2.ReadOnly := True;
             DBEdit3.ReadOnly := True;
             Image1.Enabled := False;
          end ;
  end;
end;

// щелчок в поле компонента Image (выбор картинки)
procedure TForm1.Image1Click(Sender: TObject);
var
  nFileName: string;
begin
  OpenDialog1.FileName := '*.jpg';
  if OpenDialog1.Execute then
  begin
      // пользователь выбрал изображение
      nFileName := ExtractFileName(OpenDialog1.FileName);
      CopyFile(PChar(OpenDialog1.FileName),
               PChar(aPath + 'images\'+ nFileName), false);
      ShowImage(nFileName);
      ADODataSet1.FieldValues['img'] := nFileName;
  end;
end;
```

```
// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   if ADODataSet1.State = dsEdit then  // пользователь не завершил
                                       // редактирование
    begin
      // записать редактируемую запись
      ADODataset1.UpdateBatch(arCurrent);
    end;
end;
```

Загрузка строки соединения из INI-файла

В программе работы с базой данных "Контакты" для соединения с источником данных (БД) используется компонент `ADOConnection`, свойство `ConnectionString` которого явно задает имя файла базы данных и путь к нему. Поэтому при установке программы работы с базой данных на другой компьютер необходимо, чтобы каталог базы данных находился на том же диске, что и на компьютере программиста, что не всегда удобно, а иногда и невыполнимо. Избежать подобных проблем можно, если строку соединения загружать из INI-файла в начале работы программы.

Задачу загрузки строки соединения из файла или ее формирования с учетом реального размещения базы данных можно возложить на процедуру обработки события `BeforeConnect` компонента `ADOConnection`, которое происходит непосредственно перед подключением к базе данных. Приведенная в листинге 6.6 процедура показывает, как это можно сделать. В данном примере из INI-файла (листинг 6.7) загружается не строка соединения, а только имя папки, в которой находится файл базы данных, после чего строка соединения формируется путем замены ее фрагмента.

Листинг 6.6. Обработка события `BeforeConnection`

```
procedure TForm1.ADOConnection1BeforeConnect(Sender: TObject);
var
  p1,p2: integer;
  IniFile: TIniFile;
  fn: string;          // имя INI-файла
  st: string;          // строка соединения
begin
    // загрузить строку соединения из INI-файла
    // INI-файл должен находиться в том же каталоге, что и EXE-файл
```

```
    // программы работы с БД, и его имя совпадает с именем EXE-файла
    p1 := Pos('.exe', Application.ExeName);

    fn := Copy(Application.ExeName, 1, p1-1) + '.ini';
    IniFile := TIniFile.Create(fn);

    // прочитаем из INI-файла
    // имя папки, в которой должен находиться файл БД
    // ключ aPath находится в секции data
    aPath := IniFile.ReadString('data','aPath','');

    if aPath = '' then
      MessageDlg('Нет файла: '+ fn ,mtError,[mbOk],0);

    st := ADOConnection1.ConnectionString;
    p1 := Pos('Data Source',st);
    p2 := PosEx(';',st,p1);

    Delete(st,p1,p2-p1);
    Insert('Data Source='+ aPath+ 'notebook.mdb',st,p1);

    ADOConnection1.ConnectionString := st;
end;
```

Листинг 6.7. INI-файл

```
[data]
aPath=D:\Database\
```

База данных Blackfish SQL

Сервер Blackfish SQL Server представляет собой компактный, высокопроизводительный сервер баз данных. Но он, в отличие, например, от InterBase, Microsoft SQL Server или MySQL, практически не требует администрирования, что делает его использование в информационных системах среднего уровня сложности наилучшим решением.

На компьютер разработчика сервер Blackfish SQL Server устанавливается как *служба* автоматически вместе с другими компонентами среды Delphi. Запускается сервер также автоматически, при каждом включении компьютера.

Чтобы убедиться, что сервер запущен, надо раскрыть окно **Службы** (рис. 6.14) — открыть Панель управления, сделать щелчок на значке **Администрирование** и затем (в появившемся окне **Администрирование**) — на значке **Службы**.

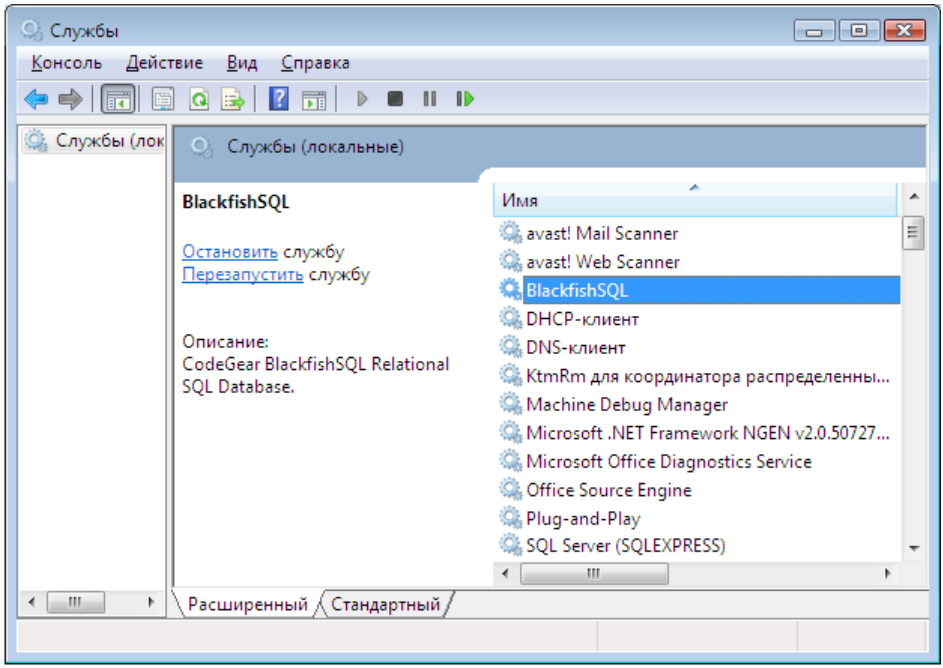


Рис. 6.14. Окно **Службы**

Доступ к серверу

Доступ к серверу Blackfish SQL Server есть только у администратора системы. Только администратор (по умолчанию, после установки сервера, имя администратора — sysdba, пароль — masterkey) может создать базу данных и открыть ее для других пользователей. Здесь следует обратить внимание на то, что в Blackfish SQL Server пользователи баз данных не регистрируются на сервере так, как это делается, например, в InterBase или в Microsoft SQL Server. Вся информация о пользователях базы данных и их полномочиях хранится не в системной базе данных на сервере, а в самой базе данных.

Создание базы данных

Задачу создания базы Blackfish SQL Server рассмотрим на примере. Создадим базу данных "Книги".

Замечание

Перед тем как приступить к разработке программы работы с базой данных Blackfish SQL Server, рекомендуется выполнить настройку сервера — в файле конфигурации BSQLServer.exe.config (он находится в каталоге Program Files\Embarcadero\RAD Studio\

8.0\bin) указать каталог, предназначенный для файлов баз данных. Имя каталога надо указать в качестве значения ключа `DataDirectory`.

Сделать это можно с помощью утилиты Database Explorer. Чтобы получить доступ к утилите, надо в окне **Project Manager** открыть вкладку **Data Explorer**. Следует обратить внимание, что Database Explorer можно запустить и из операционной системы (файл Program Files\Embarcadero\RAD Studio\8.0\bin\DataExplore.exe).

Процесс создания БД Blackfish SQL Server состоит из двух шагов. Сначала надо создать соединение, затем через созданное соединение направить серверу команды, обеспечивающие создание базы данных (`CREATE DATABASE`) и таблиц в ней (`CREATE TABLE`). Следует обратить внимание, что создать базу данных может только администратор сервера.

Чтобы создать соединение, надо в окне **Data Explorer** сделать щелчок правой кнопкой мыши в строке **BLACKFISHSQL**, в появившемся списке выбрать команду **Add New Connection** (рис. 6.15) и в появившемся окне **Add New Connection** ввести имя соединения.

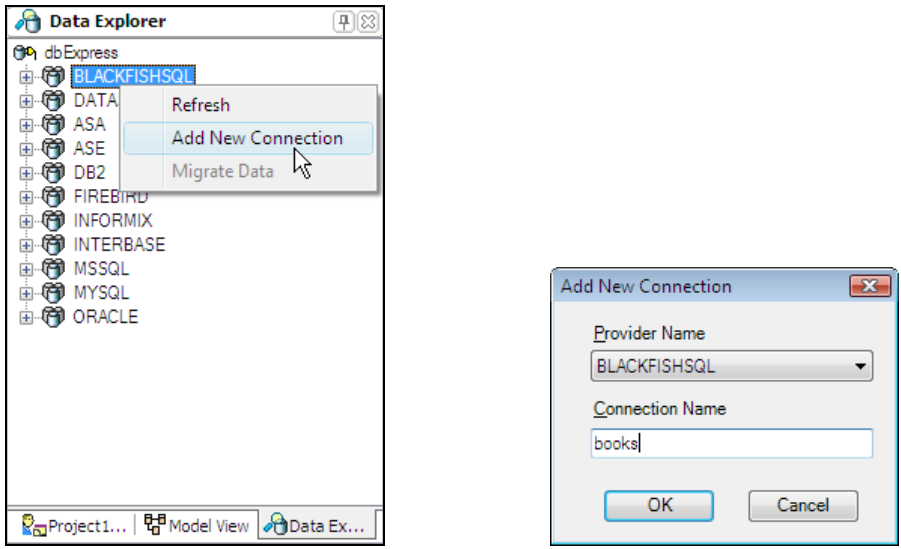


Рис. 6.15. Создание соединения с базой данных Blackfish SQL

Замечание

Информация о всех созданных соединениях находится в файле C:\Documents and Settings\All Users\Документы\RAD Studio\dbExpress\8.0\dbxconnections.ini.

После того как соединение будет создано, следует раскрыть список соединений **BLACKFISHSQL**, сделать щелчок правой кнопкой мыши на имени только что созданного соединения и в появившемся списке выбрать команду **Modify Connection** (рис. 6.16).

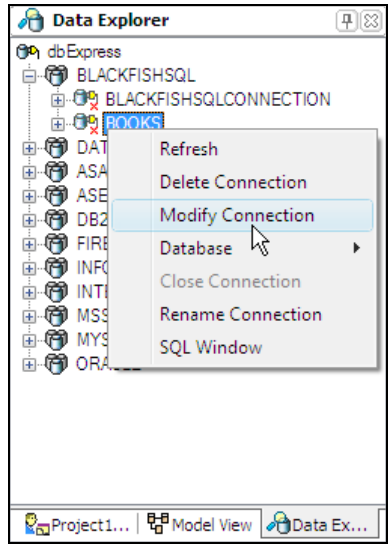


Рис. 6.16. Начало настройки соединения

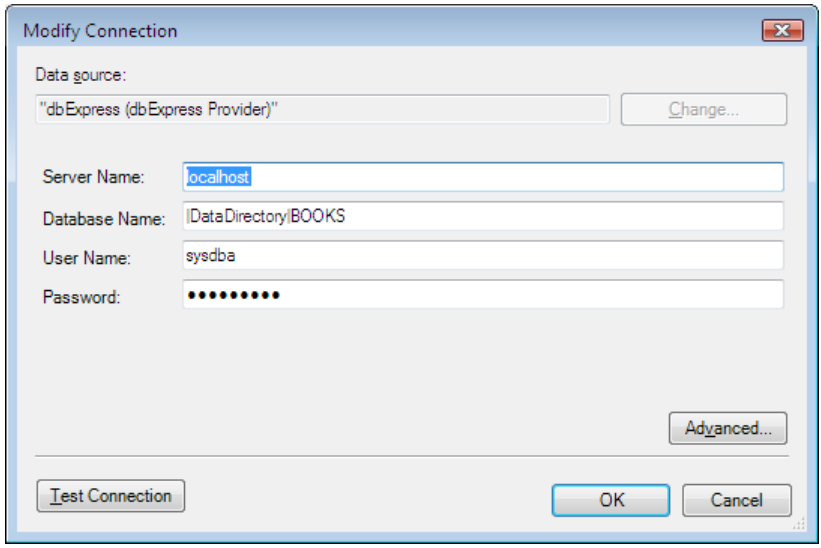


Рис. 6.17. Окно **Modify Connection**

Далее в окне **Modify Connection** (рис. 6.17) в поле **Server Name** следует ввести сетевое имя компьютера, на котором установлен Blackfish SQL Server (вместо имени компьютера можно указать `localhost`), в поле **Database Name** — имя базы данных (если перед именем базы данных указать макрос `|DataDirectory|`, то база данных будет создана в каталоге, имя которого указано в файле bsqlserver.exe.config в качестве параметра ключа `DataDirectory`), и сделать щелчок на кнопке **Advanced**. Затем в открывшемся окне **Advanced Properties** (рис. 6.18)

свойству `create` следует присвоить значение `True`. Далее надо нажать кнопку **OK**. В результате этих действий будет создана база данных Blackfish SQL — файлы books.jds, books_LOGA_0000000000 и books_LOGA_ANCHOR. Убедиться, что база создана, можно, нажав кнопку **Test Connection** во вновь ставшем доступном окне **Modify Connection**.

После того как база данных будет создана, можно приступить к созданию таблиц и наполнению их информацией. Делается это путем направления серверу соответствующих SQL-команд.

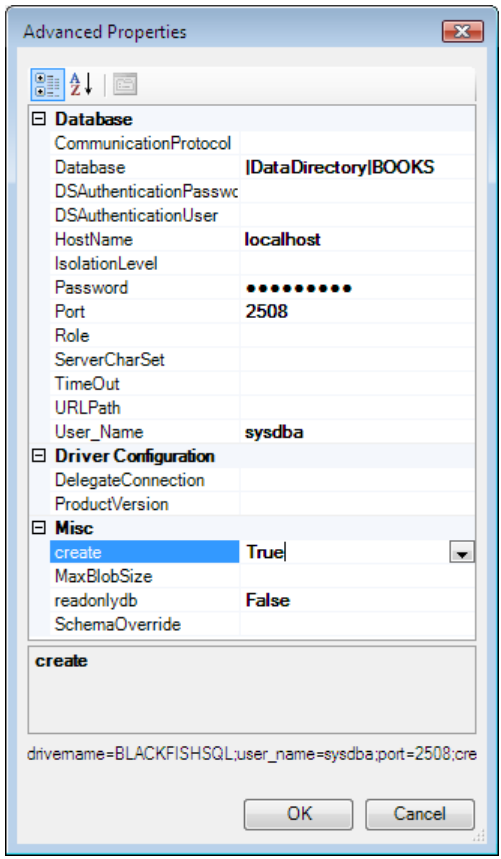


Рис. 6.18. Параметры соединения

Чтобы направить серверу SQL-команду, обеспечивающую, например, создание таблицы в базе данных, надо раскрыть список соединений, сделать щелчок правой кнопкой мыши на имени нужного соединения и в появившемся списке выбрать команду **SQL Window** (рис. 6.19). В результате этого откроется страница **Data Explorer** (рис. 6.20), в нижней части которой можно набирать SQL-команды.

Например, чтобы создать в базе данных таблицу `books`, серверу надо направить команду

```
CREATE TABLE books (title CHAR(60) NOT NULL, author CHAR(30))
```

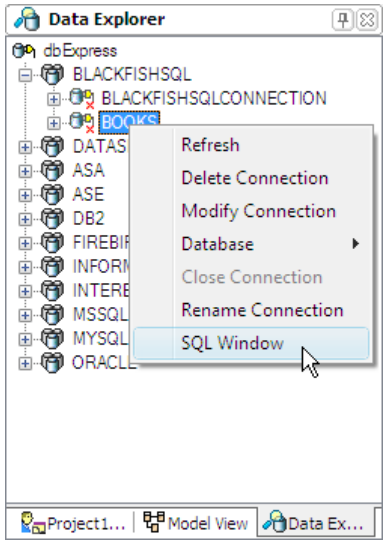


Рис. 6.19. Команда **SQL Window**

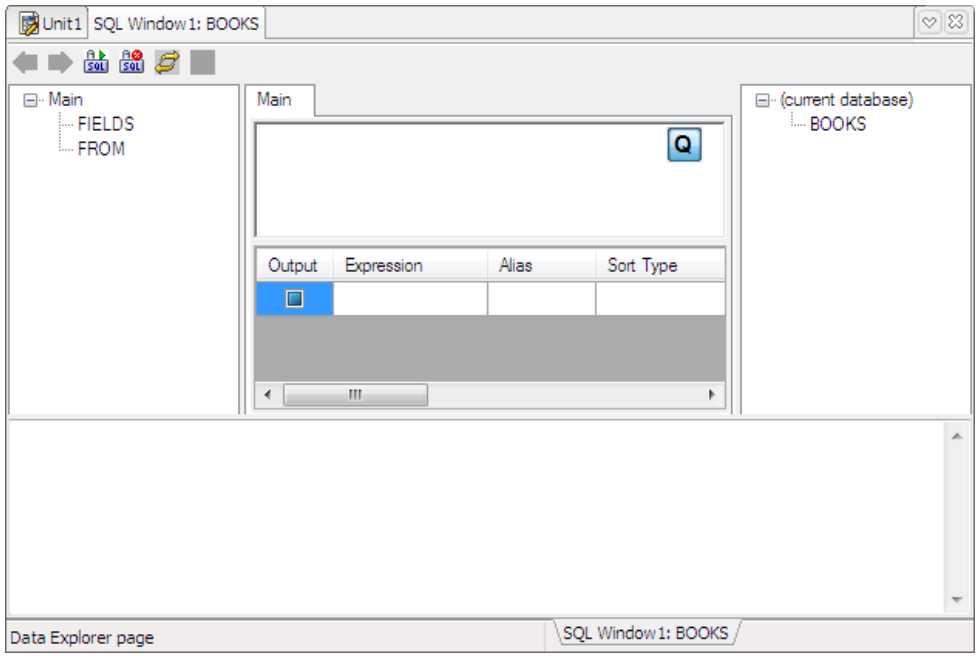


Рис. 6.20. Страница **Data Explorer**

Процесс выполнения команды активизируется щелчком на кнопке **Execute SQL** (рис. 6.21).

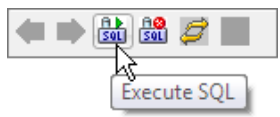


Рис. 6.21. Кнопка **Execute SQL**

Доступ к базе данных

Изначально доступ к базе данных есть только у администратора сервера. Чтобы другой пользователь тоже мог работать с базой данных, администратор (или другой обладающий правами администратора пользователь) должен зарегистрировать этого нового пользователя — создать учетную запись.

Регистрацию (создание учетной записи) пользователя базы данных обеспечивает команда

```
CREATE USER UserID PASSWORD "password"
```

где *UserID* и *password* — идентификатор и пароль пользователя.

Например, команда

```
CREATE guest PASSWORD "guest"
```

создает пользователя (учетную запись пользователя) `guest`, который для доступа к базе данных должен использовать пароль `guest`.

Права пользователей

Пользователи, имеющие доступ к базе данных, могут обладать разными правами. Например, одному пользователю может быть разрешено только просматривать записи, другому — просматривать и изменять, а третьему — просматривать, добавлять и изменять.

Определить права доступа пользователя можно как на уровне доступа к базе данных в целом, так и на уровне доступа к отдельным таблицам.

Определить права пользователя можно, направив серверу команду

```
GRANT privilege TO UserID
```

где *privilege* — список прав (табл. 6.17), *UserID* — идентификатор пользователя.

***Таблица 6.17.** Права пользователя на уровне базы данных*

Идентификатор	Права
`STARTUP`	Может открыть базу данных
`WRITE`	Может записывать информацию в базу данных
`CREATE`	Может создавать новые таблицы
`DROP`	Может удалять таблицы
`RENAME`	Может переименовывать таблицы
`ADMINISTRATOR`	Может открыть доступ к базе данных другим пользователям (создать пользователя), может изменить права существующих пользователей, может выполнять другие действия (`WRITE`, `CREATE`, `DROP`, `RENAME`), а также шифровать базы данных

Например, команда

```
GRANT STARTUP TO platon
```

открывает базу данных пользователю `platon` для просмотра, а команда

```
GRANT STARTUP, WRITE TO danila
```

предоставляет пользователю `danila` право записи в базу данных.

Команду `GRANT` можно использовать для определения прав пользователя для доступа к таблицам базы данных. В этом случае команда выглядит так:

```
GRANT privileges ON table TO UserID
```

где *`privileges`* — список прав (SQL-команд, которые пользователь может направить серверу); *`table`* — таблица базы данных; *`UserID`* — идентификатор пользователя.

Например, команды, определяющие полномочия доступа пользователей `platon` и `danila` к таблице `books`, могут быть такими:

```
GRANT SELECT ON books TO platon
GRANT SELECT, INSER, UPDATE ON books TO danila
```

Как видно, пользователь `platon` может только просматривать таблицу `books`, а пользователь `danila` — просматривать, добавлять и редактировать информацию.

Если нескольким пользователям надо предоставить одинаковые права, то вместо того, чтобы определять права для каждого пользователя, можно определить (создать) *роль*, указать права для этой роли и затем назначить роль пользователям.

Команда, обеспечивающая создание роли, в общем виде выглядит так:

```
CREATE ROLE RoleID
```

где *`RoleID`* — идентификатор роли.

Пример:

```
CREATE ROLE operator
```

Команда определения полномочий роли идентична команде определения полномочий пользователя. Например, команда

```
GRANT SELECT, INSER, UPDATE ON books TO operator
```

определяет полномочия роли `operator`.

После того как роль определена, ее можно назначить конкретному пользователю.

Команда назначения роли пользователю в общем виде выглядит так:

```
GRANT RoleID TO UserID
```

Например, команда

```
GRANT operator TO larisa
```

предоставляет пользователю `larisa` полномочия в соответствии с ролью `operator`.

В каждой базе данных по умолчанию определена роль `PUBLIC`. Эта роль обычно используется для обеспечения доступа к таблицам базы данных в режиме "только просмотр". Особенность этой роли заключается в том, что она автоматически назначается пользователям, полномочия которых явно (с помощью команды `GRANT`) не заданы. Следует обратить внимание на то, что полномочия роли `PUBLIC` по умолчанию не определены, их надо указывать явно.

Например, команда

```
GRANT SELECT ON books TO PUBLIC
```

открывает таблицу `books` для просмотра всем пользователям базы данных, права которых не указаны явно.

База данных "Книги"

Процесс создания приложения работы с базой данных Blackfish SQL рассмотрим на примере — создадим программу, обеспечивающую работу с базой данных "Книги".

Сначала надо создать базу данных (`books`), поместить в нее таблицу `books`, записать в таблицу данные, открыть доступ к базе данных пользователю `guest` и определить его полномочия. Характеристики полей таблицы `books` приведены в табл. 6.18, SQL-команды, обеспечивающие выполнение описанной ранее задачи, — в листинге 6.8.

Таблица 6.18. *Поля таблицы* `books`

Поле	Тип	Примечание
id	INT AUTOINCREMENT	Автоувеличение
title	VARCHAR(60)	Обязательное поле
author	VARCHAR(25)	

Листинг 6.8. Создание таблицы и определение полномочий пользователя `guest`

```
CREATE TABLE books(id INT AUTOINCREMENT, title VARCHAR(60) NOT NULL,
author VARCHAR(30));
INSERT INTO books(title,author) VALUES('Delphi в задачах и примерах', 'Культин
Н.Б.');
INSERT INTO books(title,author) VALUES('Microsoft Visual C# в задачах и
примерах', 'Культин Н.Б.');
INSERT INTO books(title,author) VALUES('Microsoft Visual C++ в задачах и
примерах', 'Культин Н.Б.');
CREATE USER guest PASSWORD 'guest';
GRANT SELECT ON books TO public
```

Доступ к базе данных Blackfish SQL Server обеспечивают компоненты `SQLConnection` и `SQLDataset`, находящиеся на вкладке **dbExpress** (рис. 6.22). Следует обратить внимание, что архитектура DBEXPRESS поддерживает только так называемый однонаправленный (unidirectional) доступ к данным. Это значит, что указатель текущей записи может перемещаться лишь в одном направлении — только к следующей записи. Именно поэтому для просмотра данных в режиме таблицы исполь-

зовать компонент DataGrid напрямую нельзя: при попытке переместить указатель текущей строки (записи) в предыдущую строку возникает исключение. Чтобы обеспечить возможность просмотра данных в режиме таблицы, в форму приложения необходимо добавить компонент ClientDataSet (он находится на вкладке **Data Access**), который обеспечит буферизацию данных и возможность перемещения указателя текущей записи "назад" (к предыдущей записи).

Рис. 6.22. Компоненты **dbExpress**

Форма программы работы с базой данных "Книги" приведена на рис. 6.23. Компонент SQLConnection обеспечивает соединение с базой данных, компонент SQLDataSet хранит информацию, полученную в результате выполнения запроса (команды SELECT). Компонент ClientDataSet используется в качестве буфера, он хранит данные (копию), полученные из базы, и обеспечивает возможность просмотра данных с помощью компонента DataGrid. Связь между компонентами SQLDataSet и ClientDataSet осуществляет компонент DataSetProvider, между компонентами ClientDataSet и DBGrid — компонент DataSource.

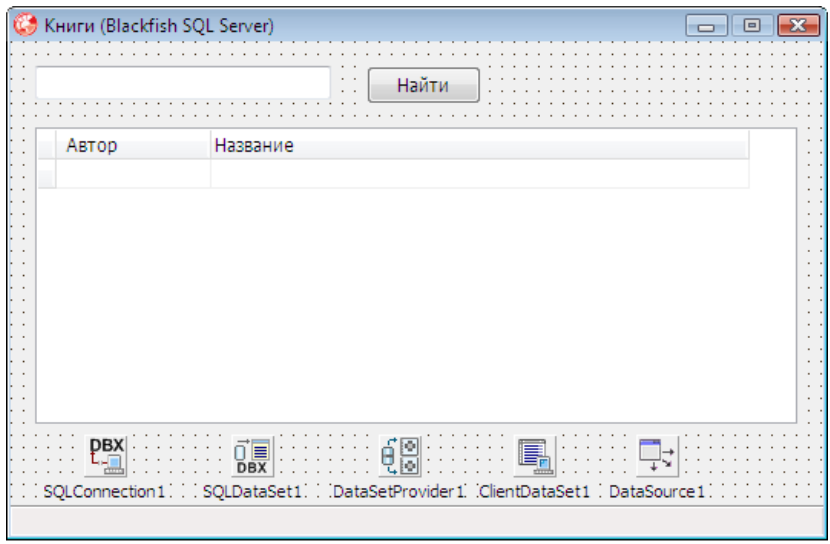


Рис. 6.23. Форма программы работы с базой данных "Книги"

Значения свойств компонентов приведены в табл. 6.19. Необходимо обратить внимание, что значения следует устанавливать в том порядке, в котором они приведены в таблице.

Таблица 6.19. *Значения свойств компонентов*

Компонент	Свойство	Значение
SQLConnection1	Name	SQLConnection1
	ConnectionName	books
	DriverName	BLACKFISHSQL
	LoginPrompt	False
	LoadParamsOnConnect	False
SQLDataSet1	Name	SQLDataSet1
	SQLConnection	SQLConnection1
	CommandText	SELECT * FROM books
DataSetProvider1	Name	DataSetProvider1
	DataSet	SQLDataSet1
ClientDataSet11	Name	ClientDataSet11
	ProviderName	DataSetProvider1
DataSource1	Name	DataSource1
	DataSet	ClientDataSet1
DataGrid1	Name	DataGrid1
	DataSource	DataSource1
	ReadOnly	True
	Columns[0].FieldName	author
	Columns[0].Title.Caption	Автор
	Columns[0].Title.Width	100
	Columns[0].Title.Font.Style	fsBold
	Columns[1].FieldName	title
	Columns[1].Title.Caption	Название
	Columns[1].Width	350
	Columns[0].Title.Font.Style	fsBold

Процедуры обработки событий приведены в листинге 6.9.

Листинг 6.9. Процедуры обработки событий

```
uses DBXCOmmon;

procedure TForm1.FormActivate(Sender: TObject);
begin

  try
    SQLConnection1.Open;
    ClientDataset1.Open;
    StatusBar1.SimpleText := 'Записей: ' + IntToStr(SQLDataset1.RecordCount);
  except
        on e:TDBXError do ShowMessage(e.Message);
   end;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  cmd: string;
begin
  cmd := 'SELECT * FROM books WHERE UPPER(title) LIKE UPPER(''%'+
         Edit1.Text + '%'')';

  { При попытке изменить значение свойства CommandText открытого набора
    данных возникает исключение. Поэтому перед тем как изменить значение
    свойства CommandText, закроем набор данных. }

  SQLConnection1.Close;
  ClientDataset1.close;

  SQLDataset1.CommandText := cmd;

  SQLConnection1.Open;
  ClientDataset1.Open;

  if SQLDataset1.RecordCount <> 0
    then StatusBar1.SimpleText :=
           'Записей: ' + IntToStr(SQLDataset1.RecordCount)
```

```
    else StatusBar1.SimpleText :=
           'В БД нет записей, удовлетворяющих критерию запроса';
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key = Chr(VK_RETURN) then
     Button1.SetFocus;
end;
```

Как было сказано, информация о соединениях, созданных программистом, находится в файле dbxconnections.ini, из которого она при настройке компонента `SQLConnection` копируется в свойство `Params`. Если вы предполагаете, что программа работы с базой данных будет устанавливаться на другие компьютеры, то свойству `LoadParamsOnConnect` компонента `SQLConnection` следует присвоить значение `True`. В этом случае параметры соединения будут загружаться из файла dbxconnections.ini при каждом открытии соединения. Такое решение позволяет выполнять настройку программы работы с базой данных путем внесения изменений в файл конфигурации dbxconnections.ini.

Параметры соединения отображаются в окне **Value List editor** (рис. 6.24), которое открывается в результате щелчка на находящейся в поле значения свойства `Params` кнопке с тремя точками.

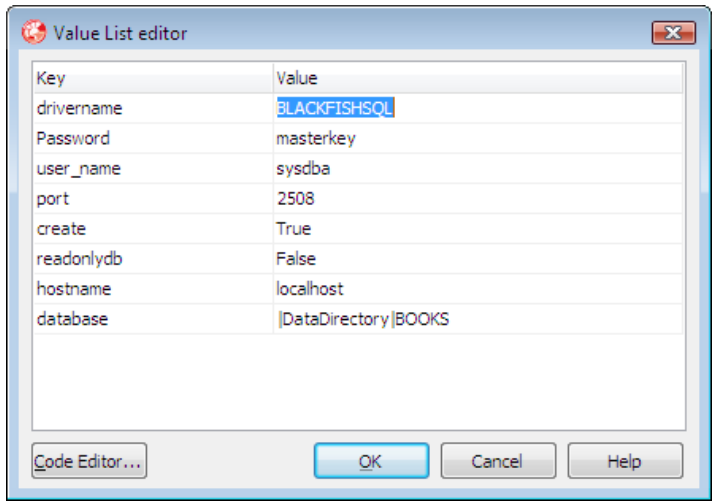


Рис. 6.24. Параметры соединения с БД Blackfish SQL Server

Развертывание приложения работы с базой данных Blackfish SQL Server

Установка и настройка сервера

Сервер Blackfish SQL Server является .NET-приложением. Поэтому, перед тем как приступить к непосредственной установке Blackfish SQL Server, необходимо убедиться, что на компьютере, на который устанавливается сервер, есть Microsoft .NET Framework (на компьютер программиста Microsoft .NET Framework устанавливается автоматически, вместе с Delphi). Следует обратить внимание, что платформа Microsoft .NET Framework используется многими современными программами, и вполне вероятно, что она уже есть на компьютере пользователя. Microsoft .NET Framework также гарантированно есть на компьютере, если на нем установлена ОС Windows Vista или Windows 7.

Сначала на диск компьютера (лучше в отдельный каталог) надо скопировать файлы BSQLServer.exe, Borland.Data.Blackfish.LocalClient.dll, BSQLServer.exe.config, а также файл лицензии (с расширением slip). После этого следует определить каталог, предназначенный для файлов баз данных. Имя этого каталога надо указать в файле конфигурации сервера (BSQLServer.exe.config) в качестве значения ключа `blackfisfsql.dataDirectory`, например:

```
<add key="blackfishsql.dataDirectory" value="D:\Database\" />
```

Следует обратить внимание, что если значение ключа `blackfisfsql.dataDirectory` задано, то клиент для доступа к базе данных вместо абсолютного пути может указать относительный (для чего в строке `Database` файла параметров соединения с базой данных путь к базе данных нужно заменить макросом `|DataDirectory|`).

После этого надо открыть окно командной строки и набрать команду

```
Каталог\BSQLServer.exe –install
```

где *Каталог* — имя каталога, в котором находятся файлы Blackfish SQL Server.

Эта команда обеспечивает установку Blackfish SQL Server как службы. Теперь при каждом включении сервера (компьютера) Blackfish SQL Server будет запускаться автоматически.

Замечание

При установке службы Blackfish SQL Server в Windows Vista окно командной строки надо открыть от имени администратора.

Следует обратить внимание на то, что по умолчанию брандмауэр Windows блокирует доступ клиента (программы работы с базой данных) к серверу. Поэтому после установки Blackfish SQL Server необходимо в список исключений брандмауэра добавить ссылку на BSQLServer (рис. 6.25).

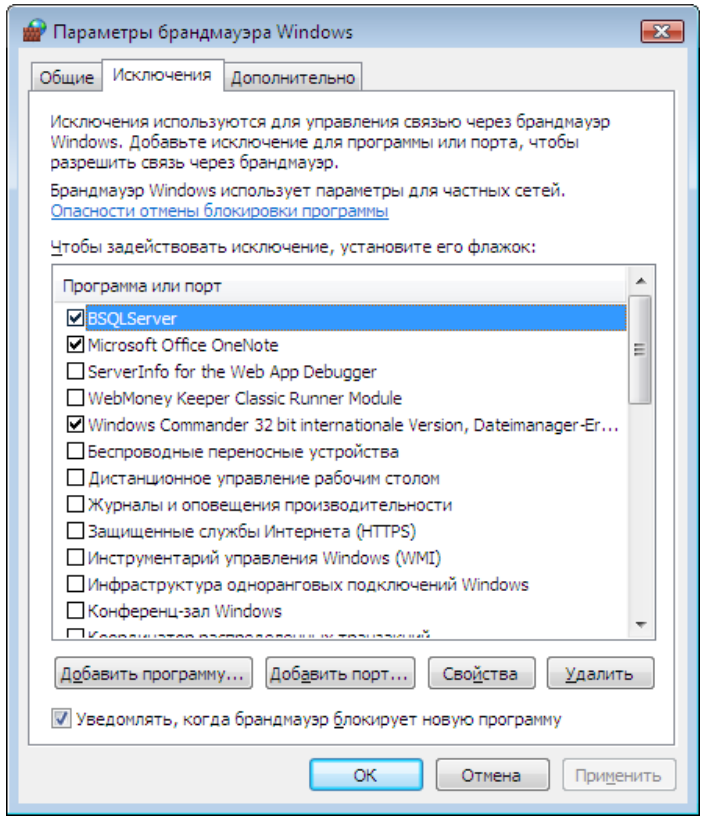


Рис. 6.25. Настройка брандмауэра

Установка программы работы с базой данных

Процесс установки программы работы с базой данных Blackfish SQL Server на компьютер пользователя состоит из двух шагов. Сначала на диск компьютера пользователя (лучше в отдельный каталог) надо скопировать файлы, образующие программу работы с базой данных (список файлов программы работы с базой данных "Книги" приведен в табл. 6.20).

Таблица 6.20. *Файлы программы работы с базой данных "Книги"*

Файл	**Комментарий**
bf_books.exe	Программа работы с базой данных "Книги"
midas.dll	Библиотека, реализующая функциональность компонента `ClientDataSet`

Таблица 6.20 *(окончание)*

Файл	Комментарий
dbxconnections.ini	Файл параметров соединения `Books` (копия файла с компьютера разработчика). Заметим, что информацию о соединениях, не используемых программой, из файла dbxconnections.ini рекомендуется удалить
dbxdrivers.ini	Файл параметров драйвера базы данных

Затем надо внести изменения в файл параметров соединения dbxconnections.ini — указать имя компьютера, на котором работает сервер Blackfish SQL Server. Если сервер и программа работы с базой данных работают на одном компьютере, то в качестве значения параметра `HostName` надо указать `localhost`. Если программа работы с базой данных и сервер работают на разных компьютерах, то в качестве значения параметра `HostName` надо указать сетевое имя компьютера, на котором работает сервер. Кроме этого в файл C:\Windows\System32\drivers\ets\hosts рабочей станции надо записать имя и IP-адрес сервера (узнать IP-адрес сервера можно запустив на сервере утилиту ipconfig.exe).

В качестве примера в листинге 6.10 приведен фрагмент файла dbxconnections.ini, а в листинге 6.11 — файла hosts. Обратите внимание, что в файле конфигурации указан макрос `|DataDirectory|`, поэтому в файле bsqlserver.exe.config должно быть указано значение ключа `blackfisfsql.dataDirectory`.

Листинг 6.10. Параметры соединения с базой данных "Книги" (dbxconnections.ini)

```
[Books]
drivername=BLACKFISHSQL
User_Name=guest
port=2508
create=False
readonlydb=False
HostName=danila
Database=|DataDirectory|Books
```

Листинг 6.11. Файл hosts

```
# (C) Корпорация Майкрософт (Microsoft Corp.), 1993-1999
#
# Это образец файла hosts, используемый Microsoft TCP/IP для Windows.
#
# Этот файл содержит сопоставления IP-адресов именам узлов.
# Каждый элемент должен располагаться в отдельной строке.
```

```
# IP-адрес должен находиться в первом столбце, за ним должно следовать
# соответствующее имя.
# IP-адрес и имя узла должны разделяться хотя бы одним пробелом.
#
# Кроме того, в некоторых строках могут быть вставлены комментарии
# (такие, как эта строка), они должны следовать за именем узла
# и отделяться от него символом '#'.
#
# Например:
#
#      102.54.94.97     rhino.acme.com          # исходный сервер
#       38.25.63.10     x.acme.com              # узел клиента x

127.0.0.1       localhost
192.168.1.2     danila
```

Глава 7

Компонент программиста

Программист может создать собственный компонент, поместить его на одну из вкладок палитры компонентов и использовать при разработке приложений точно так же, как и другие компоненты Delphi.

Новый компонент проще всего можно создать путем расширения возможностей уже существующего компонента. Например, компонент, обеспечивающий ввод (редактирование) числа, можно создать на основе компонента `Edit`, предназначенного для ввода и редактирования текста.

Процесс создания компонента рассмотрим на примере. Создадим компонент (назовем его `NkEdit`), который внешне ничем не будет отличаться от стандартного поля редактирования (компонента `Edit`), но в его поле можно будет ввести только число.

Приступая к разработке нового компонента, следует четко сформулировать его назначение. Затем необходимо определить, какой из существующих компонентов наиболее близок по своему назначению и функциональным возможностям к компоненту, который надо создать. Именно этот компонент следует выбрать в качестве базового.

Так как компонент `NkEdit` предназначен для ввода и редактирования числа (строки символов, содержащей только цифры и, возможно, десятичный разделитель), то в качестве базы для нового компонента `NkEdit` выберем компонент `Edit`. Очевидно, что у нового компонента должны быть и новые свойства (табл. 7.1), обеспечивающие решение поставленной задачи.

Таблица 7.1. *Свойства компонента* `NkEdit`

Свойство	Тип	Описание
`MaxLenInt`	`Integer`	Максимальное количество цифр целого числа или количество цифр целой части дробного числа, которое можно ввести в поле редактирования. По умолчанию значение свойства равно 6
`MaxLenFrac`	`Integer`	Максимальное количество цифр дробной части числа, которое можно ввести в поле редактирования. По умолчанию значение свойства равно 2

***Таблица 7.1** (окончание)*

Свойство	Тип	Описание
`OnlyPositive`	`Boolean`	Если значение свойства равно `True`, то в поле редактирования можно ввести только положительное число. По умолчанию значение свойства равно `False`
`Value`	`Single`	Значение, соответствующее строке, введенной в поле редактирования
`NextControl`	`TWinControl`	Элемент управления (компонент), на который перемещается фокус в результате нажатия в поле редактирования клавиши <Enter>

Модуль компонента

Замечание

Перед тем как приступить к работе по созданию нового компонента, рекомендуется создать новый каталог для модуля компонента.

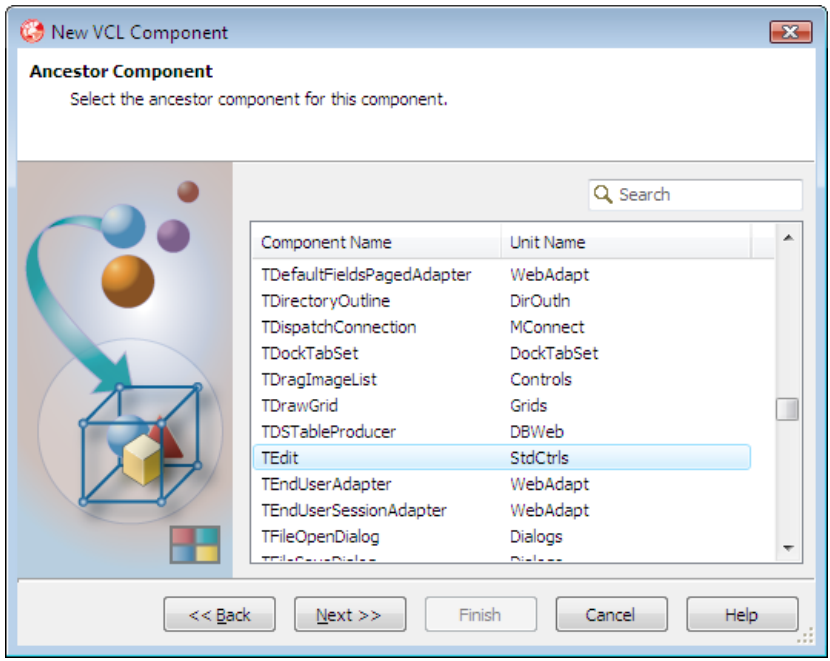


Рис. 7.1. Выбор базового класса для нового компонента

Чтобы начать работу над новым компонентом, надо в меню **Components** выбрать команду **New VCL Component**. Затем в появившемся окне **New VCL Component** следует выбрать базовый класс для создаваемого компонента. Так как компонент `NkEdit` разрабатывается на основе компонента `Edit`, то в качестве базового класса надо указать класс `TEdit`.

В следующем окне (рис. 7.2), которое становится доступным в результате щелчка на кнопке **Next**, надо задать имя класса (поле **Class Name**) компонента, вкладку палитры компонентов, на которую будет помещен значок компонента (поле **Palette Page**), и имя модуля (поле **Unit name**) создаваемого компонента.

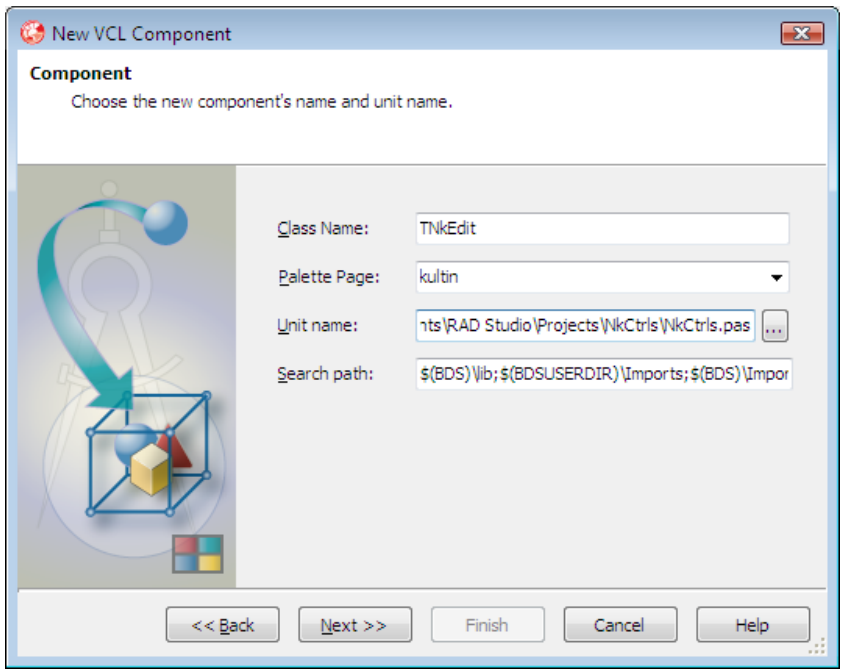


Рис. 7.2. В поле **Unit name** надо ввести имя модуля создаваемого компонента

Согласно принятому в Delphi соглашению имя класса (типа) должно начинаться с буквы "T". Поэтому в поле **Class Name** надо ввести `TNkEdit`.

По умолчанию компоненты, созданные программистом, устанавливаются на вкладку **Samples**. Если в поле **Palette Page** ввести имя еще не существующей вкладки, то в процессе установки компонента вкладка с указанным именем будет создана.

После того как будут заданы класс компонента, страница палитры компонентов и имя модуля, надо нажать кнопку **Next**. Далее, в следующем окне, надо выбрать переключатель **Create Unit** (Создать модуль) и сделать щелчок на кнопке **Finish**. В результате описанных действий будет создан модуль пакета компонентов (листинг 7.1).

Листинг 7.1. Шаблон модуля компонентов (NkCtrls.pas)

```
unit NkCtrls;

interface

uses
  SysUtils, Classes, Controls, StdCtrls;

type
  TNkEdit = class(TEdit)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('kultin', [TNkEdit]);
end;

end.
```

В созданный Delphi модуль компонента нужно добавить объявления свойств, полей, процедур и функций, обеспечивающих доступ к полям. Если на какое-либо событие компонент должен реагировать не так, как базовый, то в объявление класса нужно поместить объявление соответствующей процедуры обработки события.

В листинге 7.2 приведен модуль компонента `NkEdit` после внесения всех необходимых дополнений.

Листинг 7.2. Модуль компонентов (NkCtrls.pas)

```
unit NkCtrls;

interface
uses
  SysUtils, Classes, Controls, StdCtrls;
```

```
type
  TNkEdit = class(TEdit)
  private
    FOnlyPositive: Boolean; // True — в поле компонента можно ввести
                            // только положительное число
    FMaxLenInt: Integer;    // допустимое количество цифр целой части
    FMaxLenFrac: Integer;   // допустимое количество цифр дробной части

    FNextControl: TWinControl;   // компонент, на который перемещается
                                 // фокус в результате нажатия <Enter>

    procedure SetfValue(value: single);
    function GetfValue: single;

  protected
     // процедура обработки события KeyPress
    procedure KeyPress(var Key: char); override;

  public
    // конструктор
    constructor Create(AOwner: TComponent); override;

  published
    // объявления свойств
    property OnlyPositive: boolean
                read  FOnlyPositive
                write FOnlyPositive
                default False;

    property MaxLenInt: integer
                read FMaxLenInt
                write FMaxLenInt
                default 6;

    property MaxLenFrac: integer
                read FMaxLenFrac
                write FMaxLenFrac
                default 6;

    property Value: single
                 read GetfValue
                 write SetfValue;
```

```
    property NextControl: TWinControl
                  read FNextControl
                  write FNextControl
                  default Nil;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Kultin_2010', [TNkEdit]);
end;

// конструктор
constructor TNkEdit.Create(AOwner: TComponent);
begin
  // вызвать конструктор базового класса
  inherited Create(AOwner);

  // чтобы в поле компонента не отображалось его имя
  ControlStyle := ControlStyle - [csSetCaption];

  // конструктор имеет прямой доступ к полям
  FMaxLenInt := 6;
  FMaxLenFrac := 2;
  FOnlyPositive := False;

  // свойство Text, унаследованное от базового класса
  Text := '0';

end;

// устанавливает значение свойства Value
procedure TNkedit.SetfValue(value: Single);
begin
  Text := FloatToStrF(value,ffFixed,self.FMaxLenInt,self.MaxLenFrac);
end;

// возвращает значение свойства Value
function TNkEdit.GetfValue : single;
```

```
begin
  try
    GetfValue := StrToFloat(Text);
  except
    on e: EConvertError do
       begin
          GetfValue := 0;
          Text := '0';
       end;
  end;
end;

// нажатие клавиши в поле редактирования компонента NkEdit
procedure TNkEdit.KeyPress(var Key: Char);
var
  n : integer; // количество цифр
  p : integer; // позиция десятичного разделителя
begin
  case Key of
    '0'..'9':
    begin
        p := Pos(DecimalSeparator,self.Text);
        if (p = 0) or (self.SelStart < p)  then
            // цифра целой части
            // (SelStart — позиция курсора в поле редактирования)
            begin
                if p = 0
                    then n := Length(self.Text)
                    else n := p -1;
                if (n > 0) and (self.Text[1] = '-') then
                   n := n-1;
                if n >= self.MaxLenInt then
                   Key := #0;
            end
        else
            // цифра дробной части
            begin
              n := Length(self.Text) - p;
              if n >= self.MaxLenFrac then
                 Key := #0;
            end;
    end;
```

```
    #8,#9: ;
    #13: if NextControl <> Nil
            then NextControl.SetFocus;

    ',','.':
        begin
          Key := DecimalSeparator;
          if (Pos(DecimalSeparator,self.Text) <> 0) or
             (self.MaxLenFrac = 0) then
                Key := #0;
        end;

    '-':
        if (FOnlyPositive) or (self.SelStart > 0) or
           (Pos('-',self.Text) = 1) then
            Key := #0;

    else // остальные символы не отображать
      key:= #0;
  end;

  // вызов процедуры обработки события KeyPress базового класса
  inherited KeyPress(Key);
end;

end.
```

В секции `private` класса `TNkEdit` объявлены поля `FOnlyPositive`, `FMaxLenInt` и `FMaxLenFrac` (имена полей, согласно принятому в Delphi соглашению, начинаются с буквы `F`, от англ. *Field* — поле). Поля `FOnlyPositive`, `FMaxLenInt`, `FMaxLenFrac` и `FNextContol` хранят характеристики компонента: `FOnlyPositive` — вид числа (положительное или отрицательное), которое можно ввести в поле редактирования; `FMaxLenInt` — максимально допустимое количество цифр целой части числа; `FMaxLenFrac` — максимально допустимое количество цифр дробной части числа; `FnextControl` — имя компонента, на который будет перемещен фокус в результате нажатия в поле редактирования клавиши <Enter>. Следует обратить внимание, что поля объявлены в секции `private`, поэтому у программ, которые будут использовать компонент `NkEdit`, доступа к ним не будет.

В секции `published` объявлен конструктор. Объявление

```
constructor Create(AOwner: TComponent); override;
```

показывает, что конструктор класса `NkEdit` замещает конструктор базового класса (*override* — подменить).

Свойства объявлены в секции published, поэтому они будут отображаться в окне **Object Inspector**. В объявлении свойства указывается тип значения свойства, а также поле, которое хранит значение свойства, или функция и процедура, которые обеспечивают доступ к полю. Например, объявление

```
property OnlyPositive: boolean
            read  FOnlyPositive
            write FOnlyPositive
            default False;
```

показывает, что значением свойства OnlyPositive является содержимое поля FOnlyPositive, что свойство доступно как для чтения, так и для записи (значение свойства можно изменить во время работы программы), а также то, что по умолчанию значение свойства равно False.

Объявление свойства Value выглядит иначе:

```
property Value: single
                  read GetfValue
                  write SetfValue;
```

Следует обратить внимание, что после read указано имя функции, а после write — процедуры. Это значит, что значением свойства Value является значение функции GetValue, а устанавливает значение свойства Value процедура SetValue. Сами же функция GetValue и процедура SetValue объявлены в секции private, поэтому программе, которая будет использовать компонент NkEdit, они, как и поля, доступны не будут.

Процедура KeyPress обрабатывает соответствующее событие и обеспечивает фильтрацию символов, которые можно ввести в поле компонента NkEdit. В объявлении процедуры указана директива override. Это значит, что она подменяет соответствующую процедуру базового компонента.

В разделе implementation находятся процедуры и функции, объявленные в объявлении класса TNkEdit.

Конструктор создает компонент (инструкция inherited Create(AOwner) вызывает конструктор базового класса) и настраивает его.

Процедура TNkEdit.KeyPress обеспечивает обработку события KeyPress. В качестве параметра эта процедура получает символ, соответствующий нажатой клавише. Если символ допустимый, то процедура "ничего не делает", а просто вызывает процедуру KeyPress базового класса (в результате чего символ появляется в поле редактирования). Если символ недопустимый, то он заменяется символом "ноль", после чего также вызывается функция KeyPress базового класса.

Следует обратить внимание на процедуру Register. Она обеспечивает регистрацию типа TNkEdit в среде разработки.

После того как в модуль будут внесены все необходимые дополнения, модуль компонента надо сохранить (имя файла модуля должно совпадать с именем модуля, указанным в директиве unit). Необходимо обратить внимание, что модуль ком-

понента не является самостоятельной программой, поэтому выполнить его компиляцию нельзя (команда **Project ▸ Compile** недоступна).

Тестирование модуля компонента

После того как будет создан модуль компонента, необходимо убедиться, что компонент работает правильно. Для этого надо создать приложение, которое будет использовать созданный компонент.

Так как компонент `NkEdit` пока еще не включен ни в один из пакетов компонентов, то его значка нет в палитре компонентов и, следовательно, поместить созданный компонент на форму привычным способом нельзя. Поэтому тестируемый компонент придется создать *в коде* — поместить в текст программы инструкции, обеспечивающие создание и настройку компонента.

Приложение, обеспечивающее тестирование модуля компонента, создается следующим образом. Сначала надо активизировать процесс создания нового приложения, настроить форму и сохранить проект в том каталоге, в котором находится модуль тестируемого компонента.

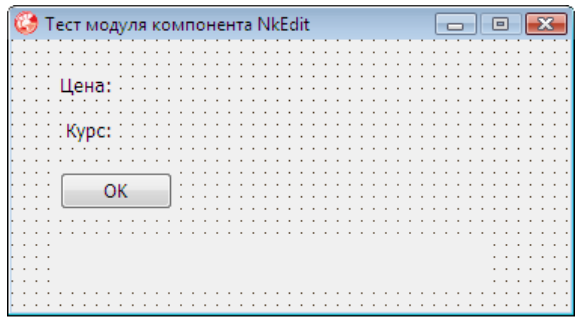


Рис. 7.3. Форма программы тестирования модуля компонента `NkEdit` (поля ввода данных будут созданы во время работы программы)

Форма программы тестирования модуля компонента `NkEdit` приведена на рис. 7.3. Она содержит три компонента `Label` и командную кнопку. Полей редактирования, предназначенных для ввода исходных данных, на форме нет. Они (два компонента `NkEdit`) будут созданы во время работы программы.

После того как проект будет сохранен, в модуль формы программы тестирования надо внести следующие дополнения (листинг 7.3):

1. В директиву `uses` добавить имя модуля тестируемого компонента — `NkCtrls`.
2. В раздел `public` объявления формы поместить объявление двух компонентов (объектов) `NkEdit`.
3. Создать процедуру обработки события `Create` и поместить в нее инструкции, обеспечивающие создание и настройку тестируемых компонентов.

Листинг 7.3. Тестирование модуля компонента `NkEdit`

```
unit NkEditTest;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls,

  // ссылка на модуль компонента
  NkCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Label3: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public

    // тестируемые компоненты
    NkEdit1 : TNkEdit;
    NkEdit2 : TNkEdit;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// обработка события Create
procedure TForm1.FormCreate(Sender: TObject);
begin
  // По умолчанию в поле компонента NkEdit можно ввести положительное
  // или отрицательное дробное число. Изменив значения свойств
  // OnlyPositive, MaxLenInt и MaxLenFrac,
  // можно выполнить "тонкую" настройку компонента
```

```
  { Внимание! Присвоить значение свойству NextControl
    компонента NkEdit1 можно только после создания компонента NkEdit2.
    В противном случае значение NextControl будет Nil. }

  // создать NkEdit1
  NkEdit1 := TNkEdit.Create(self);
  NkEdit1.Parent := self;

  // создать NkEdit2
  NkEdit2 := TNkEdit.Create(self);
  NkEdit2.Parent := self;

  // настроить NkEdit1 — поле "Цена"
  NkEdit1.OnlyPositive := True;
  NkEdit1.NextControl := NkEdit2;

  // значения свойств MaxLenInt и MaxLenFrac оставим без изменения

  // свойства Top, Left, TabOrder унаследованы от TEdit
  NkEdit1.Top := 20;
  NkEdit1.Left := 72;
  NkEdit1.TabOrder := 0;

  // настроить NkEdit2 — поле "Курс"
  NkEdit2.NextControl := Button1;
  NkEdit2.OnlyPositive := True;
  NkEdit2.MaxLenInt := 2;
  NkEdit2.MaxLenFrac := 2;

  NkEdit2.Top := 50;
  NkEdit2.Left := 72;
  NkEdit2.TabOrder := 1;
end;

// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real;  // цена в долларах
  k: real;    // курс
  rub: real;  // цена в рублях
begin
  usd := NkEdit1.Value;
```

```
  k := NkEdit2.Value;
  rub := usd * k;

  Label3.Caption := NkEdit1.Text + '$ = ' + FloatToStrF(rub,ffCurrency,6,2);
end;
end.
```

Тестируемые компоненты создает и настраивает процедура обработки события `Create` формы. Создается компонент методом `Create` класса `TNkEdit`. Настройка компонента выполняется путем изменений его свойств. Изменить можно как значения уникальных свойств компонента `NkEdit`, так и значения свойств, унаследованных от "родителя", компонента `Edit`. Необходимо обратить внимание на идентификатор `self`. Внутри процедуры обработки события он обозначает объект, событие которого обрабатывает процедура, т. е. форму. Идентификатор `self` передается конструктору объекта в качестве параметра, и тем самым выполняется привязка созданного объекта к форме. Следует обратить внимание, что свойству `Parent` вновь созданного компонента обязательно надо присвоить значение `self`. Если этого не сделать, то созданный компонент на форме не появится.

На рис. 7.4 приведено окно программы тестирования модуля компонента `NkEdit`.

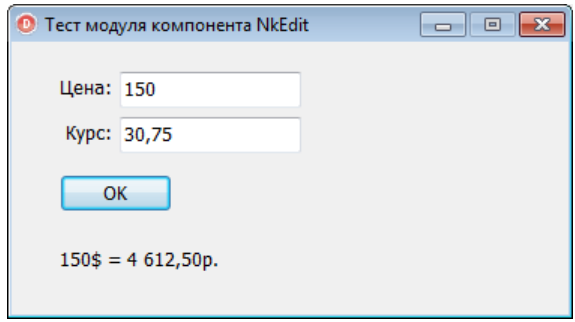


Рис. 7.4. Тестирование модуля компонента: поля ввода данных — компоненты `NkEdit`

Пакет компонентов

Для того чтобы значок компонента появился на одной из страниц палитры компонентов, его надо поместить в пакет (Package) компонентов. *Пакет компонентов* — это динамическая библиотека, в которой находится код, реализующий функциональность компонентов.

Различают пакеты времени разработки (Designtime package) и пакеты времени выполнения (Runtime package). Пакеты времени разработки используются средой разработки, пакеты времени выполнения — программами, использующими компоненты. Кроме пакетов указанных типов существуют универсальные пакеты (Design-

time and runtime package), которые могут использоваться как средой разработки, так и приложениями.

Программист может поместить свой компонент в один из уже существующих пакетов или создать новый пакет и затем поместить в него свой компонент.

Создание пакета компонентов

Как было сказано, для того чтобы компонент был доступен разработчику, он должен находиться в пакете компонентов. Компонент программиста можно добавить в один из существующих пакетов или в новый, специально созданный, пакет. Рассмотрим, как создать пакет компонентов и поместить в него компонент.

Чтобы создать пакет, надо в меню **File** выбрать команду **New ▸ Package - Delphi**. В результате чего будет создан новый модуль — пакет (чтобы увидеть модуль, надо в меню **Project** выбрать команду **View Source**). Созданный пакет рекомендуется сразу сохранить в том каталоге, где находится модуль компонента, который предполагается поместить в пакет. Для этого надо в меню **File** выбрать команду **Save Project** и указать в качестве имени проекта предполагаемое имя пакета (например, NkPackage).

После того как пакет (проект создания пакета) будет сохранен, в него надо добавить модуль компонента — в меню **Project** выбрать команду **Add to Project** и в появившемся окне указать модуль компонента. В результате этого в списке **Contains** структуры пакета, которая отображается в окне **Project Manager**, появится имя добавленного модуля (рис. 7.5).

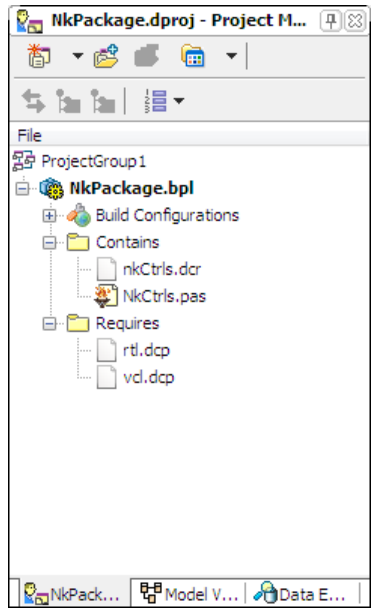


Рис. 7.5. Структура пакета отображается в окне **Project Manager**

По умолчанию для изображения созданного программистом компонента на странице палитры компонентов используется значок компонента родительского класса. Если вы хотите, чтобы созданный компонент выглядел иначе, то надо создать файл ресурсов и поместить в него битовый образ (размер 24×24 пиксела), который будет изображать компонент в палитре. В файл ресурсов можно поместить три битовых образа: 16×16, 24×24 и 32×32. Битовый образ 24×24 должен иметь имя, совпадающее с именем типа компонента. У битовых образов 16×16 и 32×32 должны быть соответственно суффиксы 16 и 32.

Создать файл ресурсов можно, например, с помощью утилиты Image Editor (рис. 7.6). После того как файл ресурсов будет создан, в файл проекта (он открывается командой **Project ▸ View Source**) создания пакета компонентов надо добавить ссылку (директива `{$R}`) на этот файл (листинг 7.4).

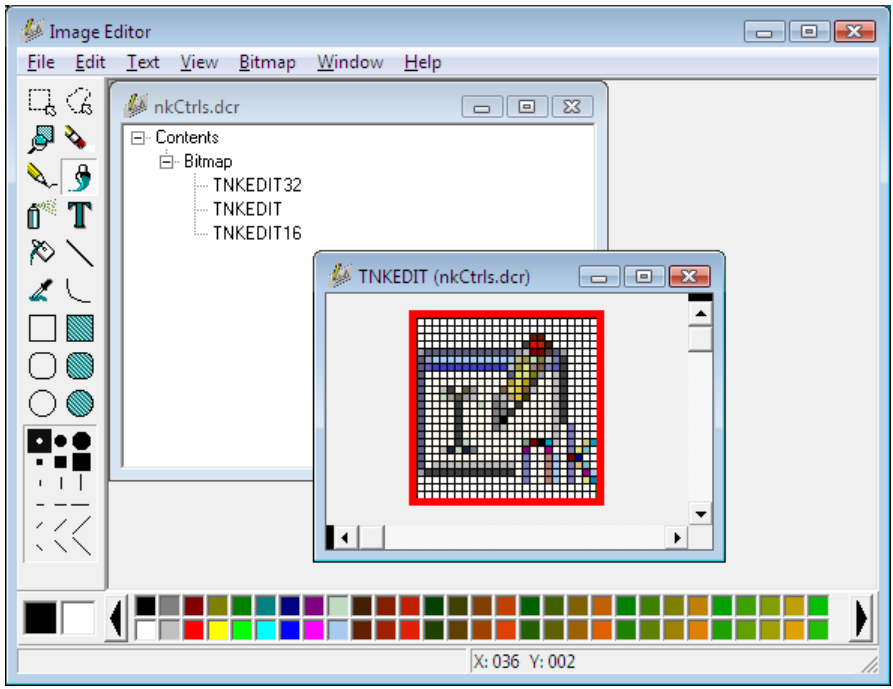


Рис. 7.6. Файл ресурсов пакета

Листинг 7.4. Файл проекта

```
package NkPackage;

{$R *.res}
{$R 'nkCtrls.dcr'}
{$ALIGN 8}
```

```
{$ASSERTIONS ON}
{$BOOLEVAL OFF}
{$DEBUGINFO ON}
{$EXTENDEDSYNTAX ON}
{$IMPORTEDDATA ON}
{$IOCHECKS ON}
{$LOCALSYMBOLS ON}
{$LONGSTRINGS ON}
{$OPENSTRINGS ON}
{$OPTIMIZATION ON}
{$OVERFLOWCHECKS OFF}
{$RANGECHECKS OFF}
{$REFERENCEINFO OFF}
{$SAFEDIVIDE OFF}
{$STACKFRAMES OFF}
{$TYPEDADDRESS OFF}
{$VARSTRINGCHECKS ON}
{$WRITEABLECONST OFF}
{$MINENUMSIZE 1}
{$IMAGEBASE $400000}
{$DESCRIPTION 'nk_package'}
{$IMPLICITBUILD ON}

requires
  rtl,
  vcl;

contains
  NkCtrls in 'NkCtrls.pas';

end.
```

Компиляция пакета компонентов

Перед тем как выполнить компиляцию пакета, рекомендуется задать имя пакета. Его надо ввести в поле в окне **Project Options** в разделе **Description** (рис. 7.7).

После того как пакет будет создан и в него будут помещены модули компонентов, можно выполнить компиляцию пакета — выбрать в меню **Project** команду **Compile**. Результатом компиляции является пакет (bpl-файл) и компилированный модуль пакета (dcp-файл).

Пакет (он создается в папке C:\Users\Public\Documents\RAD Studio\8.0\Bpl) необходим для компиляции и выполнения приложений в режиме использования run-

time-пакетов (Build with runtime packages). Компилированный модуль пакета (он создается в папке C:\Users\Public\Documents\RAD Studio\8.0\Dcp) необходим для компиляции приложений, использующих компоненты пакета, в режиме включения всего кода в exe-файл (Without runtime packages).

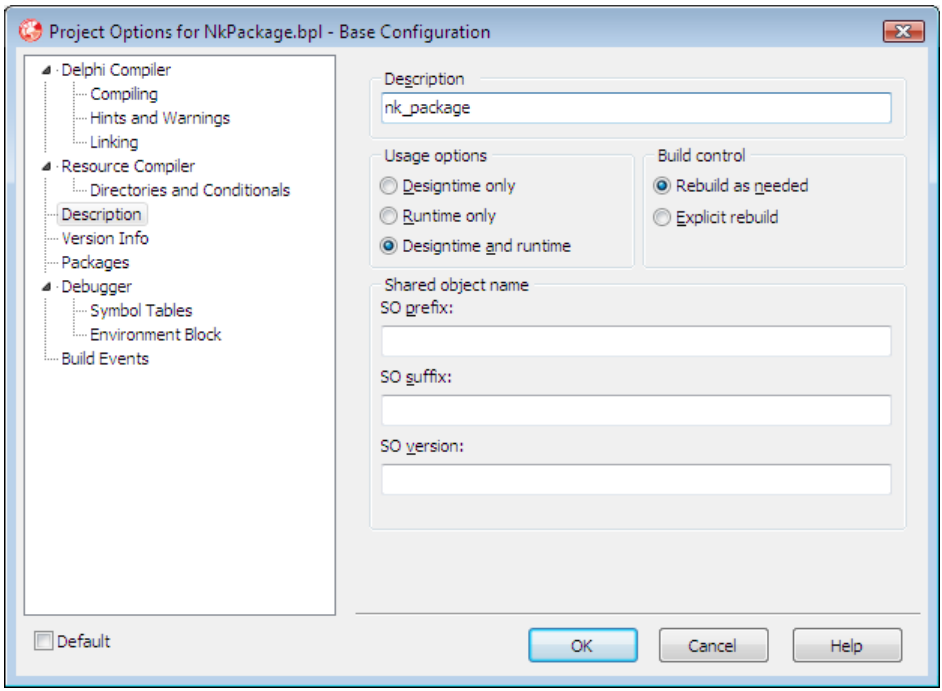


Рис. 7.7. Имя пакета надо ввести в поле **Description**

Установка пакета компонентов

Для того чтобы значок компонента, находящегося в пакете, появился в палитре компонентов, пакет надо установить.

Чтобы установить пакет, надо в меню **Component** выбрать команду **Install Packages**, в появившемся окне **Install Packages** нажать кнопку **Add**, затем в следующем окне открыть папку, в которой находится пакет, и выбрать bpl-файл. В результате описанных действий пакет будет установлен (рис. 7.8).

Следует обратить внимание на то, что палитра компонентов отображается только в режиме конструирования формы. Поэтому, чтобы убедиться, что компонент, который находится в установленном пакете, действительно установлен и стал доступен, надо активизировать процесс создания нового приложения VCL Forms. Если процесс установки был выполнен верно, в палитре компонентов на вкладке, которая указана в процедуре регистрации компонента, должен появиться значок компонента (рис. 7.9).

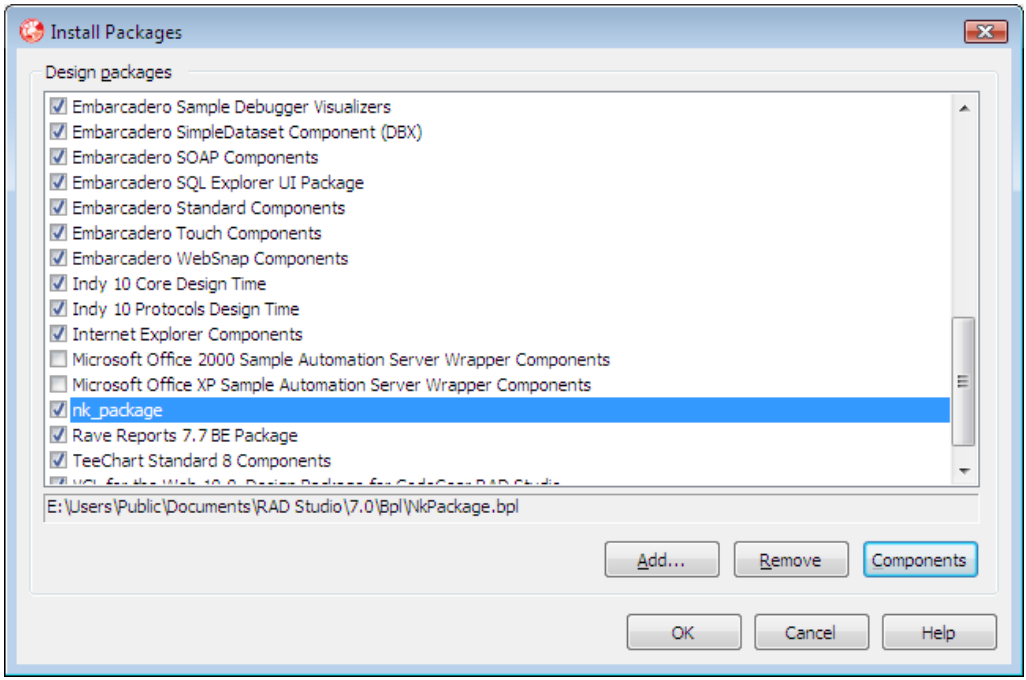


Рис. 7.8. Пакет `nk_package` установлен

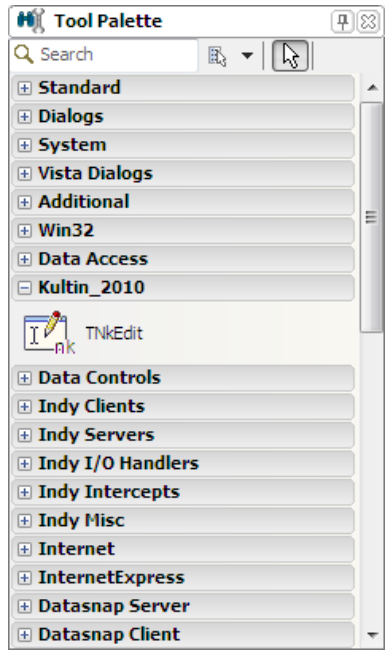


Рис. 7.9. Значок компонента `NkEdit` отображается на вкладке **Kultin_2010**

Замечание

Возможна ситуация, когда после установки компонента и последующей активизации процесса создания нового приложения (после перезагрузки Delphi) вместо уникального значка компонента (или значка компонента базового класса) отображается значок "компонент". Причина этого в том, что битовые образы значков кэшируются в реестре (раздел HKEY_CURRENT_USER\Software\CodeGear\BDS\8.0\Palette\Cashe) и загружаются оттуда. Если по какой-либо причине при первой установке пакета (в процессе отладки) значок не был создан правильно, то при последующих загрузках Delphi будет загружаться именно этот, неправильный значок, даже в том случае, если ошибка устранена (сразу после установки пакета отображается правильный значок). Для устранения описанной ситуации в командной строке запуска Delphi надо указать параметр `-nocache`.

Для компиляции приложения, использующего компоненты пакета, необходим модуль компонентов (dcu-файл). Если предполагается, что компонент будет использоваться несколькими приложениями, то модуль компонента рекомендуется поместить в общедоступную папку, например в C:\Users\Public\Documents\RAD Studio\8.0\Dcu (ее надо создать), и указать ее имя в списке папок, в которых компилятор выполняет поиск кода. Имя папки надо ввести в окне **Search path** (рис. 7.10), которое становится доступным в результате активизации процесса коррекции списка **Search path** параметров компилятора (вкладка **Delphi Compiler** окна **Project Options**).

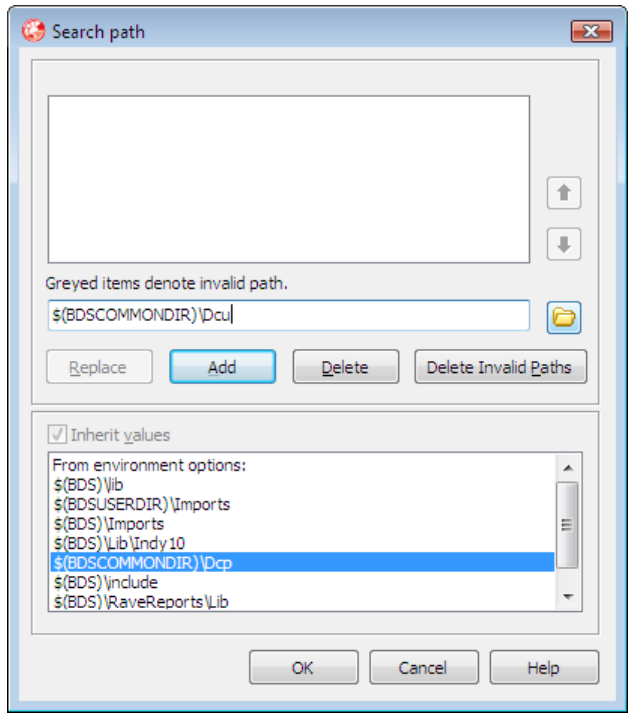


Рис. 7.10. Список папок, в которых компилятор ищет код, необходимый в процессе компиляции

Тестирование компонента

После того как значок компонента `NkEdit` появится в палитре компонентов, необходимо проверить поведение компонента во время разработки приложения (работоспособность компонента была проверена раньше, когда компонент добавлялся на форму приложения динамически, во время работы программы).

Можно считать, что компонент работает правильно, если во время разработки приложения удалось поместить компонент на форму и, используя **Object Inspector**, установить значения свойств, причем как новых, так и унаследованных от родительского класса.

Работоспособность компонента `NkEdit` можно проверить, использовав его, например, в программе "Доход". Форма приложения приведена на рис. 7.11. После того как на форму будут добавлены компоненты, следует, используя **Object Inspector** (рис. 7.12), выполнить их настройку (табл. 7.2). После чего можно приступить к созданию процедур обработки событий.

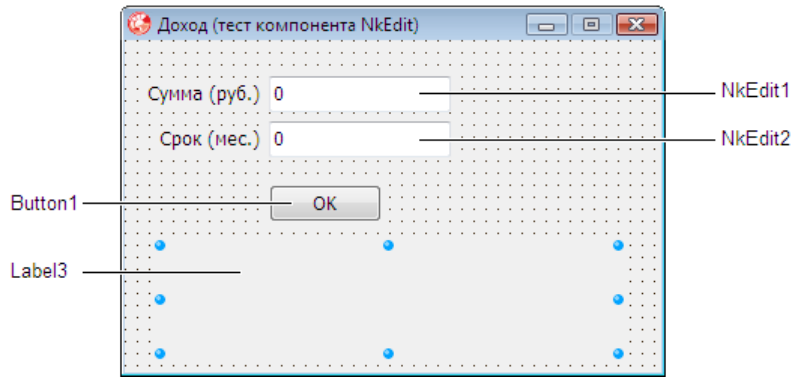


Рис. 7.11. Форма программы "Доход"

Таблица 7.2. *Значения свойств компонентов* `NkEdit`

Компонент	Свойство	Значение
NkEdit1	MaxLenInt	6
	MaxLenFrac	2
	NextControl	NkEdit2
	OnlyPositive	True
NkEdit2	MaxLenInt	2
	MaxLenFrac	0
	NextControl	Button1
	OnlyPositive	True

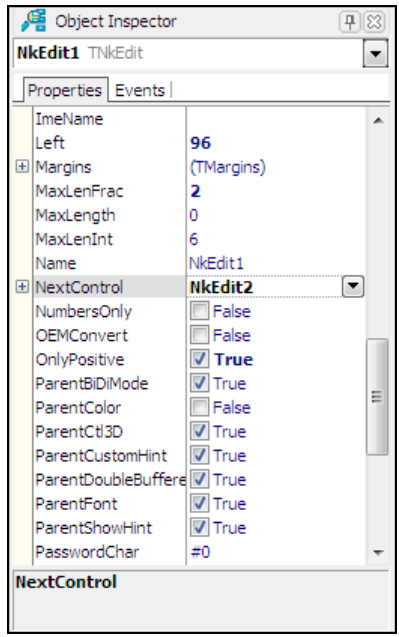


Рис. 7.12. Значения свойств `MaxLenFrac`, `MaxLenInt`, `NextControl`, `OnlyPositive` и `Value` компонента `NkEdit` можно задать в окне **Object Inspector**

В листинге 7.5 приведен модуль приложения "Доход". Здесь надо обратить внимание на следующее.

- В программе нет кода, обеспечивающего фильтрацию символов, вводимых в поля редактирования. Тем не менее во время работы программы пользователь сможет ввести в поля редактирования только положительные числа, причем в поле **Срок** можно ввести только целое число.
- В программе не используются функции `StrToFloat` и `StrToInt`. Числа, соответствующие строкам, находящимся в полях редактирования, получаются путем обращения к свойствам `Value` соответствующих компонентов `NkEdit`.

Очевидно, что использование в программе компонента `NkEdit` вместо стандартного `Edit` освобождает программиста от рутины, сокращает размер кода, делает его более понятным.

Листинг 7.5. Доход по вкладу (тест компонента `NkEdit`)

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, NkCtrls;
```

```
type
  TForm1 = class(TForm)
    NkEdit1: TNkEdit;
    NkEdit2: TNkEdit;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  sum: real;         // сумма
  period: integer;   // срок

  percent: real;    // процентов в год
  profit: real;     // доход

begin
  sum := NkEdit1.Value;
  period := Round(NkEdit2.Value);

  case period of
    1..3:   percent := 8;
    4..6:   percent := 9.5;
    7..12:  percent := 10.5;
    13..24: percent := 14;
    else percent := 17;
  end;
```

```
  profit := sum * (percent/100/12*period);

  Label3.Caption :=
        'Сумма: ' + FloatToStrF(sum,ffCurrency,6,2) + #13 +
        'Срок: ' + IntToStr(period) + 'мес.' + #13 +
        'Процент: ' + FloatToStrF(percent,ffFixed,2,2) + '%' +#13 +
        'Доход: ' + FloatToStrF(profit,ffCurrency,6,2);
end;

end.
```

Установка программы на другой компьютер

Программе, созданной в Delphi XE, необходимы как минимум две библиотеки: rtl150.bpl и vcl150.bpl (согласно принятому соглашению после имени библиотеки указывается номер ее версии, соответствующий номеру версии среды разработки). На компьютер разработчика эти библиотеки устанавливаются в процессе инсталляции Delphi. Если программа, использующая компонент, созданный программистом, скомпилирована в режиме использование пакетов времени выполнения (Build with runtime packages), то помимо стандартных библиотек для ее работы необходим пакет (bpl-файл), в котором находится компонент. На компьютере пользователя библиотеки, необходимые для работы приложения, можно поместить в каталог C:\Windows\system32 или в тот каталог, в котором находится exe-файл программы (программа ищет необходимые ей библиотеки сначала в том каталоге, из которого она запущена, а затем в каталоге system32).

Следует обратить внимание на то, что программист может существенно облегчить себе жизнь, если включит весь код, необходимый для работы программы, в exe-файл. Для этого, перед тем как выполнить компиляцию, надо на вкладке **Packages** окна **Project Options**, которое становится доступным в результате выбора в меню **Project** команды **Options**, сбросить флажок **Build with runtime packages**.

Распространение компонента

Созданный компонент можно передать коллегам-программистам для того, чтобы они могли использовать его в своих разработках. В установочный комплект надо включить bpl- и dcp-файлы пакета, dcu-файлы модулей компонентов и инструкцию по установке. Если пакет распространяется на коммерческой основе, то рекомендуется создать установщик.

Глава 8

Справочная информация

Разработчик программы должен позаботиться о том, чтобы пользователю было удобно работать с программой, чтобы при возникновении какой-либо проблемы он мог быстро получить рекомендацию по ее устранению, ответ на возникший вопрос. Решить обозначенную задачу можно, создав справочную систему, обеспечивающую доступ к справочной информации.

Справочная система HTML Help

В настоящее время в большинстве прикладных программ справочная информация отображается в формате HTML Help по запросу пользователя (в результате нажатия клавиши <F1>, выбора соответствующей команды в меню **Справка** или нажатия кнопки **Справка** в диалоговом окне) в отдельном окне. Как правило, это окно разделено на две части: в левой части отображается список разделов справочной информации (оглавление), в правой — содержимое выбранного раздела (рис. 8.1).

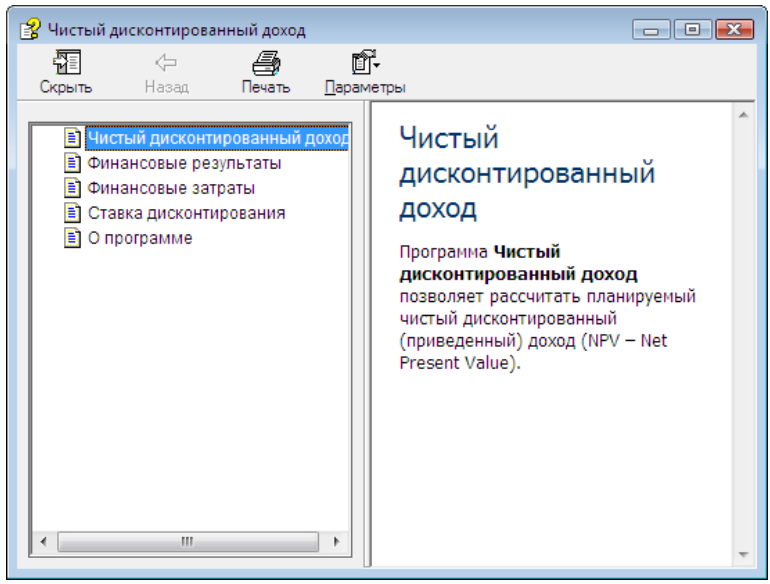


Рис. 8.1. Пример справочной информации в формате HTML Help

Основой справочной системы формата HTML Help являются *компилированные* HTML-документы (chm-файлы) — совокупность отдельных HTML-страниц справочной информации. Помимо HTML-страниц chm-файл обычно содержит список названий разделов справочной информации (оглавление) и предметный указатель (индекс).

Отображение справочной информации формата HTML Help обеспечивает системная утилита hh.exe. Таким образом, задача создания справочной системы приложения сводится к задаче создания CHM-файла.

Создать chm-файл можно с помощью утилиты Microsoft HTML Help Workshop. Сначала надо подготовить справочную информацию (поместив информацию каждого раздела в отдельный HTML-файл), затем — выполнить компиляцию. Процесс преобразования исходной справочной информации в chm-файл представлен на рис. 8.2.

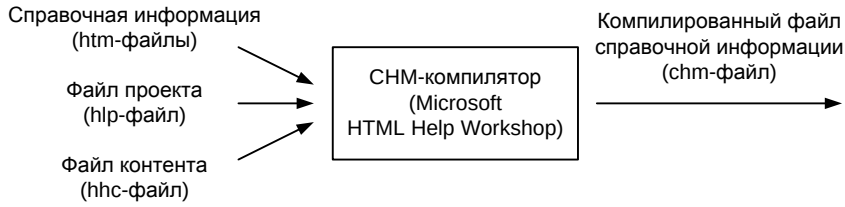


Рис. 8.2. Процесс создания chm-файла

Подготовка справочной информации

Исходным "материалом" для chm-компилятора является справочная информация, представленная в виде набора HTML-файлов.

Создать файлы справочной информации в HTML-формате можно с помощью любого HTML-редактора, в том числе и входящего в состав Delphi.

В простейшем случае всю справочную информацию можно поместить в один HTML-файл. Однако если для навигации по справочной системе предполагается использовать вкладку **Содержание**, на которой будут перечислены разделы справочной информации, то информацию каждого раздела нужно поместить в отдельный HTML-файл.

Процесс подготовки справочной информации рассмотрим на примере программы "Чистый дисконтированный доход". Пусть справочная информация будет состоять из страниц **Чистый дисконтированный доход**, **Финансовые результаты, Финансовые затраты**, **Ставка дисконтирования** и **О программе**. Следовательно, необходимо создать пять HTML-документов.

Чтобы в Delphi начать работу над новым HTML-документом, надо в меню **File** выбрать команду **New ▸ Other ▸ Web Documents ▸ HTML Page**. В результа-

те откроется вкладка **Design** HTML-редактора (рис. 8.3), в которой можно набирать текст.

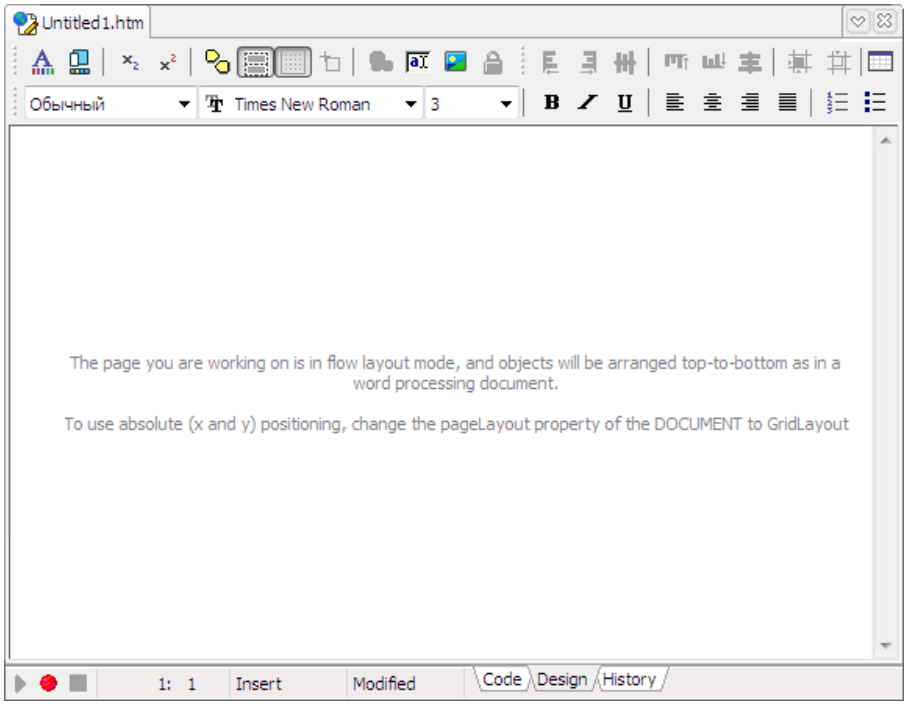


Рис. 8.3. Окно HTML-редактора (режим Design)

В верхней части окна HTML-редактора находится панель инструментов, используя которую можно задать стиль оформления абзаца, в том числе указать, что абзац является заголовком или элементом списка, выбрать шрифт, задать цвет символов, изменить стиль оформления текста (полужирный, курсив).

Заголовок страницы (раздела) справочной информации рекомендуется оформить одним из стилей **Заголовок**.

На страницу можно поместить картинку (рекомендуется заранее поместить файл картинки в каталог проекта справочной информации). Чтобы это сделать, нужно установить курсор в ту точку HTML-документа, в которую надо вставить картинку, сделать щелчок на кнопке **Insert Image** и в поле **Image Source** появившегося окна ввести имя файла иллюстрации.

После того как HTML-текст раздела справочной информации будет набран, документ надо сохранить в каталоге проекта справочной системы — выбрать в меню **File** команду **Save** и задать имя файла. Имя рекомендуется задать так, чтобы в нем присутствовал номер раздела. Например, текст раздела **Чистый дисконтированный доход** следует сохранить в файле npv_01.htm, раздела **Финансовые результаты** — в файле npv_02.htm и т. д.

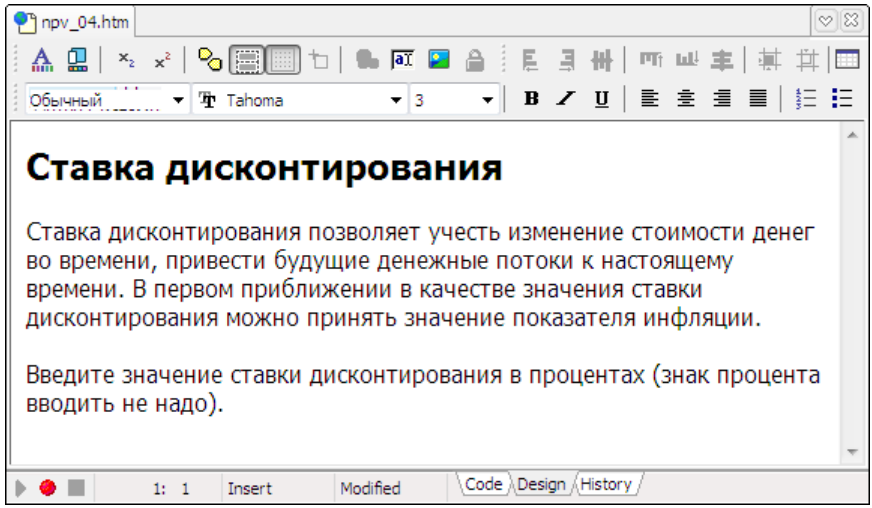


Рис. 8.4. Пример HTML-страницы справочной информации

После того как справочная информация (отдельные HTML-страницы) будет подготовлена, можно приступить к созданию chm-файла.

Microsoft HTML Help Workshop

Как было сказано, создать справочную систему формата HTML Help можно с помощью Microsoft HTML Help Workshop. Чтобы создать справочную систему (или, в простейшем случае, chm-файл) надо создать файлы проекта, контента и индекса, а затем выполнить компиляцию.

Файл проекта

Работа над новым chm-файлом начинается с создания файла проекта. Сначала в меню **File** надо выбрать команду **New ▸ Project**, затем в окне **New Project** задать имя файла проекта (файл проекта надо создать в том каталоге, в котором находятся HTML-файлы страниц справочной информации). Следует обратить внимание, что по умолчанию имя chm-файла, который будет создан в процессе компиляции, совпадает с именем файла проекта.

В качестве примера на рис. 8.5 приведено окно **HTML Help Workshop** в начале работы над проектом справочной системы программы "Доход по вкладу".

После того как файл проекта будет создан, надо сформировать список файлов, в которых находится справочная информация (предполагается, что все необходимые HTML-файлы находятся в одном каталоге, в том же, в котором сохранен файл проекта).

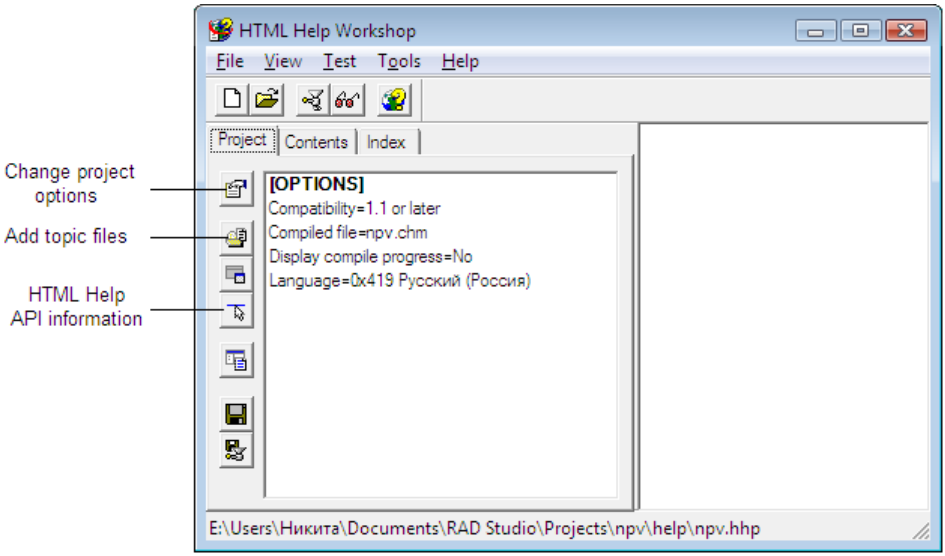


Рис. 8.5. Окно **HTML Help Workshop** в начале работы над новым проектом

Чтобы указать файлы, в которых находится справочная информация, надо:

1. Щелкнуть на кнопке **Add topic files**.
2. В появившемся окне **Topic Files** щелкнуть на кнопке **Add**.
3. В стандартном окне **Открыть** выбрать HTML-файлы, в которых находится справочная информация (нажать клавишу <Ctrl> и, удерживая ее нажатой, щелкнуть на именах нужных файлов).

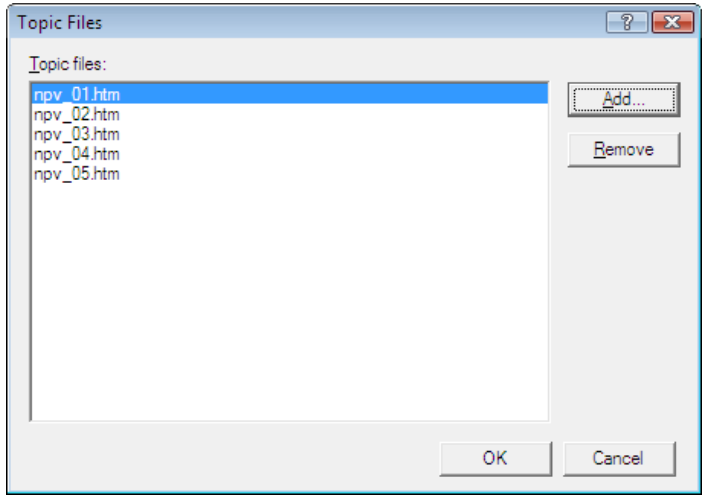


Рис. 8.6. Формирование списка файлов,
в которых находится справочная информация

В результате описанных действий в окне **Topic Files** появится список файлов (рис. 8.6), в которых находится справочная информация, а после щелчка на кнопке **OK** в файл проекта (его содержимое отображается на вкладке **Project**) будет добавлен раздел `FILES`, в котором будут перечислены HTML-файлы, содержащие справочную информацию (рис. 8.7).

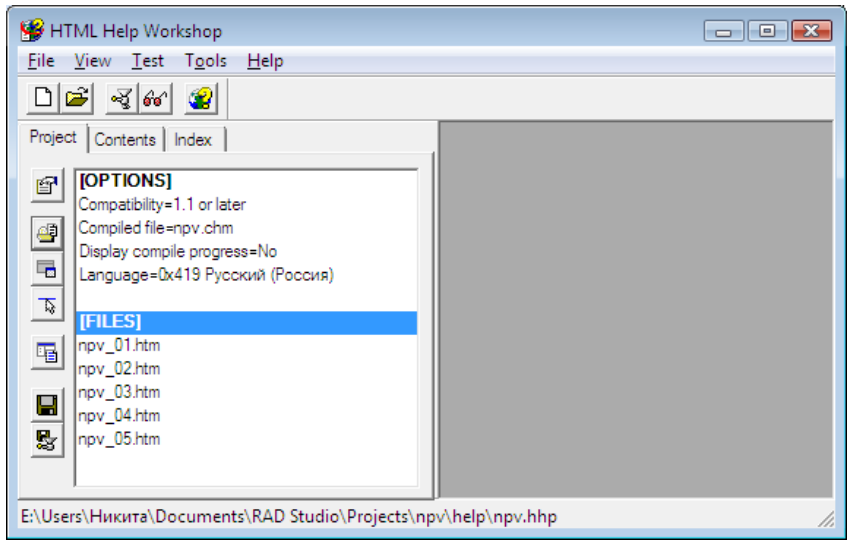


Рис. 8.7. В разделе `FILES` перечислены файлы, в которых находится справочная информация

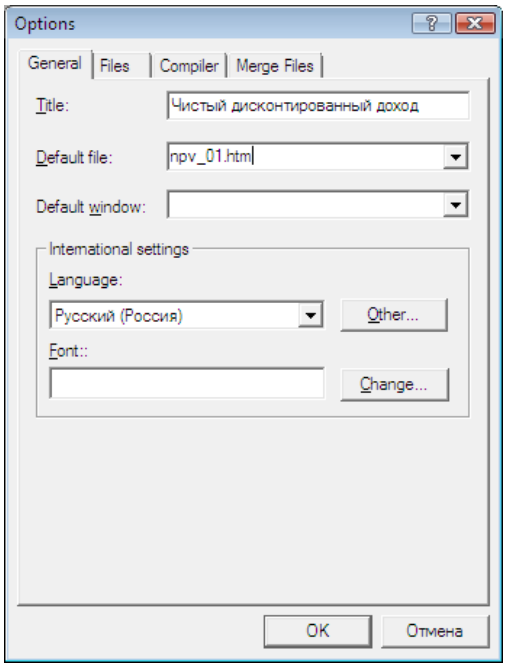


Рис. 8.8. В поле **Title** надо ввести заголовок окна справочной информации, а в поле **Default file** — имя файла главной страницы

Следующее, что нужно сделать, — задать главную (стартовую) страницу и заголовок окна справочной информации. Заголовок окна и имя файла главной страницы надо ввести соответственно в поля **Title** и **Default file** вкладки **General** окна **Options** (рис. 8.8), которое становится доступным в результате щелчка на кнопке **Change project options** (см. рис. 8.5).

Оглавление

Если для навигации по справочной системе предполагается использовать вкладку **Оглавление**, то надо создать *файл контента*. Чтобы это сделать, нужно щелкнуть на ярлыке вкладки **Contents**, подтвердить создание нового файла контента (**Create a new contents file**) и задать его имя (в качестве имени файла контента рекомендуется указать имя проекта). В результате выполнения описанных действий в каталоге проекта будет создан файл контента (hhc-файл) и станет доступной вкладка **Contents** (рис. 8.9).

Оглавление справочной информации формируется путем добавления на вкладку **Contents** ссылок на страницы справочной информации.

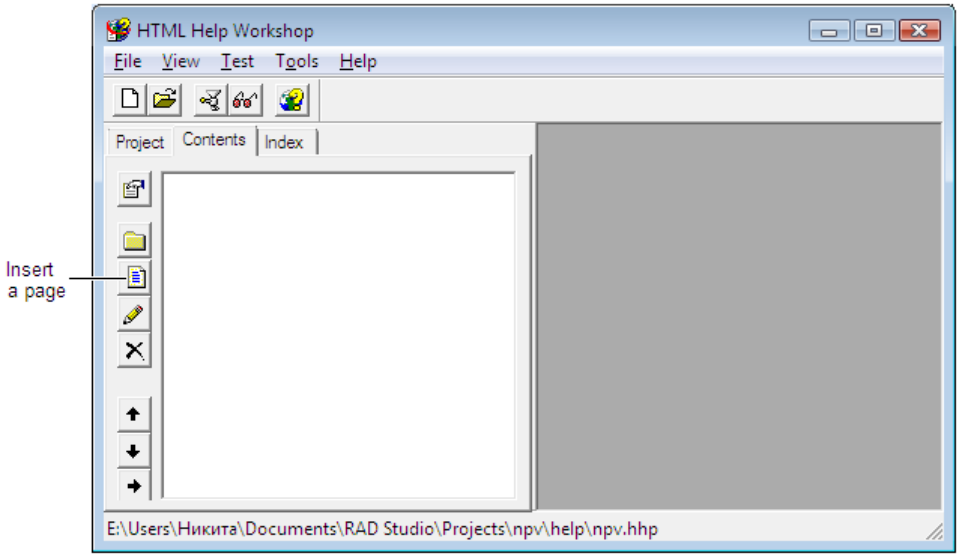


Рис. 8.9. Вкладка **Contents**

Чтобы на вкладку **Contents** добавить ссылку на страницу справочной информации, надо:

1. Щелкнуть на кнопке **Insert a page**.

2. В поле **Entry title** вкладки **General** открывшегося окна **Table of Contents Entry** ввести название раздела справочной информации и щелкнуть на кнопке **Add** (рис. 8.10).

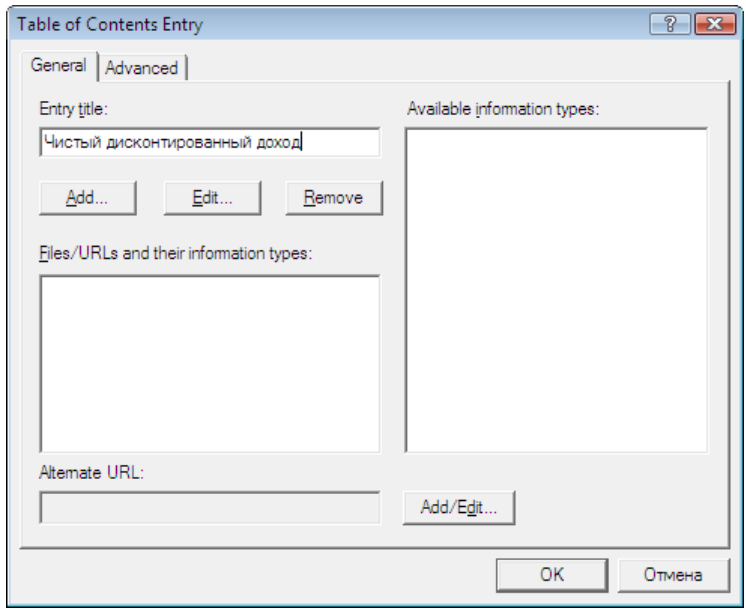


Рис. 8.10. В поле **Entry title** надо ввести название раздела и щелкнуть на кнопке **Add**

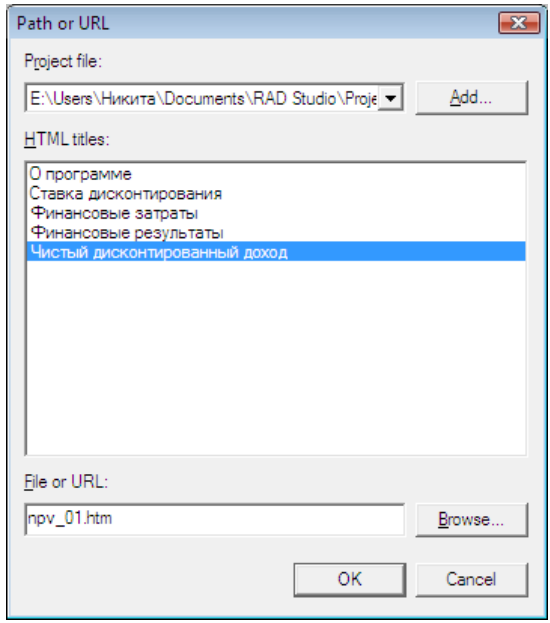


Рис. 8.11. В списке **HTML titles** надо выбрать HTML-страницу

В результате станет доступным окно **Path or URL** (рис. 8.11), в списке **HTML titles** которого будут перечислены названия HTML-документов, включенных в проект (если в HTML-файле нет тега `<Title>`, то вместо названия документа отображается имя файла).

3. В списке **HTML titles** окна **Path or URL** выбрать HTML-документ, который надо отобразить в окне справочной информации в результате выбора ссылки на страницу (названия раздела справочной информации).
4. Щелкнуть на кнопке **OK**.

Указанную последовательность действий надо выполнить для каждого раздела справочной информации.

В качестве примера на рис. 8.12 приведено окно **HTML Help Workshop**, вкладка **Contents** которого содержит оглавление справочной информации программы "Доход по вкладу".

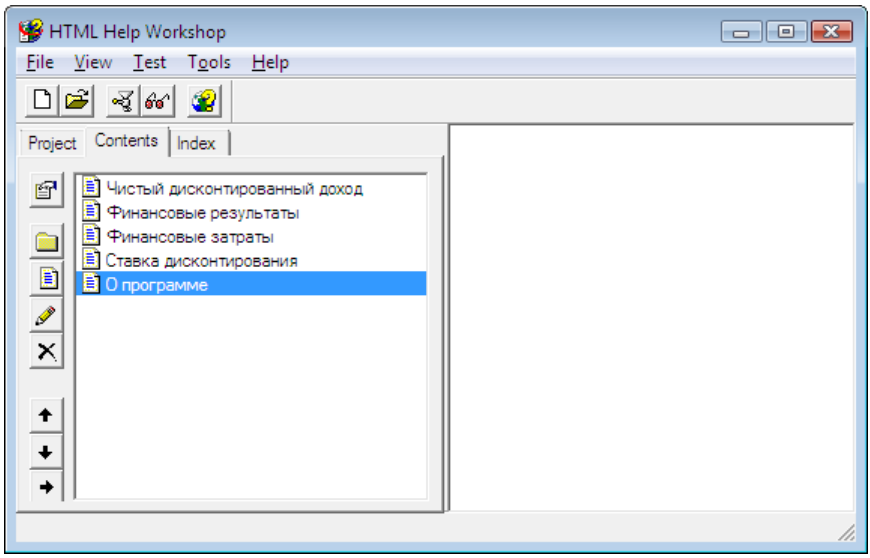


Рис. 8.12. Вкладка **Contents** содержит оглавление справочной информации

Идентификаторы разделов

Обычно когда пользователь, нажав клавишу <F1>, запрашивает справочную информацию, в окне справочной системы сразу отображается нужный раздел (страница). Такой режим отображения справочной информации называется контекстно-зависимым. Для того чтобы программист мог реализовать контекстно-зависимый режим отображения справочной информации, необходимо каждой странице назначить идентификатор.

Инструкция назначения идентификатора в общем виде выглядит так:

```
#define Страница Идентификатор; комментарий
```

Например, инструкция

```
#define npv_01 1; Чистый дисконтированный доход
```

объявляет идентификатор для страницы npv_01.htm. Следует обратить внимание, что при объявлении идентификатора расширение файла не указывается.

Инструкции объявления идентификаторов следует поместить в отдельный файл. В качестве примера в листинге 8.1 показан файл объявления идентификаторов разделов справочной информации программы "Доход по вкладу". Следует обратить внимание, что файл с расширением h — это обычный текстовый файл.

Листинг 8.1. Объявление идентификаторов разделов справочной информации (profit.h)

```
#define npv_01  1; Чистый дисконтированный доход
#define npv_02  2; Финансовые результаты
#define npv_03  3; Финансовые затраты
#define npv_04  4; Ставка дисконтирования
#define npv_05  5; О программе
```

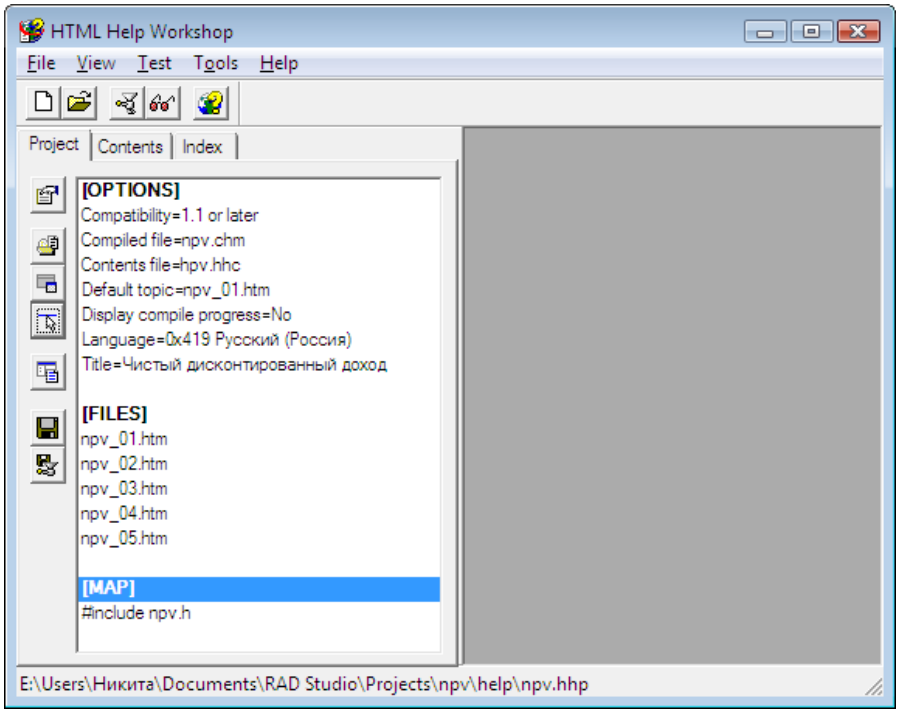


Рис. 8.13. Раздел `MAP` содержит ссылку на файл идентификаторов разделов справочной информации

После того как h-файл будет подготовлен (его следует сохранить в каталоге проекта справочной системы), надо:

1. Выбрать вкладку **Project** и сделать щелчок на кнопке **HTML Help API Information**.
2. Выбрать вкладку **Map** и щелкнуть на кнопке **Header file**.
3. В появившемся окне **Include file** ввести имя h-файла и щелкнуть на кнопке **OK**.

В результате описанных действий в файл проекта будет добавлен раздел `MAP` (рис. 8.13), в котором будет ссылка (директива `#include`) на файл объявления идентификаторов разделов справочной информации.

Компиляция

После того как будут определены файлы, в которых находится справочная информация, и подготовлена информация для формирования вкладки **Содержание** (создан файл контента), можно выполнить компиляцию.

Исходной информацией для CHM-компилятора HTML Help Workshop являются файл проекта (hpp-файл), файл контента (hhc-файл) и файлы справочной информации (HTML-файлы). Результатом компиляции является файл справочной информации (chm-файл).

Чтобы выполнить компиляцию, надо в меню **File** выбрать команду **Compile** и затем в появившемся окне **Create a compiled file** сделать щелчок на кнопке **Compile** (рис. 8.14).

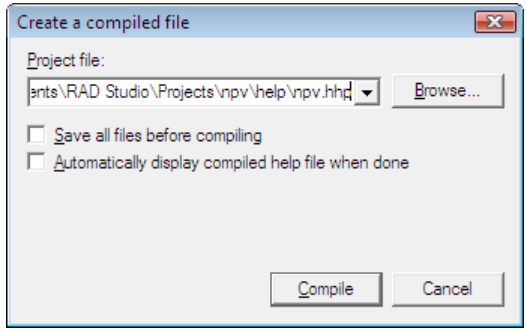


Рис. 8.14. Чтобы создать chm-файл, надо щелкнуть на кнопке **Compile**

В результате в каталоге проекта будет создан chm-файл справочной информации. Чтобы убедиться, что файл справочной информации создан правильно, надо в меню **View** выбрать команду **Compiled Help File**. На экране должно появиться окно справочной информации.

Отображение справочной информации

Отображение справочной информации формата HTML Help осуществляет утилита hh.exe, которая по отношению к приложению является внешней программой.

Запуск внешних программ обеспечивает функция `WinExec`. В качестве параметров функции `WinExec` надо указать команду запуска программы и режим отображения окна, в котором программа должна работать. Команда запуска записывается точно так же, как если бы она набиралась в окне **Выполнить** или в окне командной строки операционной системы. Например, инструкция запуска утилиты hh.exe с целью отображения справочной информации, которая находится в файле npv.chm, выглядит так:

```
WinExec('hh.exe npv.chm', SW_RESTORE);
```

В результате выполнения приведенной команды на экране появится окно справочной информации, в котором будет отображено содержимое первого раздела.

Если необходимо отобразить конкретный раздел справочной информации, то в команде запуска утилиты hh.exe надо указать параметр `-mapid` и идентификатор нужного раздела. Например, в результате выполнения инструкции

```
WinExec('hh.exe -mapid 3 npv.chm', SW_RESTORE);
```

на экране появится окно справочной информации, в котором будет отображена информация раздела **Финансовые затраты** (см. листинг 8.1).

Отображение справочной информации демонстрирует программа "Чистый дисконтированный доход" (ее форма приведена на рис. 8.15). Во время работы с программой пользователь может получить справку, сделав щелчок на кнопке **Справка**. Процедура обработки события `Click` на кнопке **Справка**, а также процедура обработки события `Close` формы приведены в листинге. 8.2.

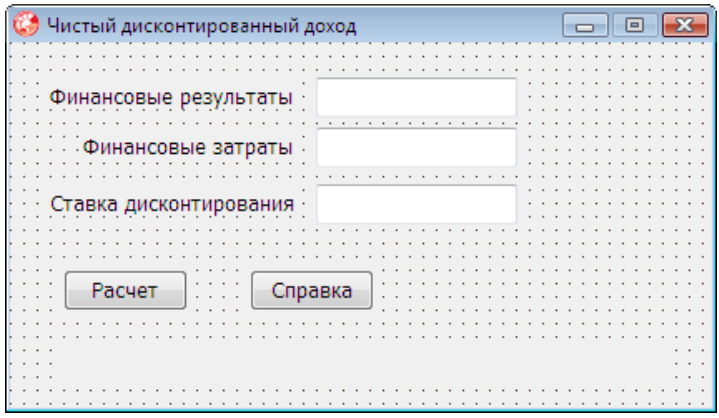


Рис. 8.15. Справочная информация отображается в результате щелчка на кнопке **Справка**

Листинг 8.2. Отображение справочной информации

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Keyboard;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Button1: TButton;
    Button2: TButton;
    Label4: TLabel;
    procedure Button2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// щелчок на кнопке "Расчет"
procedure TForm1.Button1Click(Sender: TObject);
var
  p: real; // поступления от продаж
```

```
  r: real; // расходы
  d: real; // ставка дисконтирования

  npv: real; // чистый дисконтированный доход

  begin

    try
      p := StrToFloat(Edit1.Text);
      r := StrToFloat(Edit2.Text);
      d := StrToFloat(Edit3.Text);

      npv := (p - r) / (1.0 + d);

      Label4.Caption := 'Чистый дисконтированный доход (NPV): ' +
                        FloatToStrF(npv,ffCurrency,6,2);

    except
      on e:Exception do
          MessageDlg('Ошибка исходных данных: ' + e.Message,
                     mtError,[mbOk],0);
    end;
end;

// щелчок на кнопке "Справка"
procedure TForm1.Button2Click(Sender: TObject);
var
    h : HWND; // идентификатор (дескриптор) окна
begin
    h := FindWindow('HH Parent','Чистый дисконтированный доход');
    if h = 0 then
        WinExec('hh.exe npv.chm',SW_RESTORE)
    else
        begin
          ShowWindow(h,SW_RESTORE);
          Windows.SetForegroundWindow(h);
        end;
end;

// щелчок на кнопке "Закрыть окно"
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
var
    h : HWND;
begin
    h := FindWindow('HH Parent','Чистый дисконтированный доход');
    if h <> 0 then
        // открыто окно справочной информации
        SendMessage (h,WM_CLOSE,0,0);
end;
end.
```

Необходимо обратить внимание на следующее. Утилита hh.exe является внешней программой. Поэтому, если не предпринимать никаких усилий, и в процедуре обработки события `Click` на кнопке **Справка** указать только инструкцию запуска утилиты hh.exe, то каждый щелчок на кнопке **Справка** будет запускать новый экземпляр утилиты hh.exe (открывать новое окно справочной информации). Чтобы избежать ситуации, когда одновременно будут открыты несколько окон, в которых отображается одна и та же справочная информация, в процедуру обработки события `Click` добавлен код, запускающий утилиту hh.exe только в том случае, если она еще не запущена. Проверяет, запущена ли утилита hh.exe или нет, функция `FindWindow` (если программа запущена, то существует окно, в котором программа работает). В качестве параметров функции `FindWindow` передается тип окна (в результате запуска утилиты hh.exe создается окно класса `HH Parent`) и текст, который должен быть в заголовке окна (в заголовке окна справочной информации программы "Доход по вкладу" отображается текст **Доход по вкладу**). Если окно справочной информации не найдено (в этом случае значение функции `FindWindow` равно нулю), то функция `WinExec` запускает утилиту hh.exe. Если программа отображения справочной информации уже запущена (окно программы существует), то функция `ShowWindow` делает окно активным, а функция `SetForegroundWindow` перемещает его на передний план.

Процедура обработки события `Close` формы также использует функцию `FindWindow`, чтобы проверить, открыто окно справочной информации или нет. Если окно открыто, то функция `SendMessage` направляет ему сообщение (команду) `WM_CLOSE` (Закрыть). В результате этого окно справочной информации закрывается. После чего программа завершает работу.

Глава 9

Создание установочного диска

Перенести программу, созданную в Delphi, на другой компьютер можно, например, с помощью "флэшки", просто скопировав файлы программы на диск компьютера пользователя. При этом следует понимать, что программа, которая работает на компьютере разработчика, у пользователя может и не запуститься, например, из-за отсутствия нужных ей библиотек или файлов данных.

Задачу установки и настройки приложения можно возложить на пользователя, передав ему все файлы, образующие программу, и инструкцию по установке. Однако лучше следовать принятой практике, когда установку прикладной программы на компьютер пользователя выполняет специальная программа-установщик.

Создать установщик можно точно так же, как и любое другое приложение. Однако лучше воспользоваться одной из специальных утилит, обеспечивающих решение этой задачи, например InstallAware.

Утилита InstallAware

Утилита InstallAware является одним из современных инструментов создания инсталляционных программ. Она позволяет создать программу установки (самораспаковывающийся архив, образ установочного диска), обеспечивающую не только копирование файлов на компьютер пользователя, но и настройку системы. Программа установки, созданная с помощью InstallAware, имеет хорошо знакомый большинству пользователей и интуитивно понятный интерфейс.

В комплект поставки RAD Studio XE утилита InstallAware включена в версии Express. На компьютер программиста она устанавливается путем выбора соответствующей команды в стартовом окне установщика RAD Studio.

Новый проект

Процесс создания установщика в InstallAware рассмотрим на примере — создадим программу установки игры "Сапер".

Перед тем как приступить к непосредственной работе в InstallAware, надо составить список файлов, которые необходимо перенести на компьютер пользователя (табл. 9.1). Следует обратить внимание, что в данном случае предполагается, что весь код программы находится в exe-файле, т. е. программа скомпилирована в режиме "без использования пакетов".

Таблица 9.1. *Файлы, которые нужно установить на компьютер пользователя*

Файл	**Описание**	**Куда устанавливать**
saper.exe	Программа	Program Files\Saper
saper.chm	Файл справочной информации	Program Files\Saper
readme.txt	Краткая информация о программе	Program Files\Saper

Запускается InstallAware обычным образом — выбором в меню **Пуск** команды **InstallAware Install ▸ InstallAware Express**.

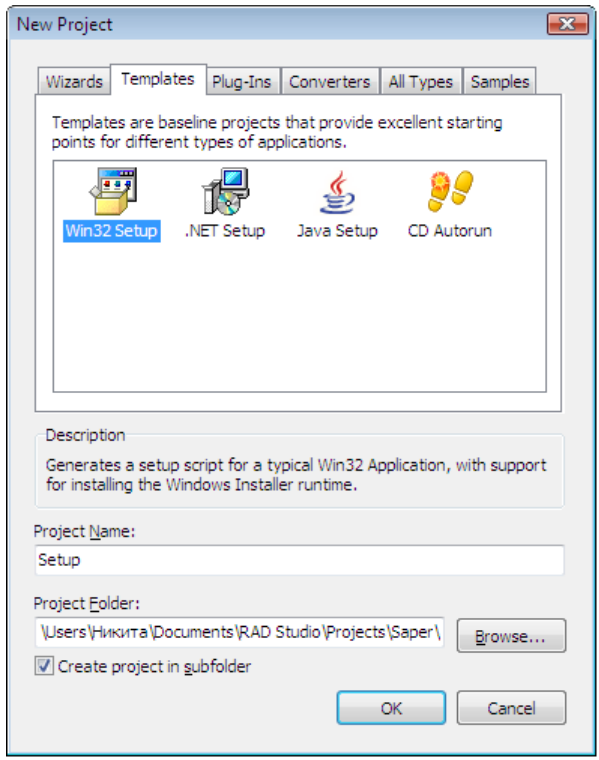


Рис. 9.1. Начало работы над новым проектом

Чтобы начать работу по созданию программы установки, нужно в меню **File** выбрать команду **New ▸ Win32 Setup**, в появившемся окне **New Project** (рис. 9.1)

ввести название проекта (поле **Project Name**) и указать каталог (поле **Project Folder**), в который следует поместить файлы проекта создания программы установки (в рассматриваемом примере каталог проекта будет создан в каталоге приложения, для которого создается программа установки).

Следует обратить внимание на флажок **Create project in subfolder**. Если он установлен (а по умолчанию это так), то в результате щелчка на кнопке **OK** в каталоге, имя которого указано в поле **Project Folder**, будет создана папка проекта, а в ней — файл проекта создания программы установки (с расширением mpr).

Окно утилиты InstallAware в начале работы над новым проектом приведено на рис. 9.2. В левой части окна в виде списка представлены этапы и шаги процесса создания программы установки, в центральной отображаются действия (шаги), относящиеся к выбранному в данный момент этапу.

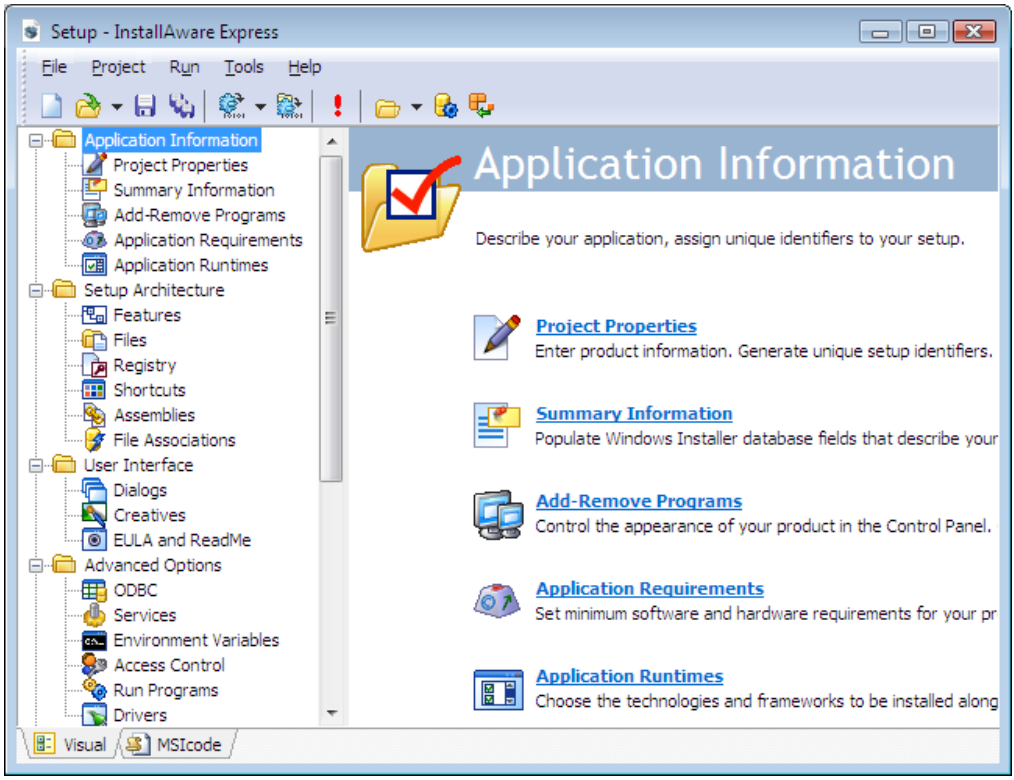


Рис. 9.2. Окно утилиты InstallAware в начале работы над проектом

Чтобы создать программу установки, надо последовательно заполнить формы, которые отображаются в результате выбора названия шага (**Application Information**, **Setup Architecture**, **User Interface**, **Advanced Options**), и активизировать процесс создания программы установки.

Общая информация

В формы, которые появляются в результате выбора соответствующих команд в группе **Application Information**, надо ввести общую информацию об устанавливаемой программе.

Программа и ее разработчик

В форме **Project Properties** (рис. 9.3) в поля **Product Name** и **Product Version** надо ввести соответственно название устанавливаемой программы и номер версии, в поле **Manufacturer** — имя разработчика программы. В поле **Target Folder** надо ввести имя папки компьютера пользователя, в которую надо установить приложение. Так, по умолчанию в этом поле находится шаблон `$PROGRAMFILES$\$TITLE$`, то приложение будет установлено в папку с именем, совпадающим с именем приложения (шаблон `$TITLE$` заменяется текстом, находящимся в поле **Product Name**), которая будет создана в папке Program Files (шаблон `$PROGRAMFILES$`).

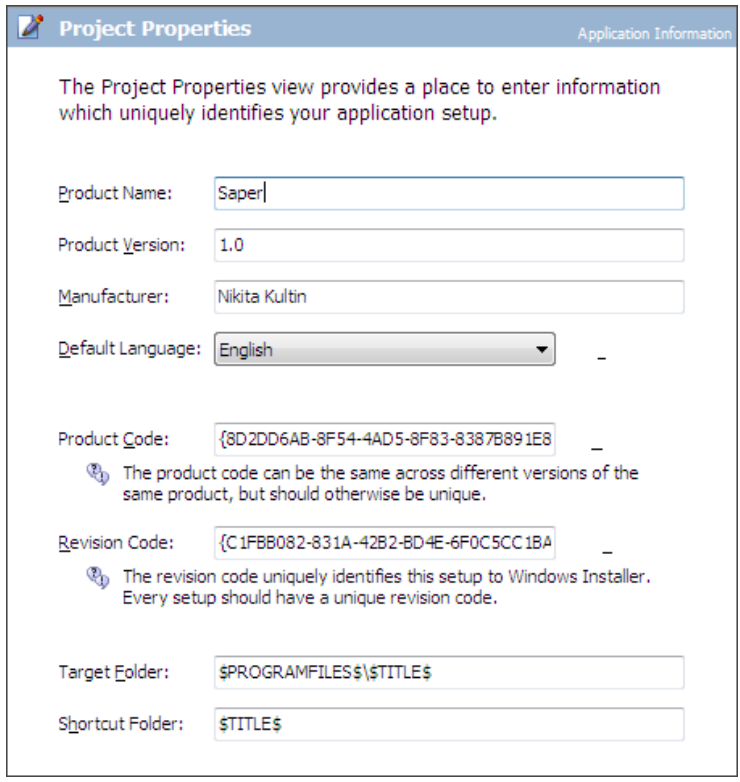


Рис. 9.3. Форма **Project Properties**

В поля следующей формы (рис. 9.4), которая становится доступной в результате выбора команды **Summary Information**, надо ввести информацию, которая будет отображаться в окне программы установки.

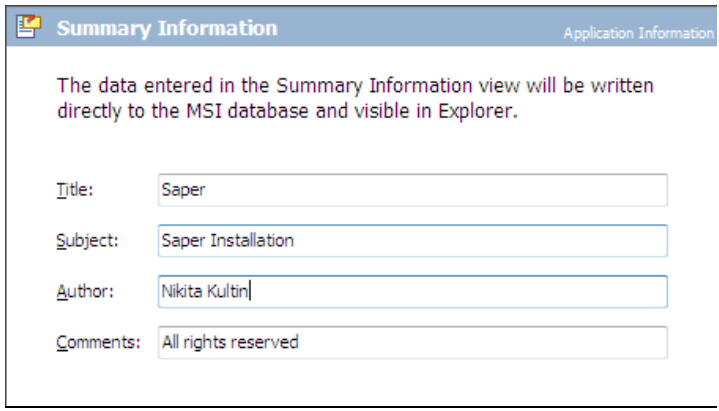


Рис. 9.4. Форма **Summary Information**

В поля формы **Add-Remove Information** надо ввести информацию, которая должна отображаться в окне **Поддержка** (это окно появляется в результате выбора в окне **Установка и удаление программ** операционной системы ссылки **Сведения о поддержке**). В форме надо указать производителя (поставщика) программного продукта, контакт, обеспечивающий поддержку и сопровождение, ссылки на ресурсы поддержки и обновления. Если производитель программы не обеспечивает сопровождение и поддержку программы через Интернет, то соответствующие поля следует оставить незаполненными (удалить значения, указанные по умолчанию).

Требования к системе

Если устанавливаемое приложение предъявляет особые требования к аппаратуре (объем оперативной памяти, разрешение экрана, количество отображаемых цветов) и к программному обеспечению компьютера (версия операционной системы, Internet Explorer, IIS), то их надо указать путем выбора соответствующих значений в форме **Application Requirements**.

Компоненты

В форме **Application Runtimes** надо указать компоненты (приложения и динамические библиотеки), которые необходимы для работы устанавливаемой программы. Например, приложению работы с базой данных требуется соответствующий сервер баз данных.

Архитектура

Команды группы **Setup Architecture** позволяют определить *архитектуру* устанавливаемого приложения: описать его элементы, составить список файлов, которые надо установить на компьютер пользователя, описать ярлыки, которые надо создать.

Возможности

Даже простое приложение можно представить как совокупность нескольких компонентов. Например, программа "Сапер" — это непосредственно программа и справочная информация. *Возможности* пользователя при работе с программой определяются составом установленных компонентов. Например, если на компьютер пользователя установлены файлы справочной системы, то у пользователя есть *возможность* получить доступ к справочной информации, а если файлы справочной информации не установлены, то такой *возможности* нет. Форма **Features** позволяет разделить функциональные *возможности* программы (компоненты, которые могут устанавливаться по отдельности). Разделение компонентов программы на несколько групп (возможностей) по функциональному признаку позволяет организовать многовариантную, в том числе и определяемую пользователем, установку программы.

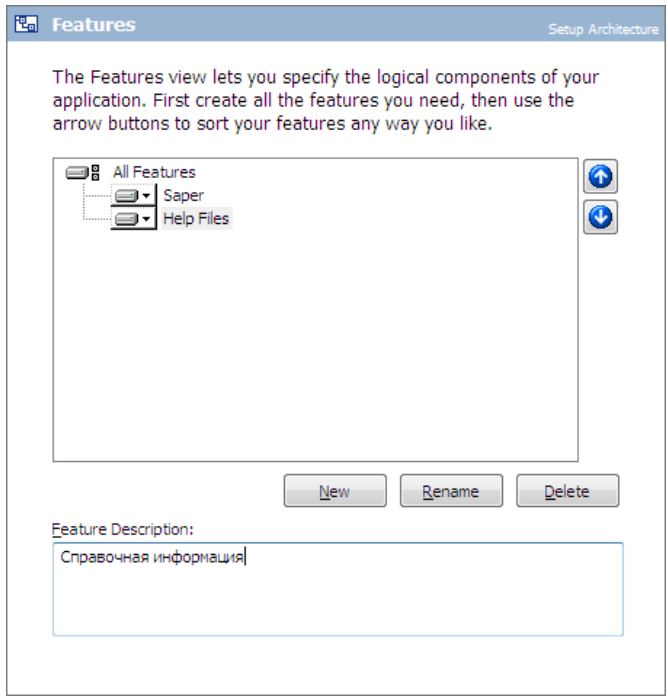


Рис. 9.5. Форма **Features**

По умолчанию определена только одна группа (один элемент в списке **All Features**), что предполагает один вариант установки. Чтобы пользователь мог выбрать устанавливаемые компоненты, надо создать несколько групп и для каждой группы указать компоненты (файлы), реализующие соответствующую функциональность. Чтобы создать группу, надо открыть форму **Features** (рис. 9.5), сделать щелчок на кнопке **New**, ввести название группы и ее описание.

Файлы

Следующее, что надо сделать, — указать файлы, которые необходимо установить на компьютер пользователя. Делается это в форме **Files**. Сначала в левом верхнем списке надо выбрать папку, в которой находятся файлы приложения (в рассматриваемом примере — C:\Users\Никита\Documents\RAD Studio\Projects\Saper). Затем в левом нижнем списке надо указать каталог компьютера пользователя, в который надо поместить файлы приложения. Следует обратить внимание на то, что в этом списке указаны не имена, а псевдонимы каталогов.

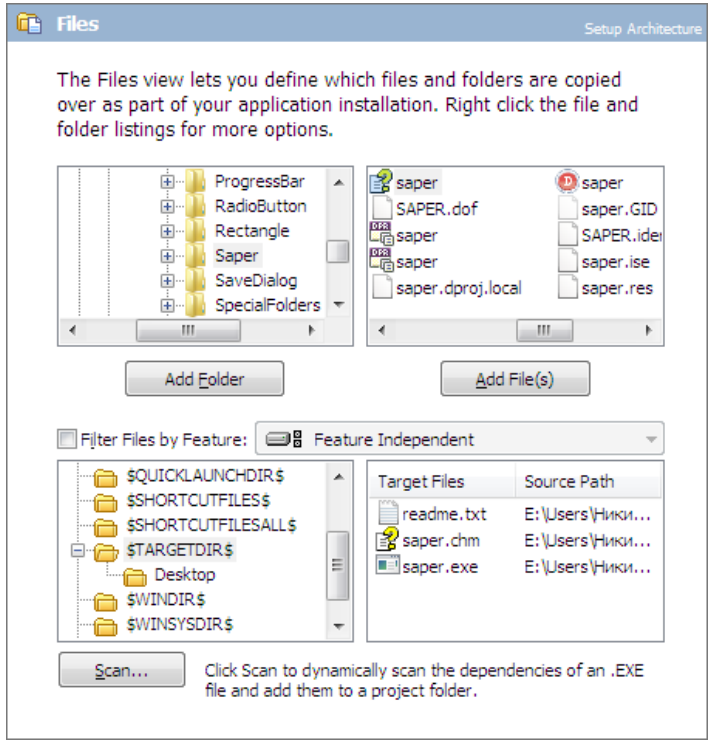


Рис. 9.6. Форма **Files**

Например, псевдоним $PROGRAMFILES$ обозначает каталог Program Files, а псевдоним $TARGETDIR$ — каталог устанавливаемого приложения (каталог, за-

данный в форме **Project Properties**, см. рис. 9.3). После выбора каталога-приемника следует указать файлы, которые надо установить в выбранный каталог. Выбор файлов выполняется в правом верхнем списке. После выбора файла (или группы файлов) надо нажать кнопку **Add File(s)**. Вид формы **Files** после выбора файлов, которые должны быть перенесены на компьютер пользователя в процессе установки программы "Сапер", приведен на рис. 9.6.

Если предполагается, что во время установки пользователь сможет выбрать нужные ему функциональные возможности устанавливаемой программы, то необходимо установить переключатель **Filter Files by Feature**, в ставшем доступном списке выбрать *возможность (см. предыдущий раздел)* и указать файлы, определяющие эту возможность.

Ярлыки

Запустить программу, установленную на компьютере, можно выбором команды в меню программ, которое становится доступным в результате щелчка на кнопке **Пуск**, или щелчком на находящемся на рабочем столе значке (ярлыке) программы. Указать, куда следует поместить ярлык, обеспечивающий запуск программы или выполнение другого действия, например отображение справочной информации, можно в форме **Shortcuts** (рис. 9.7).

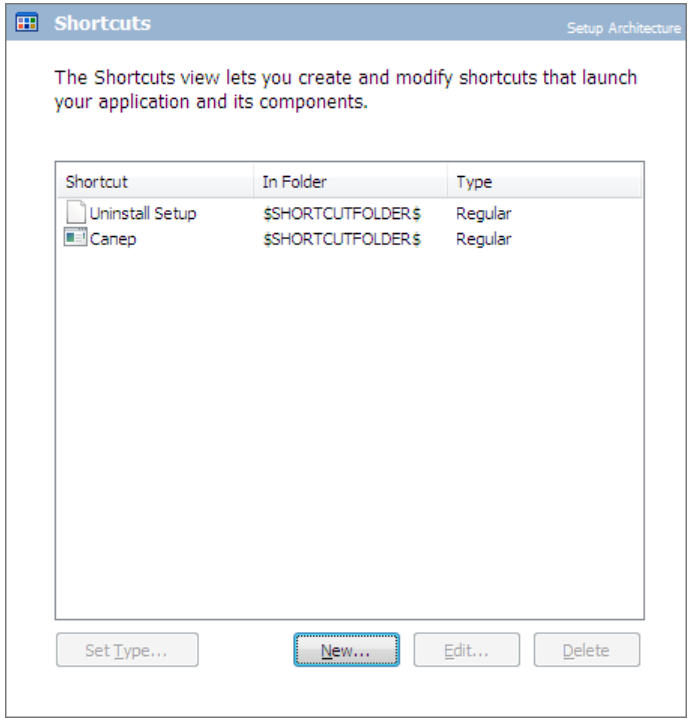


Рис. 9.7. Форма **Shortcuts**

Чтобы создать ярлык, обеспечивающий запуск программы, надо в форме **Shortcuts** сделать щелчок на кнопке **New** и в поля появившегося окна **New Shortcut** (рис. 9.8) ввести информацию, необходимую для создания ярлыка.

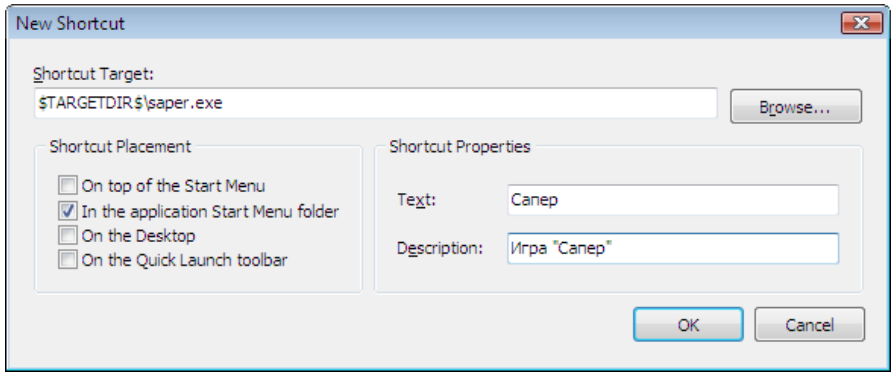


Рис. 9.8. Окно **New Shortcut**

В поле **Shortcuts Target** надо ввести имя файла, который будет открыт в результате щелчка на ярлыке. В группе **Shortcuts Placement** надо указать расположение ярлыка (**In the application Start Menu folder** — в меню программы; **On the Desktop** — на рабочем столе; **On the Quick Launch toolbar** — на панели быстрого запуска).

В поле **Text** надо ввести название ярлыка, а в поле **Description** — текст подсказки (она будет отображаться при позиционировании указателя мыши на ярлыке).

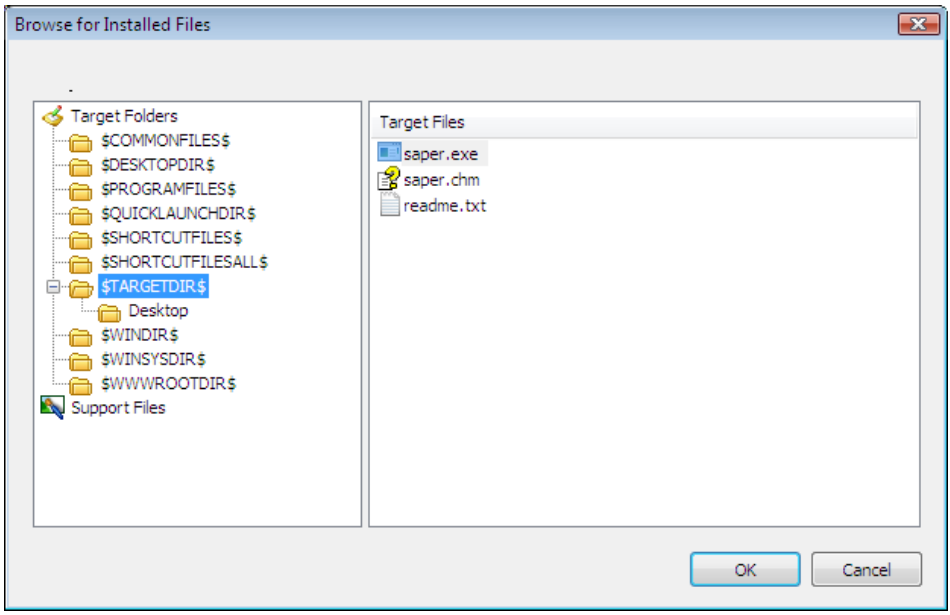


Рис. 9.9. Выбор файла

Чтобы в поле ввести имя файла, нажмите кнопку **Browse** и в появившемся окне (рис. 9.9) выберите файл. Обратите внимание, что в окне отображаются не каталоги компьютера программиста, а каталоги компьютера пользователя.

Интерфейс

В процессе установки программы пользователю, как правило, предоставляется возможность выбрать, куда надо установить программу, а также устанавливаемые компоненты. Взаимодействие программы установки с пользователем обеспечивают соответствующие диалоги.

Команды группы **User Interface** позволяют задать диалоги, которые будут отображаться в процессе подготовки к установке, во время и по окончании установки приложения.

Диалоги

В форме **Dialogs** (рис. 9.10) перечислены диалоги, которые обеспечивают взаимодействие программы установки с пользователем. Назначение часто используемых диалогов приведено в табл. 9.2.

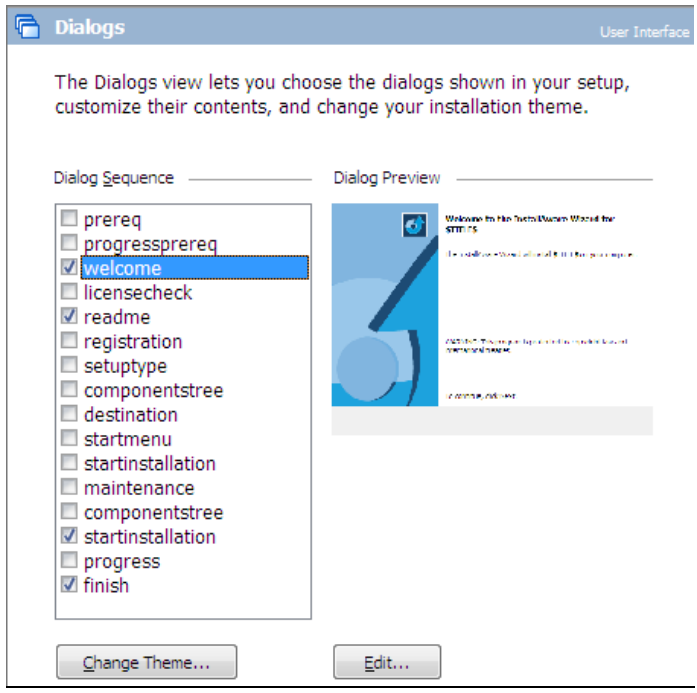


Рис. 9.10. Вкладка **Dialogs**

Таблица 9.2. *Диалоги процесса установки*

Диалог	Назначение
welcome	Вывод информационного сообщения
licensecheck	Отображение лицензионного соглашения. Позволяет прервать процесс установки программы в случае несогласия пользователя с условиями лицензионного соглашения
readme	Отображение информации об устанавливаемой программе
registration	Запрашивает информацию о пользователе (имя, название организации)
setuptype	Предоставляет пользователю возможность выбрать вариант установки (**Typical** — обычная установка, **Minimal** — минимальная установка, **Custom** — выборочная установка)
componentstree	Предоставляет пользователю возможность выбора устанавливаемых компонентов при выборочной (**Custom**) установке
destination	Предоставляет пользователю возможность изменения заданного по умолчанию каталога, в который будет установлено приложение
startmenu	Позволяет пользователю задать имя меню, которое будет создано в списке **Программы** и в которое будет помещен ярлык, обеспечивающий, например, запуск программы
startinstallation	Вывод информации, введенной пользователем на предыдущих шагах, с целью ее проверки перед началом непосредственной установки
progress	Отображает процесс установки (показывает процент выполненной работы по установке программы)
finish	Информирует о завершении установки. Позволяет запустить установленную программу и (или) активизировать отображение ReadMe-файла

Чтобы диалог появился на экране во время работы инсталляционной программы, необходимо отметить соответствующий флажок. В нашем случае (мы создаем программу установки игры "Сапер") надо выбрать диалоги **welcome**, **destination**, **startinstallation**, **progress** и **finish**.

Следует обратить внимание на кнопку **Change Theme**, которая позволяет выбрать стиль оформления диалоговых окон.

Информация о программе и лицензионное соглашение

Форма **EULA and ReadMe** (рис. 9.11) позволяет задать текст лицензионного соглашения (EULA) и информацию о программе (ReadMe), которые будут отобра-

жаться в соответствующих окнах. Лицензионное соглашение и информацию о программе можно набрать непосредственно в области редактирования вкладки или загрузить из файла. В первом случае, после того как текст будет набран, надо сделать щелчок на кнопке **Save**, во втором, для выбора файла, — на кнопке **Load**.

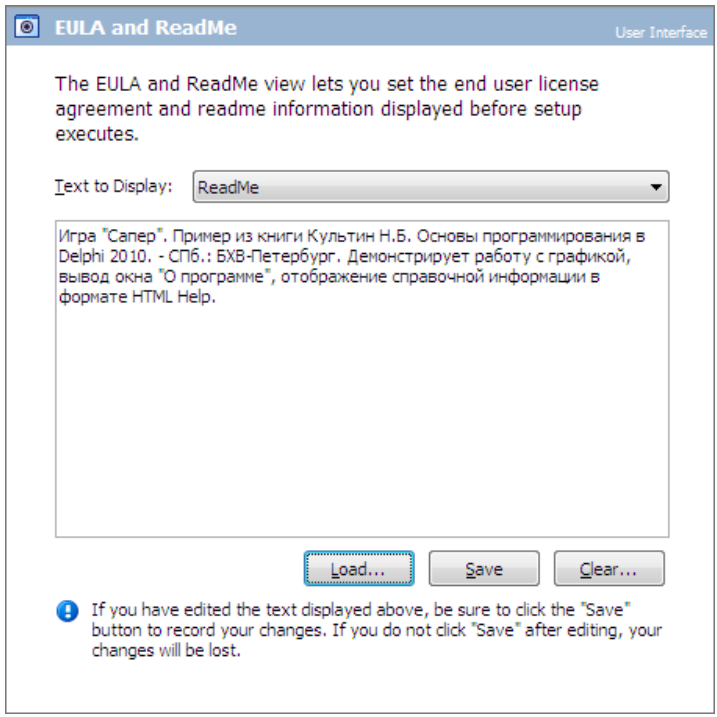


Рис. 9.11. Вкладка **EULA and ReadMe**

Образ установочного диска

По умолчанию InstallAware создает образ установочного диска — совокупность файлов и каталогов. Для экономии места на носителе или, например, для обеспечения удобства передачи (распространения) программы через сеть (локальную или Интернет) вместо обычного образа установочного диска можно создать образ установочного диска в виде самораспаковывающегося архива (Compressed Single Self-Installing EXE).

Тип образа установочного диска (несжатый или самораспаковывающийся архив) можно задать в форме **Build Settings** (рис. 9.12).

Чтобы активизировать процесс образа установочного диска, надо в меню **Project** выбрать команду **Build** или сделать щелчок на кнопке **Build current project** (рис. 9.13).

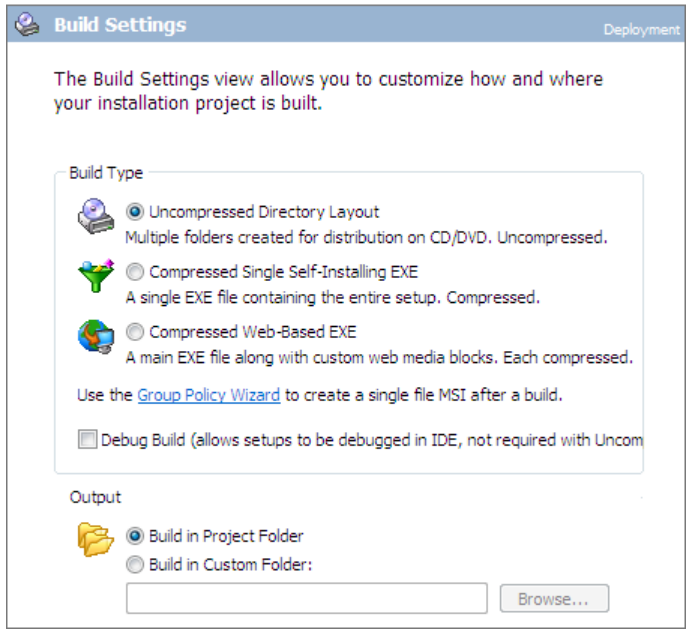


Рис. 9.12. Вкладка **Build Settings**

Рис. 9.13. Кнопка **Build current project**

После того как процесс создания образа установочного диска будет завершен, нужно проверить, как работает программа установки — в меню **Run** выбрать команду **Run** (начнется процесс установки программы и на экране появится окно **Welcome**).

По завершении процесса установки программы следует проверить, создан ли ярлык (в папке меню **Пуск** или на рабочем столе), обеспечивающий запуск приложения, щелчком на ярлыке запустить программу и убедиться, что она работает.

Далее следует проверить, что установленную программу можно удалить с компьютера. Чтобы это сделать, надо через Панель управления открыть окно **Программы и компоненты**, найти установленную программу в списке программ и сделать щелчок на кнопке **Удалить**.

Если программа установки на компьютере разработчика работает правильно (обеспечивает установку и удаление приложения), то необходимо проверить, как она работает на другом компьютере. Для этого надо создать установочный диск — записать содержимое каталога *Проект*\Release\Uncompressed или *Проект*\Release\Single, если был выбран вариант создания упакованного образа установочного диска, на CD, DVD или "флэшку". Затем следует установить программу на компьютер пользователя с диска. Если установка будет выполнена успешно, то работу по созданию программы установки можно считать законченной.

Глава 10

Примеры программ

Экзаменатор

Тестирование широко применяется для оценки уровня знаний в учебных заведениях, при приеме на работу, для оценки квалификации персонала и т. п. Тест — это последовательность вопросов, на которые испытуемый должен ответить, выбрав правильный ответ из нескольких предложенных вариантов. Оценка выставляется в зависимости от количества правильных ответов.

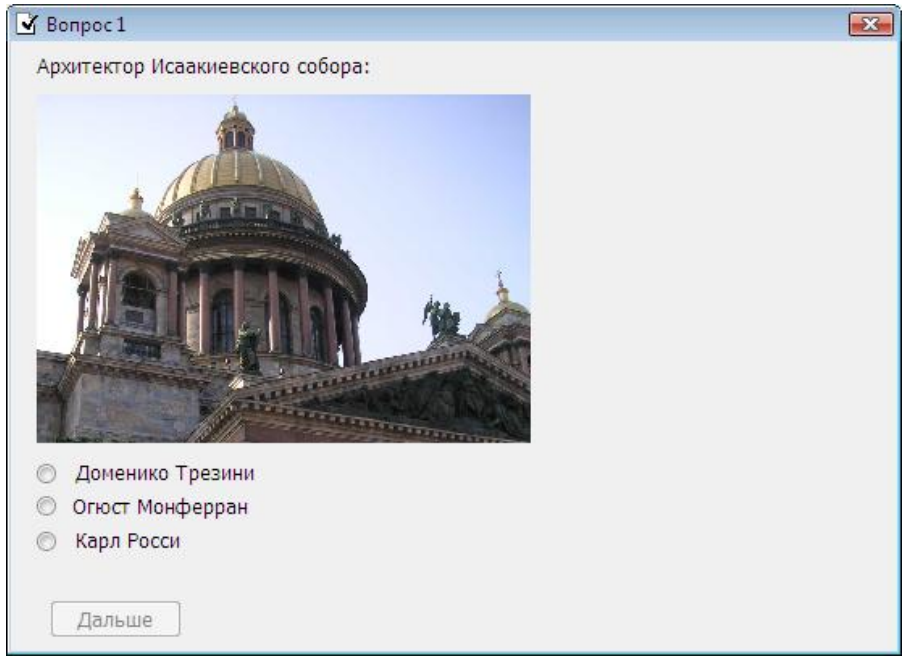


Рис. 10.1. Задача испытуемого — выбрать правильный ответ

Рассмотрим программу "Экзаменатор" (рис. 10.1), которая позволяет автоматизировать процесс тестирования.

Требования к программе

В результате анализа используемых на практике методик тестирования были сформулированы следующие требования к программе:

- программа должна быть универсальной, у пользователя должна быть возможность самостоятельной, без участия программиста, подготовки теста;
- программа должна работать с тестом произвольной длины, т. е. не должно быть ограничения на количество вопросов в тесте;
- ответ на вопрос должен осуществляться путем выбора одного ответа из нескольких (не более четырех) вариантов;
- в программе должна быть заблокирована возможность возврата к предыдущему вопросу. Если вопрос предложен, то на него должен быть дан ответ;
- результат тестирования должен быть отнесен к одному из четырех уровней, например: "отлично", "хорошо", "удовлетворительно" или "плохо";
- вопрос может сопровождаться иллюстрацией.

Файл теста

Универсальность программы тестирования обеспечивается тем, что вопросы находятся в текстовом файле, имя которого указывается в команде запуска программы. Помимо вопросов (здесь и далее: вопрос — это вопрос и несколько вариантов ответа) в файле теста находится информация, необходимая для выставления оценки.

Файл теста состоит из:

- заголовка;
- раздела оценок;
- раздела вопросов.

Заголовок состоит из двух абзацев: первый абзац — название теста (отображается в заголовке окна программы тестирования), второй — вводная информация, например о назначении теста и критерии выставления оценки.

Вот пример заголовка:

```
История Санкт-Петербурга

Сейчас Вам будут предложены вопросы о памятниках и архитектурных сооружениях
Санкт-Петербурга. Вы должны из нескольких предложенных вариантов ответа вы-
брать правильный.
```

За заголовком следует раздел оценок, в котором указывается количество баллов (количество правильных ответов), необходимое для достижения уровня, и сообщение, информирующее испытуемого о достижении уровня. В простейшем случае сообщение — это оценка. Для каждого уровня указывается количество правильных ответов и, в следующей строке, сообщение. Вот пример раздела оценок:

```
100

Отлично

85

Хорошо
```

```
60
Удовлетворительно
50
Плохо
```

За разделом оценок следует раздел вопросов теста.

Каждый вопрос начинается текстом вопроса. В следующей строке указывается количество альтернативных ответов, номер правильного ответа и признак наличия иллюстрации. Если вопрос сопровождается иллюстрацией, то значение признака должно быть равно единице, если нет — нулю. В следующей строке, если значение признака наличия иллюстрации равно единице, указывается имя файла иллюстрации. Далее следуют альтернативные ответы, каждый из которых должен представлять собой абзац текста.

Вот пример вопроса:

```
Архитектор Зимнего дворца
3 1 1
herm.jpg
Бартоломео Растрелли
Карл Росси
Огюст Монферран
```

В приведенном примере к вопросу даны три варианта ответа, правильным является первый ответ (архитектор Зимнего дворца — Бартоломео Растрелли). К вопросу есть иллюстрация (третье число во второй строке — единица), которая находится в файле herm.jpg.

В листинге 10.1 в качестве примера приведен текст файла вопросов для проверки знания истории памятников и архитектурных сооружений Санкт-Петербурга.

Листинг 10.1. Файл теста (spb.txt)

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и архитектурных
сооружениях Санкт-Петербурга. Вы должны из нескольких предложенных вариантов
ответа выбрать правильный.
7
Вы прекрасно знаете историю Санкт-Петербурга!
6
Вы много знаете о Санкт-Петербурге, но на некоторые вопросы ответили неверно.
5
Вы недостаточно хорошо знаете историю Санкт-Петербурга.
4
Вы, вероятно, только начали знакомиться с историей Санкт-Петербурга?
Архитектор Исаакиевского собора:
3 2 1
is.jpg
Доменико Трезини
```

```
Огюст Монферран
Карл Росси
Александровская колонна воздвигнута в 1834 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 1 1
herm.jpg
Бартоломео Растрелли
Карл Росси
Огюст Монферран
Михайловский (Инженерный) замок — жемчужина архитектуры Петербурга, построен
по проекту:
3 3 0
Винченцо Бренна
Старова Ивана Егоровича
Баженова Василия Ивановича
Остров, на котором находится Ботанический сад, основанный императором Петром I,
называется:
3 3 1
bot.jpg
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 памятника Петру I "Медный всадник" выполнена
2 1 0
Фальконе
Клодтом
```

Файл теста можно подготовить в текстовом редакторе, который сохраняет "чистый" (без символов форматирования) текст, например в Блокноте. В момент записи текста на диск вместо стандартного расширения txt следует указать какое-либо другое, например exm (от англ. *Examiner* — экзаменатор). Такое решение позволит настроить операционную систему так, что тест будет запускаться автоматически, в результате двойного щелчка на имени файла теста.

Форма приложения

Форма программы тестирования приведена на рис. 10.2.

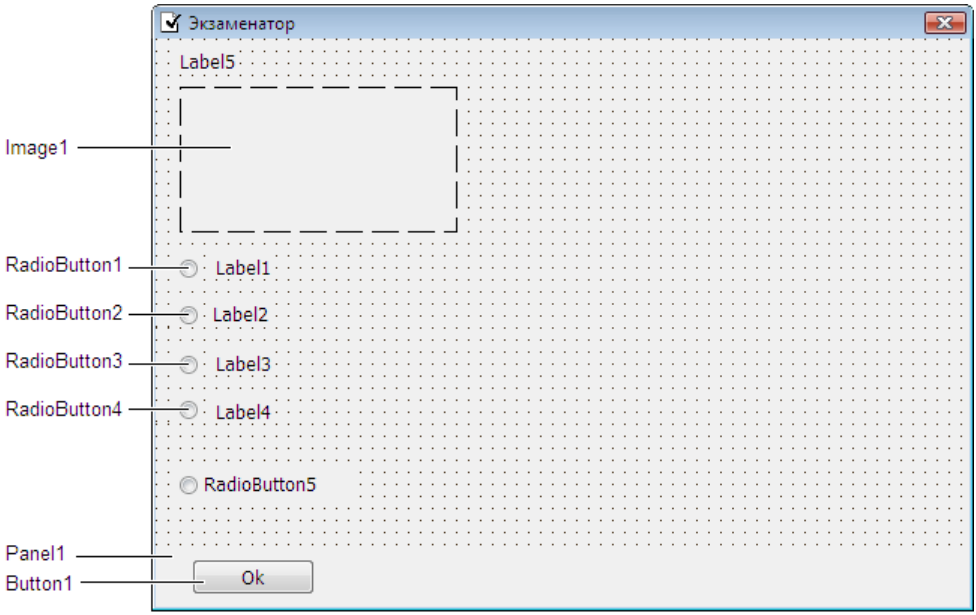


Рис. 10.2. Форма программы "Экзаменатор"

Вопрос отображается в поле `Label5`, варианты ответа — в полях `Label1`—`Label4`. Переключатели `RadioButton1`—`RadioButton4` обеспечивают выбор ответа, а `RadioButton5` (во время работы программы он не отображается) — сброс переключателей выбора ответа. Кнопка `Button1` (она доступна только в том случае, если ответ выбран) обеспечивает переход к следующему вопросу. Следует обратить внимание, что текст на кнопке `Button1` во время работы программы меняется. В начале и в конце работы программы на кнопке находится текст **Ok**, а в процессе тестирования — **Дальше**. Если вопрос сопровождается иллюстрацией, то она отображается в поле компонента `Image1`.

В табл. 10.1 приведены значения свойств формы, в табл. 10.2 — значения свойств компонентов.

***Таблица 10.1.** Значения свойств формы*

Свойство	Значение
`BorderIcons.biMinimize`	`False`
`BorderIcons.biMaximize`	`False`
`BorderStyle`	`bsSingle`

Таблица 10.2. *Значения свойств компонентов*

Компонент	Свойство	Значение
Label1	WordWrap	True
	AutoSize	True
Label2	WordWrap	True
	AutoSize	True
Label3	WordWrap	True
	AutoSize	True
Label4	WordWrap	True
	AutoSize	True
Label5	WordWrap	True
	AutoSize	True
Image1	Proportional	True
Panel1	Align	alBottom
	BevelOuter	bvNone
	Height	41
RadioButton5	Visible	False

Следует обратить внимание, что значения свойств, определяющих положение компонентов, предназначенных для отображения вопроса, альтернативных ответов и иллюстрации, вычисляются во время работы программы, после того как будет прочитан очередной вопрос. Положение компонента `Label1` отсчитывается от нижней границы компонента `Label5` или, если вопрос сопровождается иллюстрацией, от нижней границы компонента `Image1`. Положение компонента `Label2` отсчитывается от нижней границы компонента `Label1`. Аналогичным образом вычисляется положение компонентов `Label3` и `Label4`.

Отображение иллюстрации

Очевидно, что размер области формы, которая может быть использована для отображения иллюстрации, зависит от длины вопроса, а также от количества и длины альтернативных ответов. Чем длиннее вопрос и ответы, чем больше места в поле формы они занимают, тем меньше места остается для отображения иллюстрации.

В рассматриваемой программе размер области отображения иллюстрации (компонента `Image1`) вычисляется после прочтения очередного вопроса, после "загрузки" в компоненты `Label` вопроса и альтернативных ответов, когда становится известным их размер.

Доступ к файлу теста

Для того чтобы программа "Экзаменатор" была действительно универсальной, у пользователя должна быть возможность задать имя файла теста.

Обеспечить возможность настройки программы на работу с конкретным файлом можно несколькими способами. Например, программа может считать имя файла теста из файла конфигурации (ini-файла) или получить его из командной строки.

Командная строка — это строка, которую пользователь должен набрать в окне **Запуск программы**, для того чтобы запустить программу. В простейшем случае командная строка — это имя exe-файла. В командной строке после имени exe-файла можно указать дополнительную информацию, которую надо передать программе, например имя файла, с которым должна работать программа.

Программа может получить параметр, указанный в командной строке, как значение функции `ParamStr(n)`, где `n` — порядковый номер параметра. Количество параметров командной строки находится в глобальной переменной `ParamCount`. Следует обратить внимание, что значение `ParamStr(0)` — это полное имя exe-файла программы.

Далее приведен фрагмент кода, который демонстрирует использование функции `ParamStr`.

```
if ParamCount = 0
     then begin
             ShowMessage('Ошибка! Не задан файл вопросов теста.');
             exit;
     end;
fn := ParamStr(1); // имя файла — параметр команды запуска программы
```

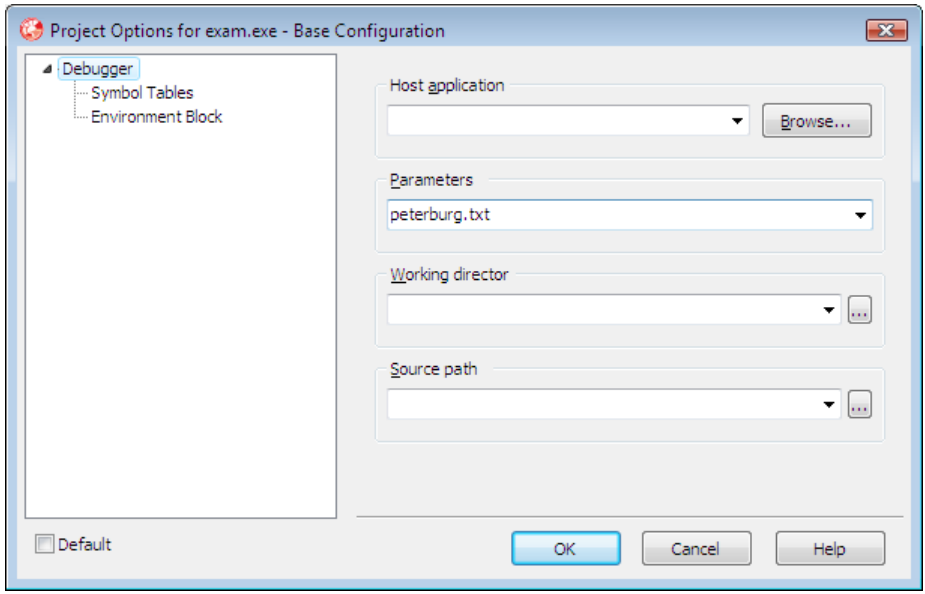


Рис. 10.3. Параметры командной строки надо ввести в поле **Parameters**

Как было сказано, если программе нужны параметры, то они указываются в команде ее запуска после имени exe-файла. При запуске программы из среды разработки, если программе необходимы параметры, их надо указать в поле **Parameters** окна **Project Options** (рис. 10.3), которое становится доступным в результате выбора в меню **Run** команды **Parameters**.

Текст программы

После настройки формы и компонентов можно приступить к набору текста программы (листинг 10.2). Сначала в раздел `private` объявления формы надо поместить объявления вспомогательных процедур и функций, а в раздел `var` секции `implementation` — объявления переменных. После этого следует набрать процедуры и функции, а затем создать процедуры обработки событий. Следует обратить внимание, что событие `Click` на компонентах `RadioButton1`—`RadioButton4` обрабатывает одна процедура.

Листинг 10.2. Экзаменатор

```
{ Универсальная программа тестирования.
  (c) Культин Н.Б., 2006-2010

  Тест загружается из файла, имя которого указано
  в команде запуска программы. }

unit ExamMainForn;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls,

  jpeg; // обеспечивает отображение JPEG-иллюстраций

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
```

```
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    RadioButton5: TRadioButton;
    Image1: TImage;
    Button1: TButton;
    Panel1: TPanel;

    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure RadioButtonClick(Sender: TObject);

  private
    // эти объявления вставлены сюда вручную
    procedure Info;
    function VoprosToScr: boolean;
    procedure ShowPicture; // выводит иллюстрацию
    procedure ResetForm;   // "очистка" формы перед выводом вопроса
    procedure Itog;        // результат тестирования
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

var
  f:TextFile;
  fn:string; // имя файла вопросов

  level:array[1..4] of integer; // сумма, соответствующая уровню
  mes:array[1..4] of string;    // сообщение, соответствующее уровню

  vopr: record
    text: string;    // вопрос
    src: string;     // иллюстрация
    otv: array[1..4] of string; // варианты ответа
```

```
    notv: integer;   // количество вариантов ответа
    right: integer;  // номер правильного ответа
  end;

  cv: integer = 0; // номер вопроса
  otv : integer;   // номер выбранного ответа
  sum: integer;

// информации о тесте
procedure Tform1.Info;
var
   s: string;
begin
   readln(f,s);
   Form1.Caption := s;
   readln(f,s);
   Label5.caption:=s;
end;

// прочитать из файла информацию об оценках
procedure GetLevel;
var
   i:integer;
begin
     for i:=1 to 4 do
     begin
         readln(f,level[i]); // оценка
         readln(f,mes[i]);   // сообщение

     end;
end;

// отображение иллюстрации
procedure TForm1.ShowPicture;
var
  w,h: integer; // максимально возможные размеры картинки
begin
  // вычислить допустимые размеры картинки
  w:=ClientWidth - 10;
  h:=ClientHeight - Panel1.Height - 10
                  - Label5.Top
                  - Label5.Height - 10;
```

```
  // вопросы
  if Label1.Caption <> ''
     then h:=h-Label1.Height-10;
  if Label2.Caption <> ''
     then h:=h-Label2.Height-10;
  if Label3.Caption <> ''
     then h:=h-Label3.Height-10;
  if Label4.Caption <> ''
     then h:=h-Label4.Height-10;

  // если размер картинки меньше w на h, то она не масштабируется
  Image1.Top:=Form1.Label5.Top+Label5.Height+10;
  if Image1.Picture.Height > h
        then Image1.Height:=h
        else Image1.Height:= Image1.Picture.Height;
  if Image1.Picture.Width > w
        then Image1.Width:=w
        else Image1.Width:=Image1.Picture.Width;

  Image1.Visible := True;
end;

// вывести вопрос
function TForm1.VoprosToScr: boolean;
  var
   i: integer;
   p: integer;
  begin

     if EOF(f) then
     begin
       // достигнут конец файла
       VoprosToScr := False;
       exit;
     end;

     readln(f, vopr.text); // прочитать вопрос

     if  vopr.text = '*' then
     begin
       // признак конца теста
```

```
  VoprosToScr := False;
  exit;
end;

cv := cv + 1;
caption:='Вопрос ' + IntToStr(cv);
Label5.caption:= vopr.text;  // вывести вопрос

// прочитать информацию об ответе: количество вариантов,
// номер правильного ответа и признак наличия иллюстрации
readln(f,vopr.notv,vopr.right, p); // p - признак иллюстрации

if p <> 0 then // есть иллюстрация
begin
   readln(f,vopr.src);  // имя файла иллюстрации
   Image1.Tag:=1;
     try
       Image1.Picture.LoadFromFile(vopr.src);
       except
         on E:EFOpenError do
            Image1.Tag:=0;
     end
end
else Image1.Tag := 0; // нет иллюстрации

// читаем варианты ответа
for i:= 1 to vopr.notv do
   readln(f,vopr.otv[i]);

for i:= 1 to vopr.notv do
   case i of
       1: Label1.caption:= vopr.otv[i];
       2: Label2.caption:= vopr.otv[i];
       3: Label3.caption:= vopr.otv[i];
       4: Label4.caption:= vopr.otv[i];
   end;

// здесь прочитана иллюстрация и альтернативные ответы

// текст вопроса уже выведен
if Image1.Tag =1  // есть иллюстрация к вопросу
   then ShowPicture;
```

```
    // вывод альтернативных ответов
    if Label1.Caption <> ''
    then begin
       if Image1.Tag =1
           then Label1.top:=Image1.Top+Image1.Height+10
           else Label1.top:=Label5.Top+Label5.Height+10;
       RadioButton1.top:=Label1.top;
       Label1.visible:=TRUE;
       RadioButton1.Visible:=TRUE;
    end;

    if Label2.Caption <> ''
    then begin
       Label2.top:=Label1.top+ Label1.height+5;
       RadioButton2.top:=Label2.top;
       Label2.visible:=TRUE;
       RadioButton2.Visible:=TRUE;
    end;

    if Label3.Caption <> ''
    then begin
       Label3.top:=Label2.top+ Label2.height+5;
       RadioButton3.top:=Label3.top;
       Label3.visible:=TRUE;
       RadioButton3.Visible:=TRUE;
    end;

    if Label4.Caption <> ''
    then begin
       Label4.top:=Label3.top+ Label3.height+5;
       RadioButton4.top:=Label4.top;
       Label4.visible:=TRUE;
       RadioButton4.Visible:=TRUE;
    end;

    Label6.Caption := '';
    VoprosToScr := True;
  end;

procedure TForm1.ResetForm;
  begin
    // сделать невидимыми все метки и переключатели
```

```
    Label1.Visible:=FALSE;
    Label1.caption:='';
    Label1.width:=ClientWidth-Label1.left-5;
    RadioButton1.Visible:=FALSE;

    Label2.Visible:=FALSE;
    Label2.caption:='';
    Label2.width:=ClientWidth-Label2.left-5;
    RadioButton2.Visible:=FALSE;

    Label3.Visible:=FALSE;
    Label3.caption:='';
    Label3.width:=ClientWidth-Label3.left-5;
    RadioButton3.Visible:=FALSE;

    Label4.Visible:=FALSE;
    Label4.caption:='';
    Label4.width:=ClientWidth-Label4.left-5;
    RadioButton4.Visible:=FALSE;

    Label5.width:=ClientWidth-Label5.left-5;

    Image1.Visible:=FALSE;
end;

// определение достигнутого уровня
procedure TForm1.Itog;
  var
   i:integer;
   buf:string;
  begin
   buf:='';
   buf:='Результаты тестирования' + #13 +
        'Всего вопросов: '+ IntToStr(cv) + #13 +
        'Правильных ответов: ' + IntToStr(sum);
   i:=1;
   while (sum < level[i]) and (i<4) do
         i:=i+1;
   buf:=buf+ #13+ mes[i];
   Label5.Top:= 20;
   Label5.caption:=buf;
  end;
```

```
procedure TForm1.FormActivate(Sender: TObject);
  begin
     if ParamCount = 0
     then begin
             Label5.caption:= 'Не задан файл вопросов теста.';
             Button1.caption:='Ok';
             Button1.tag:=2;
             Button1.Enabled:=TRUE
          end
     else begin
          fn := ParamStr(1);
          assignfile(f,fn);
          try
            reset(f);
          except
            on EFOpenError do
               begin
                    ShowMessage('Файл теста '+fn+' не найден.');
                    Button1.caption:='Ok';
                    Button1.tag:=2;
                    Button1.Enabled:=TRUE;
                    exit;
               end;
          end;

          // ключ /t активизирует режим отладки/обучения
          if ParamStr(2) = '/t' then
              Label6.Visible := True;

          ResetForm;
          Info;       // прочитать и вывести информацию о тесте
          GetLevel;   // прочитать информацию об уровнях оценок
     end;
end;

// щелчок на кнопке Button1
procedure TForm1.Button1Click(Sender: TObject);
  begin
     case Button1.tag of
       0: begin
             Button1.caption:='Дальше';
```

```
                Button1.tag:=1;
                RadioButton5.Checked:=TRUE;
                // вывод первого вопроса
                Button1.Enabled:=False;
                ResetForm;
                VoprosToScr;
            end;
        1: begin // вывод остальных вопросов
                if otv = vopr.right then
                   sum := sum + 1;
                RadioButton5.Checked:=TRUE;
                Button1.Enabled:=False;
                ResetForm;
                if not VoprosToScr then
                      begin
                         closefile(f);
                         Button1.caption:='Ok';
                         Form1.caption:='Результат';
                         Button1.tag:=2;
                         Button1.Enabled:=TRUE;
                         Label6.Visible := False;
                         Itog; // вывести результат
                      end;
            end;
        2: begin // завершение работы
             Form1.Close;
            end;
      end;
end;

// процедура обработки события Click
// для компонентов RadioButton1-RadioButton4
procedure TForm1.RadioButtonClick(Sender: TObject);
begin
  if sender = RadioButton1
    then otv:=1
          else  if sender = RadioButton2
                then otv:=2
                else   if sender = RadioButton3
                             then otv:=3
                             else otv:=4;
```

```
  Button1.enabled:=TRUE;
  Label6.Caption := 'Выбран ответ: ' + IntToStr(otv) +
                ' Правильный ответ: ' + IntToStr(vopr.right);
end;

end.
```

Работает программа "Экзаменатор" так. После запуска программы процедура обработки события `FormActivate` проверяет, указан ли при запуске программы параметр "имя файла теста". Если параметр не указан (в этом случае значение функции `ParamCount` равно нулю), в поле `Label5` выводится сообщение об ошибке, и после щелчка на кнопке **Ok** программа завершает работу. Если параметр указан, то имя файла теста (значение функции `ParamStr(1)`) записывается в переменную `fn` и делается попытка открыть файл. Если файл открыт, вызывается процедура `Info`, которая считывает из файла теста информацию о тесте и выводит ее в поле метки `Label5`. Затем вызывается процедура `GetLevel`, которая считывает из файла теста информацию об уровнях оценки и записывает ее в массивы `level` и `mes`. Следует обратить внимание, что в программе реализован режим проверки файла теста: если в командной строке указан второй параметр и этот параметр — строка `/t`, то свойству `Visible` компонента `Label6` присваивается значение `True`. В результате во время работы программы в поле компонента `Label6` после выбора варианта ответа в нижней части окна отображается номер правильного ответа. После того как испытуемый прочтет информацию о тесте и сделает щелчок на кнопке **Ok**, начинается процесс тестирования.

Следует обратить внимание, что кнопка `Button1` используется для завершения работы программы (если не указан файл теста), активизации процесса тестирования, перехода к следующему вопросу (после выбора варианта ответа) и завершения работы программы. Какое из перечисленных действий будет выполнено в результате щелчка на кнопке `Button1`, определяет значение свойства `Tag` этой кнопки (см. процедуру обработки события `Click`). В начале работы программы значение свойства `Tag` кнопки `Button1` равно нулю. Поэтому в результате первого щелчка на кнопке выполняется та часть программы, которая обеспечивает отображение первого вопроса и записывает в свойство `Tag` единицу. В процессе тестирования значение свойства `Tag` равно единице. Поэтому процедура обработки события `Click` увеличивает на единицу счетчик правильных ответов (если выбран правильный ответ) и вызывает функцию `VoprosToScr`, которая считывает из файла очередной вопрос и выводит его на форму. Если текущий вопрос последний (в этом случае значение функции равно `False`), то вызывается процедура `Itog`, которая выводит результат тестирования, и в свойство `Tag` кнопки `Button1` записывается двойка. В результате, после следующего щелчка на кнопке `Button1`, программа завершает работу.

Отображение вопроса выполняет функция `VoprosToScr`. Сначала она делает попытку прочитать из файла очередной вопрос. Если прочитанная строка — "звез-

дочка", то это значит, что достигнут конец теста (как показал опыт, необходимо явно указать признак конца теста). Если попытка прочитать была успешна, из файла теста считывается информация о количестве вариантов ответа, номер правильного ответа и признак наличия иллюстрации. Далее считывается имя файла иллюстрации (если признак наличия иллюстрации равен единице) и варианты ответа. Следует обратить внимание, что хотя вопросы считываются непосредственно в свойства `Caption` компонентов `Label`, на форме они сразу после прочтения не отображаются (значение свойства `Visible` компонентов `Label1` перед чтением очередного вопроса устанавливается равным `False`). После этого, если к вопросу есть иллюстрация, вызывается процедура `ShowPicture`, которая выводит иллюстрации. Затем выводятся варианты ответа. Необходимо обратить внимание, что положение области отображения первого ответа (компонента `Label1`) отсчитывается от нижней границы области отображения иллюстрации (компонента `Image1`) или от нижней границы области отображения вопроса (компонента `Label5`). Процедура `ShowPicture` выводит иллюстрацию, но сначала на основе информации о размере компонентов `Label1`—`Label5` (значение свойства `AutoSize` равно `True`, поэтому размер компонента определяется текстом, который загружен в свойство `Caption`) она вычисляет размер области, которую можно использовать для отображения иллюстрации.

Процедура `ResetForm` путем выбора невидимого переключателя `RadioButton5` сбрасывает переключатели выбора ответа и делает недоступной кнопку **Дальше** (`Button1`).

Обработку события `Click` на переключателях `RadioButton1`—`RadioButton4` выполняет одна, общая для всех компонентов, процедура `TForm1.RadioButtonClick`. Параметр `Sender` этой процедуры позволяет определить, на каком объекте произошло событие. Процедура фиксирует в переменной `otv` номер переключателя и соответственно номер выбранного ответа. Также она делает доступной кнопку перехода к следующему вопросу.

Процедура `Itog`, сравнивая набранную сумму баллов `sum` со значением элементов массива `level`, определяет, какого уровня достиг испытуемый, и выводит соответствующее сообщение с присвоением значения свойству `Label5.Caption`.

Запуск программы

Программа "Экзаменатор" получает имя файла теста из командной строки. Поэтому чтобы "запустить тест", пользователь должен в окне **Запуск программы** набрать имя программы тестирования и указать файл теста. Это не совсем удобно. Чтобы облегчить жизнь пользователю, можно настроить операционную систему так, что программа тестирования будет запускаться в результате щелчка на значке файла теста. Настройка выполняется следующим образом. Сначала надо раскрыть папку, в которой находится файл теста (exm-файл), и сделать двойной щелчок на значке файла. Так как расширение exm не является стандартным, то система не знает, какую программу надо запустить, чтобы открыть exm-файл. Поэтому она

предложит указать программу, с помощью которой надо открыть файл — на экране появится окно **Выбор программы**. В этом окне нужно сделать щелчок на кнопке **Обзор**, раскрыть папку, в которой находится программа "Экзаменатор", и выбрать exe-файл. После этого надо установить флажок **Использовать выбранную программу для всех файлов такого типа**. Вид окна **Выбор программы** после выполнения всех перечисленных действий приведен на рис. 10.4.

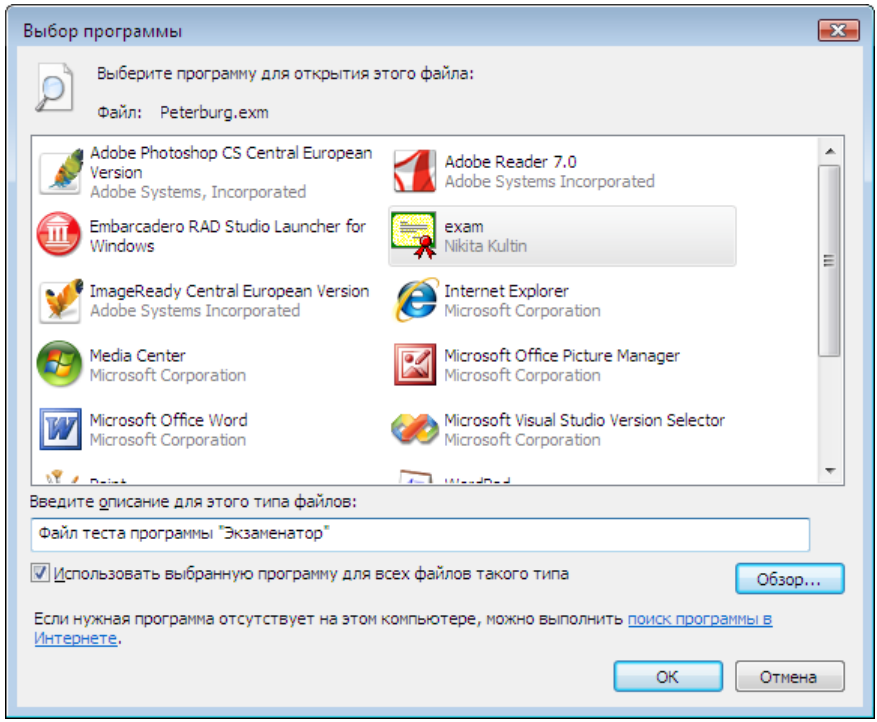


Рис. 10.4. Теперь файлы с расширением exm будет открывать "Экзаменатор"

В результате описанных действий в реестр операционной системы будет внесена информация о том, что файлы с расширением exm надо открывать с помощью программы "Экзаменатор" (имя файла, на котором сделан щелчок, передается программе как параметр). Следует обратить внимание, что задачу настройки операционной системы можно возложить на программу, обеспечивающую ее установку.

Сапер

Всем, кто работает с операционной системой 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. 10.5.

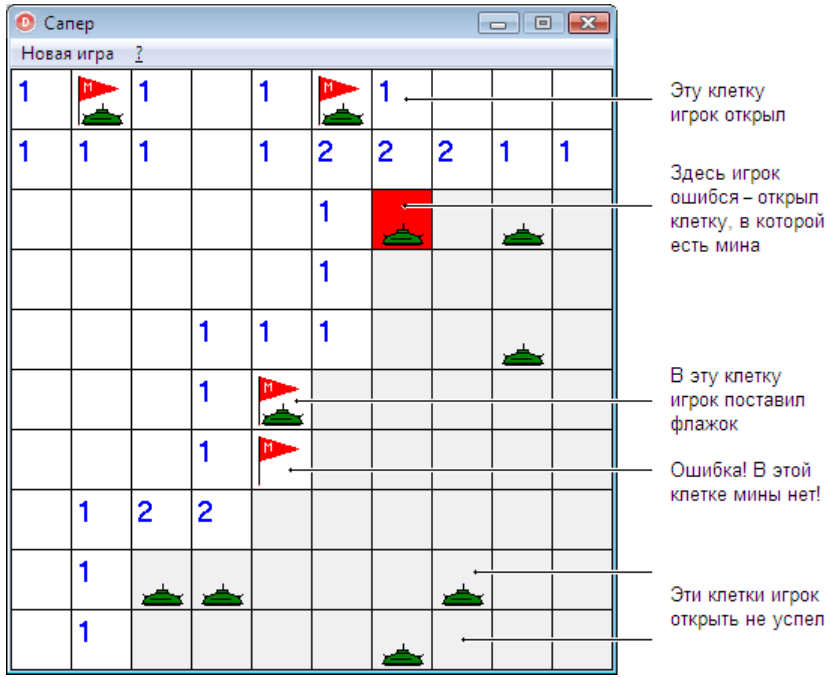


Рис. 10.5. Окно программы "Сапер"

Правила 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. 10.6), первые и последние столбцы и строки соответствуют границе игрового поля.

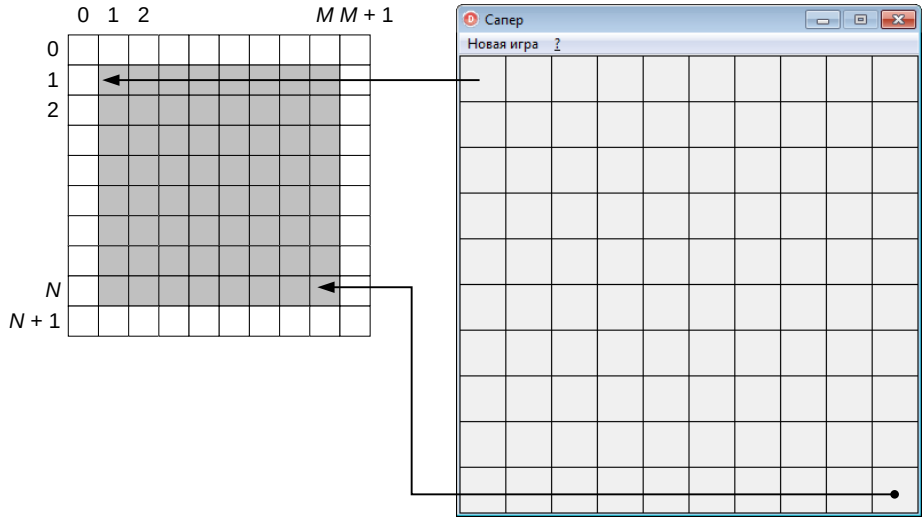


Рис. 10.6. Клетке игрового поля соответствует элемент массива

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 есть, соответствуют числа от 1 до 8. Элементы массива, соответствующие клеткам, в которых находятся мины, имеют значение 9.

Элементы массива, соответствующие границе поля, содержат –3.

В качестве примера на рис. 10.7 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. 10.7. 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 — в клетку, в которой есть мина, то значение соответствующего элемента массива станет 109. Если флажок поставлен ошибочно, например в пустую клетку, элемент массива будет содержать число 100. Если игрок открыл клетку, то значение элемента массива увеличивается на 200. Такой способ кодирования позволяет сохранить информацию об исходном состоянии клетки.

Форма

Главная (стартовая) форма игры "Сапер" приведена на рис. 10.8, значения ее свойств — в табл. 10.3.

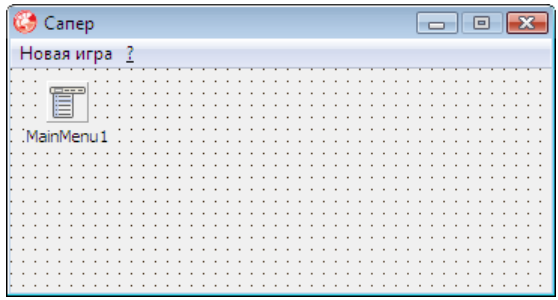


Рис. 10.8. Главная форма программы "Сапер"

Таблица 10.3. *Значения свойств стартовой формы*

Свойство	Значение
`Caption`	`Сапер`
`BorderStyle`	`bsSingle`
`BorderIcons.biMaximize`	`False`
`Position`	`poDesktopCenter`

Следует обратить внимание, что размер формы не соответствует размеру игрового поля. Нужный размер формы будет установлен во время работы программы. Делает это функция обработки события `FormActivate`, которая на основе информации о размере игрового поля (количестве клеток по вертикали и горизонтали) и клеток, устанавливает значение свойств `ClientHeight` и `ClientWidth`, определяющих размер клиентской области главного окна программы.

Главное окно программы содержит только один компонент — `MainMenu1`, который представляет собой главное меню программы. Структура меню (окно конструктора меню) приведена на рис. 10.9.

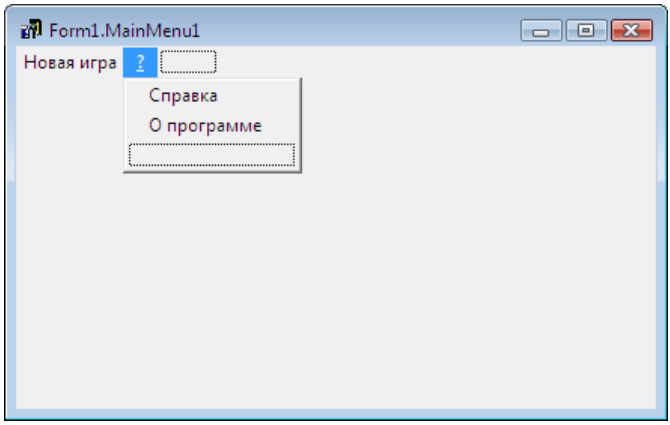


Рис. 10.9. Структура меню программы "Сапер"

После того как будет сформирована структура меню (процесс настройки меню подробно описан в *главе 3*), надо создать процедуры обработки события `Click` для команд **Новая игра**, **Справка** и **О программе**.

Игровое поле

На разных этапах игры игровое поле выглядит по-разному. Сначала поле просто разделено на клетки. Во время игры в результате щелчка правой кнопкой мыши в клетке появляется флажок. Щелчок левой кнопкой тоже меняет вид клетки: клетка меняет цвет и в ней появляется цифра или мина (игра на этом заканчивается). Рассмотрим объекты, свойства и методы, обеспечивающие работу с графикой.

Начало игры

В начале игры программа должна 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`

```
// новая игра – расставить мины и для каждой клетки
// вычислить, сколько мин находится в соседних клетках
procedure NewGame();
```

```
var
     row,col : integer; // координаты клетки
     n : integer;       // количество поставленных мин
     k : integer;       // количество мин в соседних клетках
begin
     // очистим элементы массива, соответствующие клеткам игрового поля
     for row :=1 to MR do
        for col :=1 to MC do
          Pole[row,col] := 0;

     // расставим мины
     Randomize();  // инициализация ГСЧ
     n := 0;       // количество мин
     repeat
       row := Random(MR) + 1;
       col := Random(MC) + 1;
       if (Pole[row,col] <> 9) then
         begin
            Pole[row,col] := 9;
            n := n+1;
         end;
     until (n = NM);

     // для каждой клетки вычислим кол-во мин в соседних клетках
     for row := 1 to MR do
         for col := 1 to MC do
            if (Pole[row,col] <> 9) then
               begin
                 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;
     status := 0;  // начало игры
     nMin   := 0;  // нет обнаруженных мин
     nFlag  := 0;  // нет флагов
end;
```

После того как процедура `NewGame` расставит мины, процедура `ShowPole` (ее текст приведен в листинге 10.4) выводит изображение игрового поля.

Листинг 10.4. Процедура `ShowPole`

```
// показывает поле
procedure TForm1.ShowPole(status : integer);
    var
      row,col : integer;
    begin
       for row := 1 to MR do
         for col := 1 to MC do
              Kletka(row, col, status);
    end;
```

Процедура `ShowPole` выводит изображение поля последовательно, клетка за клеткой. Вывод изображения отдельной клетки выполняет процедура `Kletka`, ее текст приведен в листинге 10.5. Процедура `Kletka` используется для вывода изображения поля в начале игры, во время игры и в ее конце. В начале игры (значение параметра `status` равно нулю) функция выводит только контур клетки, во время игры — количество мин в соседних клетках или флажок, а в конце отображает исходное состояние клетки и действия пользователя. Информацию о фазе игры процедура `Kletka` получает через параметр `status`.

Листинг 10.5. Процедура `Kletka`

```
// рисует клетку
procedure TForm1.Kletka(row, col, status : integer);
var
  x,y : integer; // координаты области вывода
begin
  x := (col-1)* W + 1;
  y := (row-1)* H + 1;

  case status of
       0:  begin
              Canvas.Brush.Color := clBtnFace;
              Canvas.Rectangle(x-1,y-1,x+W,y+H);
              exit;
           end;

        1: begin
             if Pole[row,col] < 100
```

```
        then
          // не открытая — серая
          Canvas.Brush.Color := clBtnFace
        else
          // открытая — белая
          Canvas.Brush.Color := clWhite;

      Canvas.Rectangle(x-1,y-1,x+W,y+H); // нарисовать клетку

      if  (Pole[row,col] >= 101) and (Pole[row,col] <= 108)
      then begin
          Canvas.Font.Size := 13;
          Canvas.Font.Color := clBlue;
          Canvas.TextOut(x+3,y+2,IntToStr(Pole[row,col] -100));
      end;

      if  (Pole[row,col] >= 201) and (Pole[row,col] <= 209)
         then Flag(x,y);
  end;

2: begin
   // игра завершена
   if Pole[row,col] < 100 then
      Canvas.Brush.Color := clBtnFace
   else
      Canvas.Brush.Color := clWhite;

   Canvas.Rectangle(x-1,y-1,x+W,y+H);

   if (Pole[row,col] >= 101) and (Pole[row,col] <= 108) then
   begin
      Canvas.Font.Size := 13;
      Canvas.Font.Color := clBlue;
      Canvas.TextOut(x+3,y+2,IntToStr(Pole[row,col] -100));
      exit;
   end;

   // возможно, в клетке мина, флаг или мина, помеченная флагом

   // в клетке мина, но она была открыта
   if Pole[row,col] = 9 then
       Mina(x, y);
```

```
            // мина + флаг
            if (Pole[row,col] >= 200) then
               Flag(x, y);

            if (Pole[row,col] > 200) then
               Mina(x, y);

            if Pole[row,col] = 109 then
            begin
               Canvas.Brush.Color := clRed;
               Canvas.Rectangle(x-1,y-1,x+W,y+H);
               Mina(x,y);
            end;
         end;
    end;
end;
```

Игра

Во время игры программа воспринимает нажатия кнопок мыши и в соответствии с правилами игры открывает клетки или ставит в клетки флажки.

Основную работу выполняет процедура обработки события `MouseDown` (ее 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, в которой игрок нажал кнопку мыши, в координаты клетки игрового поля. Затем делает необходимые изменения в массиве `Pole` и, если нажата правая кнопка, вызывает функцию `Flag`, которая рисует в клетке флажок. Если нажата левая кнопка в клетке, в которой нет мины, то эта клетка открывается, на экран появляется ее содержимое. Если нажата левая кнопка в клетке, в которой есть мина, то вызывается процедура `ShowPole`, которая показывает все мины, в том числе и те, которые игрок не успел найти.

Листинг 10.6. Обработка события `MouseDown` на поверхности игрового поля

```
// нажатие кнопки мыши на игровом поле
procedure TForm1.Form1MouseDown(Sender: TObject; Button: TMouseButton;
                                Shift: TShiftState; X, Y: Integer);
var
  row, col : integer;
begin
  if status = 2 then // игра завершена
     exit;
```

```
if status = 0 then // первый щелчок
   status := 1;

// преобразуем координаты мыши в индексы клетки поля
row := Trunc(y/H) + 1;
col := Trunc(x/W) + 1;

if Button = mbLeft then
   begin
     if Pole[row,col] = 9 then
        begin // открыта клетка, в которой есть мина
          Pole[row,col] := Pole[row,col] + 100;
          status := 2;  // игра закончена
          ShowPole(status);
        end
      else if Pole[row,col] < 9 then
             Open(row,col);
   end
else
   if Button = mbRight then
     if Pole[row,col] > 200 then
        begin
          // уберем флаг и закроем клетку
          nFlag := nFlag - 1;
          Pole[row,col] := Pole[row,col] - 200;
          x := (col-1)* W + 1;
          y := (row-1)* H + 1;
          Canvas.Brush.Color := clLtGray;
          Canvas.Rectangle(x-1,y-1,x+W,y+H);
        end
      else
        begin // поставить в клетку флаг
          nFlag := nFlag + 1;
          if  Pole[row,col] = 9
            then nMin := nMin + 1;
          Pole[row,col] := Pole[row,col]+ 200;
          if (nMin = NM) and (nFlag = NM) then
            begin
              status := 2;  // игра закончена
```

```
                ShowPole(status);
              end
            else Kletka(row, col, status);
          end;
end;
```

Процедура `Flag` (листинг 10.7) рисует флажок. Флажок (рис. 10.10) состоит из четырех примитивов: линии (древко), замкнутого контура (флаг) и ломаной линии (буква "М"). Процедура рисует флажок, используя метод базовой точки, т. е. координаты всех точек, определяющих положение элементов рисунка, отсчитываются от базовой точки.

Процедура `Mina` (листинг 10.8) рисует мину. Мина (рис. 10.11) состоит из восьми примитивов: два прямоугольника и сектор образуют корпус мины, остальные элементы рисунка — линии ("усы" и полоски на корпусе).

Обеим процедурам в качестве параметров передаются координаты базовой точки рисунка и указатель на объект, на поверхности которого надо рисовать.

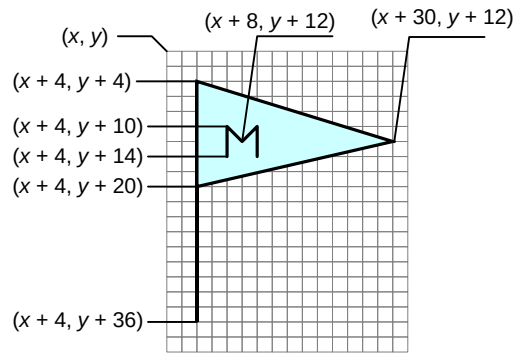


Рис. 10.10. Флажок

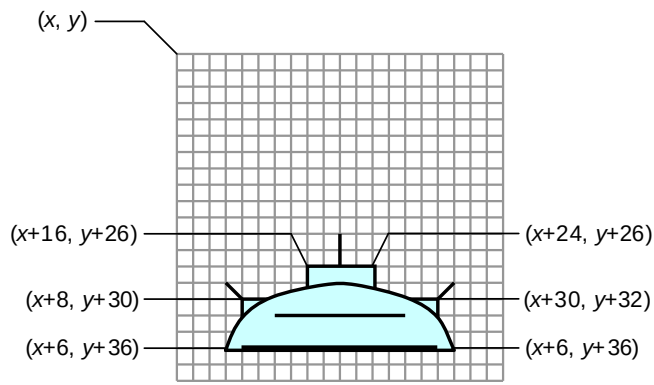


Рис. 10.11. Мина

Листинг 10.7. Процедура `Flag`

```
// рисует флаг
procedure TForm1.Flag(x, y : integer);
    var
       p : array [0..3] of TPoint;   // координаты точек флажка
       m : array [0..4] of TPoint;   // буква М
    begin
        // зададим координаты точек флажка
        p[0].x:=x+4;   p[0].y:=y+4;
        p[1].x:=x+30;  p[1].y:=y+12;
        p[2].x:=x+4;   p[2].y:=y+20;
        p[3].x:=x+4;   p[3].y:=y+36;  // нижняя точка древка

        m[0].x:=x+8; m[0].y:=y+14;
        m[1].x:=x+8; m[1].y:=y+8;
        m[2].x:=x+10; m[2].y:=y+10;
        m[3].x:=x+12; m[3].y:=y+8;
        m[4].x:=x+12; m[4].y:=y+14;

        with Canvas do
          begin
            // установим цвет кисти и карандаша
            Brush.Color := clRed;
            Pen.Color := clRed;

            Polygon(p);  // флажок

            // древко
            Pen.Color := clBlack;
            MoveTo(p[0].x, p[0].y);
           LineTo(p[3].x, p[3].y);

            // буква М
            Pen.Color := clWhite;
            Polyline(m);

            Pen.Color := clBlack;
          end;
      end;
```

Листинг 10.8. Процедура `Mina`

```
// рисует мину
procedure TForm1.Mina(x, y : integer);
begin
    with Canvas do
        begin
           Brush.Color := clGreen;
           Pen.Color := clBlack;
           Rectangle(x+16,y+26,x+24,y+30);
           Rectangle(x+8,y+30,x+16,y+34);
           Rectangle(x+24,y+30,x+32,y+34);
           Pie(x+6,y+28,x+34,y+44,x+34,y+36,x+6,y+36);

           MoveTo(x+12,y+32); LineTo(x+26,y+32);
           MoveTo(x+8,y+36);  LineTo(x+32,y+36);
           MoveTo(x+20,y+22); LineTo(x+20,y+26);
           MoveTo(x+8, y+30); LineTo(x+6,y+28);
           MoveTo(x+32,y+30); LineTo(x+34,y+28);
         end;
end;
```

Справочная информация

В результате выбора в меню **?** команды **Справка** появляется окно справочной информации (рис. 10.12). Активизирует процесс отображения справочной информации процедура обработки события `Click` (листинг 10.9) на элементе меню `N3`. Следует обратить внимание, что перед тем как вызвать процедуру `WinExec`, которая запускает утилиту отображения справочной информации, процедура проверяет, не отображается ли уже справочная информация. Если окно справочной информации открыто, то вызывается функция `SetForegroundWindow`, которая перемещает окно справочной информации на передний план.

Листинг 10.9. Отображение справочной информации

```
// выбор из меню ? команды "Справка"
procedure TForm1.N3Click(Sender: TObject);
var
    h : HWND;
begin
    h := FindWindow('HH Parent','Сапер');
    if h = 0 then
        WinExec('hh.exe saper.chm', SW_RESTORE)
```

```
    else
        begin
          ShowWindow(h,SW_RESTORE);
          Windows.SetForegroundWindow(h);
        end;
end;
```

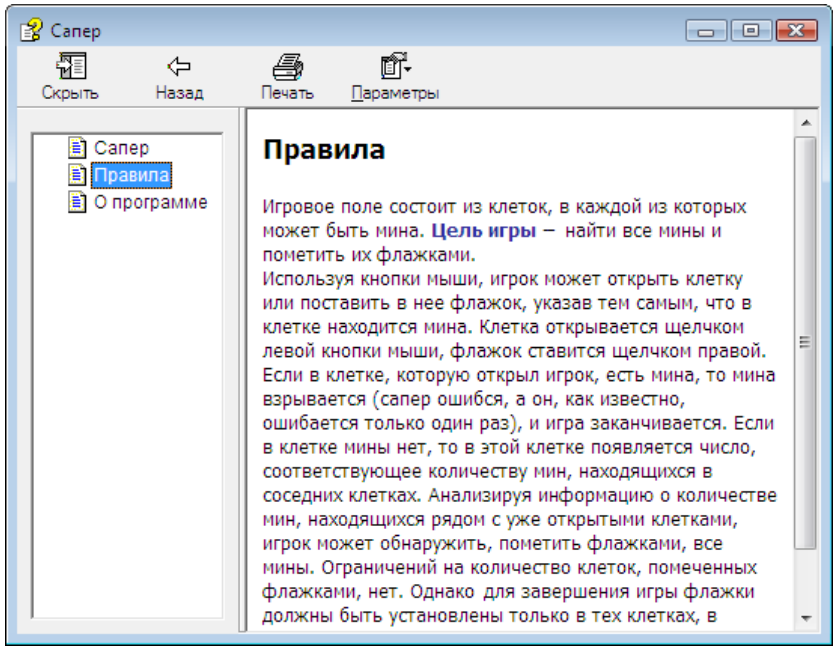


Рис. 10.12. Окно справочной системы программы "Сапер"

Информация о программе

При выборе из меню **?** команды **О программе** на экране появляется одноименное окно (рис. 10.13).

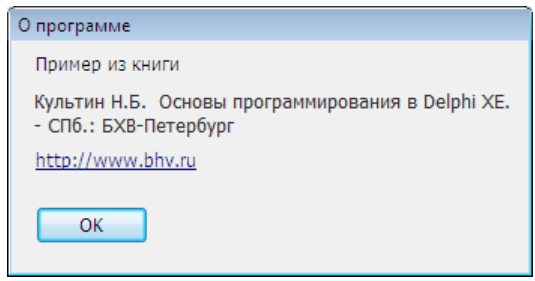


Рис. 10.13. Выбрав ссылку, можно перейти на страницу издательства "БХВ-Петербург"

Чтобы программа во время своей работы могла вывести на экран окно, отличное от главного (стартового), в проект нужно добавить форму. Делается это выбором в меню **File** команды **New ▸ Form - Delphi for Win32**. В результате выполнения этой команды в проект добавляются новая форма и соответствующий ей модуль.

Форма **О программе** (`Form2`) приведена на рис. 10.14, значения свойств формы и компонентов — в табл. 10.4 и 10.5.

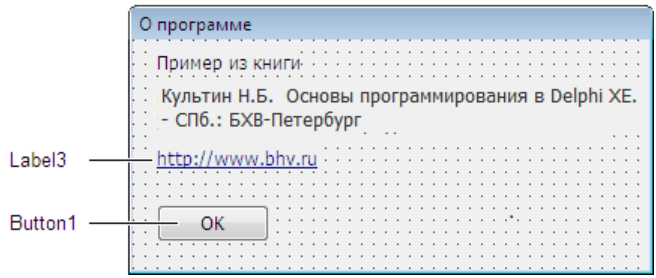


Рис. 10.14. Форма **О программе**

Таблица 10.4. *Значения свойств формы* ***О программе***

Свойство	**Значение**
`Name`	`Form2`
`Caption`	`О программе`
`BorderStyle`	`bsSingle`
`BorderIcons.biSystemMenu`	`False`
`Position`	`poMainFormCenter`

Таблица 10.5. *Значения свойств компонентов формы* ***О программе***

Компонент	**Свойство**	**Значение**

`Label3`	`Font.Color`
	`Font.Style.Underline`
	`Cursor`
		`clNavy`
		`True`
		`crHandPoint`

`Button1`	`ModalResult`	`mrOk`

Вывод окна **О программе** выполняет функция обработки события `Click`, которое происходит в результате выбора в меню **?** команды **О программе** (лис-

тинг 10.10). Следует обратить внимание, что в секцию `implementation` главного модуля необходимо поместить ссылку на модуль формы **О программе** — директиву `uses AboutForm`. Непосредственно отображение окна выполняет метод `ShowModal`, который выводит окно, как *модальный* диалог. Модальный диалог перехватывает все события, адресованные другим окнам приложения, в том числе и главному. Таким образом, пока модальный диалог находится на экране, продолжить работу с приложением, которое вывело модальный диалог, нельзя.

Листинг 10.10. Отображение окна *О программе*

```
procedure TForm1.N4Click(Sender: TObject);
begin
  Form2.ShowModal;
end;
```

На поверхности формы **О программе** есть ссылка на Web-страницу издательства "БХВ-Петербург". Предполагается, что в результате щелчка на Web-ссылке в окне браузера будет открыта указанная страница. Запуск браузера обеспечивает функция `ShellExecute` (листинг 10.11). Эта функция достаточно универсальна. Она обеспечивает выполнение операций с файлами, тип которых известен операционной системе. В данном случае необходимо открыть Web-страницу, поэтому в качестве параметров функции указаны команда `open` и адрес страницы. Следует обратить внимание, что функция `ShellExecute` не запускает конкретную программу, а информирует операционную систему о необходимости открыть указанный файл. Поэтому в результате щелчка на ссылке будет запущен браузер, установленный на компьютере пользователя.

Листинг 10.11. Щелчок на Web-ссылке

```
procedure TForm2.Label3Click(Sender: TObject);
begin
  { Чтобы функция ShellExecute была доступна, в директиву
    uses надо добавить ShellAPI. }

  ShellExecute(Application.Handle,
               'open', PChar(Label3.Caption), '', '', SW_RESTORE);
end;
```

Окно **О программе** закрывается в результате щелчка на кнопке **OK**. Здесь необходимо обратить внимание, что событие `Click` на кнопке **OK** не обрабатывается. Окно закрывается, т. к. значение свойства `ModalResult` кнопки `Button1` равно `mrOK` (по умолчанию значение этого свойства равно `mrNone`).

Текст программы

Полный текст программы "Сапер" приведен в листингах 10.12 (модуль главной формы) и 10.13 (модуль формы **О программе**).

Листинг 10.12. Модуль главной формы (MainForm.pas)

```
unit MainForm;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls, OleCtrls;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;

    procedure Form1Create(Sender: TObject);
    procedure Form1Paint(Sender: TObject);
    procedure Form1MouseDown(Sender: TObject; Button: TMouseButton;
                             Shift: TShiftState; X, Y: Integer);
    procedure N1Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);

  private
    Procedure Kletka(row, col, status : integer); // рисует клетку
    Procedure ShowPole(status : integer);
    Procedure Mina(x, y : integer);     // рисует мину
    Procedure Flag(x, y : integer);     // рисует флаг
    Procedure Open(row, col : integer); // открывает текущую и соседние
                                        // клетки, в которых нет мин
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

uses AboutForm; // ссылка на модуль формы "О программе"

{$R *.DFM}

const
   MR = 10; // кол-во клеток по вертикали
   MC = 10; // кол-во клеток по горизонтали
   NM = 10; // кол-во мин

   W    = 40;  // ширина клетки поля
   H    = 40;  // высота клетки поля

var
   Pole: array[0..MR+1, 0.. MC+1] of integer; // минное поле
   // значение элемента массива:
   // 0..8 — количество мин в соседних клетках
   // 9 — в клетке мина
   // 100..109 — клетка открыта
   // 200..209 — в клетку поставлен флаг

   nMin  : integer;  // кол-во найденных мин
   nFlag : integer;  // кол-во поставленных флагов

   status : integer; // 0 — начало игры; 1 — игра; 2 — результат

procedure NewGame(); forward; // генерирует новое поле

// рисует клетку
procedure TForm1.Kletka(row, col, status : integer);
   var
      x,y : integer; // координаты области вывода
   begin
      x := (col-1)* W + 1;
      y := (row-1)* H + 1;

      case status of
        0: begin
```

```
      Canvas.Brush.Color := clBtnFace;
      Canvas.Rectangle(x-1,y-1,x+W,y+H);
      exit;
   end;
1: begin
      if Pole[row,col] < 100
        then
          // не открытая — серая
          Canvas.Brush.Color := clBtnFace
        else
          // открытая — белая
          Canvas.Brush.Color := clWhite;

      Canvas.Rectangle(x-1,y-1,x+W,y+H); // нарисовать клетку

      if (Pole[row,col] >= 101) and
         (Pole[row,col] <= 108) then
      begin
            Canvas.Font.Size := 13;
            Canvas.Font.Color := clBlue;
            Canvas.TextOut(x+3,y+2,IntToStr(Pole[row,col]-100));
      end;

      if (Pole[row,col] >= 201) and (Pole[row,col] <= 209)
          then Flag(x,y);
   end;

2: begin
      // игра завершена
      if Pole[row,col] < 100 then
         Canvas.Brush.Color := clBtnFace
      else
         Canvas.Brush.Color := clWhite;

      Canvas.Rectangle(x-1,y-1,x+W,y+H);

      if (Pole[row,col] >= 101) and (Pole[row,col] <= 108) then
      begin
         Canvas.Font.Size := 13;
         Canvas.Font.Color := clBlue;
         Canvas.TextOut(x+3,y+2,IntToStr(Pole[row,col]-100));
         exit;
      end;
```

```
            // возможно, в клетке мина, флаг или мина,
            // помеченная флагом мина
            if Pole[row,col] = 9 then
                Mina(x, y);

            // мина + флаг
            if (Pole[row,col] >= 200) then
               Flag(x, y);

            if (Pole[row,col] > 200) then
               Mina(x, y);

            if Pole[row,col] = 109 then
            begin
               Canvas.Brush.Color := clRed;
               Canvas.Rectangle(x-1,y-1,x+W,y+H);
               Mina(x,y);
            end;
         end;
      end;
 end;

// рисует поле
procedure TForm1.ShowPole(status : integer);
    var
      row,col : integer;
    begin
       for row := 1 to MR do
         for col := 1 to MC do
              Kletka(row, col, status);
    end;

// рекурсивная функция открывает текущую и все соседние
// клетки, в которых нет мин
procedure TForm1.Open(row, col : integer);
begin
     if Pole[row,col] = 0 then
       begin
         Pole[row,col] := 100;
         Kletka(row,col, 1);
         Open(row,col-1);
```

```
        Open(row-1,col);
        Open(row,col+1);
        Open(row+1,col);

        // примыкающие диагонально
        Open(row-1,col-1);
        Open(row-1,col+1);
        Open(row+1,col-1);
        Open(row+1,col+1);
      end
    else
      if (Pole[row,col] < 100) and (Pole[row,col] <> -3) then
         begin
             Pole[row,col] := Pole[row,col] + 100;
             Kletka(row, col, 1);
         end;
end;

// новая игра — расставить мины и для каждой клетки
// вычислить, сколько мин находится в соседних клетках
procedure NewGame();
var
    row,col : integer; // координаты клетки
    n : integer;       // количество поставленных мин
    k : integer;       // количество мин в соседних клетках
begin
    // очистим элементы массива, соответствующие клеткам игрового поля
    for row :=1 to MR do
       for col :=1 to MC do
             Pole[row,col] := 0;

    // расставим мины
    Randomize();  // инициализация ГСЧ
    n := 0;       // количество мин
    repeat
      row := Random(MR) + 1;
      col := Random(MC) + 1;
      if (Pole[row,col] <> 9) then
        begin
           Pole[row,col] := 9;
```

```
            n := n+1;
        end;
    until (n = NM);

    // для каждой клетки вычислим кол-во мин в соседних клетках
    for row := 1 to MR do
        for col := 1 to MC do
          if (Pole[row,col] <> 9) then
        begin
           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;

    status := 0;  // начало игры
    nMin   := 0;  // нет обнаруженных мин
    nFlag  := 0;  // нет флагов
end;

// рисует мину
procedure TForm1.Mina(x, y : integer);
begin
    with Canvas do
        begin
           Brush.Color := clGreen;
           Pen.Color := clBlack;
             Rectangle(x+16,y+26,x+24,y+30);
             Rectangle(x+8,y+30,x+16,y+34);
             Rectangle(x+24,y+30,x+32,y+34);
             Pie(x+6,y+28,x+34,y+44,x+34,y+36,x+6,y+36);

             MoveTo(x+12,y+32); LineTo(x+26,y+32);
             MoveTo(x+8,y+36);  LineTo(x+32,y+36);
             MoveTo(x+20,y+22); LineTo(x+20,y+26);
```

```
            MoveTo(x+8, y+30); LineTo(x+6,y+28);
          MoveTo(x+32,y+30); LineTo(x+34,y+28);
        end;
end;

// рисует флаг
procedure TForm1.Flag(x, y : integer);
var
   p : array [0..3] of TPoint;  // координаты точек флажка
   m : array [0..4] of TPoint;  // буква М
begin
   // зададим координаты точек флажка
   p[0].x:=x+4;   p[0].y:=y+4;
   p[1].x:=x+30;  p[1].y:=y+12;
   p[2].x:=x+4;   p[2].y:=y+20;
   p[3].x:=x+4;   p[3].y:=y+36;  // нижняя точка древка

   m[0].x:=x+8; m[0].y:=y+14;
   m[1].x:=x+8; m[1].y:=y+8;
   m[2].x:=x+10; m[2].y:=y+10;
   m[3].x:=x+12; m[3].y:=y+8;
   m[4].x:=x+12; m[4].y:=y+14;

   with Canvas do
      begin
         // установим цвет кисти и карандаша
         Brush.Color := clRed;
         Pen.Color := clRed;

         Polygon(p);  // флажок

         // древко
         Pen.Color := clBlack;
         MoveTo(p[0].x, p[0].y);
         LineTo(p[3].x, p[3].y);

         // буква М
         Pen.Color := clWhite;
         Polyline(m);

         Pen.Color := clBlack;
      end;
end;
```

```
// выбор из меню ? команды "О программе"
procedure TForm1.Form1Create(Sender: TObject);
var
  row,col : integer;
begin
   // В неотображаемые элементы массива, которые соответствуют
   // клеткам по границе игрового поля, запишем число -3.
   // Это значение используется функцией Open для завершения
   // рекурсивного процесса открытия соседних пустых клеток
   for row :=0 to MR+1 do
        for col :=0 to MC+1 do
             Pole[row,col] := -3;

   NewGame();  // "разбросать" мины
   Form1.ClientHeight := H*MR + 1;
   Form1.ClientWidth := W*MC + 1;
end;

// нажатие кнопки мыши на игровом поле
procedure TForm1.Form1MouseDown(Sender: TObject; Button: TMouseButton;
                                Shift: TShiftState; X, Y: Integer);
var
   row, col : integer;
begin
   if status = 2 then // игра завершена
      exit;

   if status = 0 then // первый щелчок
      status := 1;

   // преобразуем координаты мыши в индексы клетки поля
   row := Trunc(y/H) + 1;
   col := Trunc(x/W) + 1;

   if Button = mbLeft then
      begin
         if Pole[row,col] = 9 then
            begin // открыта клетка, в которой есть мина
               Pole[row,col] := Pole[row,col] + 100;
               status := 2;  // игра закончена
               ShowPole(status);
            end
```

```
        else if Pole[row,col] < 9 then
                Open(row,col);
      end
   else
      if Button = mbRight then
         if Pole[row,col] > 200 then
            begin
               // уберем флаг и закроем клетку
               nFlag := nFlag - 1;
               Pole[row,col] := Pole[row,col]-200; // уберем флаг
               x := (col-1)* W + 1;
               y := (row-1)* H + 1;
               Canvas.Brush.Color := clLtGray;
               Canvas.Rectangle(x-1,y-1,x+W,y+H);
            end
         else
            begin // поставить в клетку флаг
               nFlag := nFlag + 1;
               if  Pole[row,col] = 9
                  then nMin := nMin + 1;
               Pole[row,col]:=Pole[row,col]+200;// поставили флаг
               if (nMin = NM) and (nFlag = NM) then
                   begin
                      status := 2;  // игра закончена
                      ShowPole(status);
                   end
               else Kletka(row, col, status);
            end;
end;

// выбор меню "Новая игра"
procedure TForm1.N1Click(Sender: TObject);
begin
  NewGame();
  ShowPole(status);
end;

// выбор из меню ? команды "Справка"
procedure TForm1.N3Click(Sender: TObject);
var
    h : HWND;
begin
    h := FindWindow('HH Parent','Сапер');
```

```
    if h = 0 then
        WinExec('hh.exe saper.chm', SW_RESTORE)
    else
        begin
          ShowWindow(h,SW_RESTORE);
          Windows.SetForegroundWindow(h);
        end;
end;

// команда "О программе"
procedure TForm1.N4Click(Sender: TObject);
begin
  Form2.ShowModal; // отобразить окно "О программе"
end;

// обработка события Paint
procedure TForm1.Form1Paint(Sender: TObject);
begin
  // отобразить игровое поле
  ShowPole(status);
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
    h : HWND;
begin
    h := FindWindow('HH Parent','Сапер');
    if h <> 0 then
        // открыто окно справочной информации
        SendMessage (h,WM_CLOSE,0,0);
end;

end.
```

Листинг 10.13. Модуль формы *О программе* (AboutForm.pas)

```
unit AboutForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
```

```
  Forms, Dialogs, StdCtrls,
  shellapi;  // для доступа к ShellExecute

type
  TForm2 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    procedure Label3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

// щелчок на URL (на Label3)
procedure TForm2.Label3Click(Sender: TObject);
begin
  { Чтобы функция ShellExecute была доступна, в директиву
    uses надо добавить ShellAPI. }
  ShellExecute(Application.Handle,
              'open', PChar(Label3.Caption),'', '', SW_RESTORE);

end;

end.
```

MP3-плеер

Программа "MP3-плеер", как не трудно догадаться, позволяет прослушивать файлы MP3. Отличительной ее особенностью является то, что в окне (рис. 10.15) не отображается заголовок (тем не менее пользователь может переместить окно), а регулировка громкости осуществляется непосредственно в окне программы.

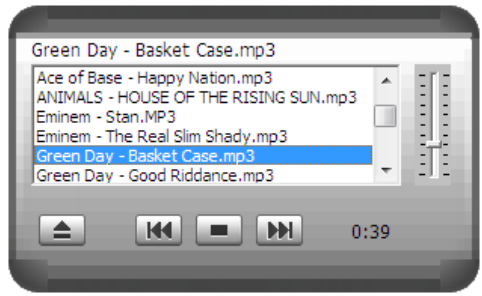


Рис. 10.15. Окно программы "MP3-плеер"

Форма

Форма программы "MP3-плеер" приведена на рис. 10.16.

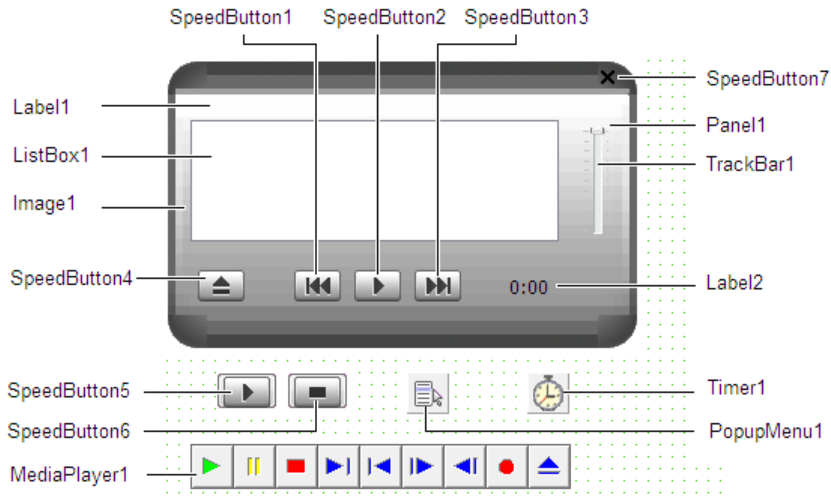


Рис. 10.16. Форма программы "MP3-плеер"

Компонент `Image1`, его следует поместить на форму первым, используется для отображения фонового рисунка. Компоненты `SpeedButton1`—`SpeedButton3` обеспечивают управление работой плеера. Кнопка `SpeedButton4` используется для активизации стандартного диалога **Обзор папок**. В поле компонента `ListBox1` во время работы программы отображается список MP3-файлов, находящихся в выбранном пользователем каталоге. Компонент `TrackBar1` служит для регулировки громкости воспроизведения. Компонент `Panel1` используется в качестве подложки компонента `TrackBar1`. Во время работы программы в поле компонента `Label1` отображается имя воспроизводимого файла, а в поле компонента `Label2` — время воспроизведения. Компонент `MediaPlayer1` обеспечивает воспроизведение MP3-

файлов. Компоненты `SpeedButton4` и `SpeedButton5` обеспечивают хранение картинок **Play** и **Stop**. Компонент `PopupMenu1` обеспечивает отображение контекстного меню, команда **Закрыть** которого используется для завершения работы программы (заголовок в окне программы не отображается, поэтому закрыть окно программы обычным образом, щелчком на кнопке **Закрыть**, нельзя).

Значения свойств формы приведены в табл. 10.6. Следует обратить внимание, что цвет формы (значение свойства `Color`) совпадает с "прозрачным" цветом (свойство `TransparentColor`), поэтому во время работы программы окно программы будет прозрачным.

Значения свойств компонентов приведены в табл. 10.7. После настройки компонентов следует изменить значения свойств `Width` и `Height` формы в соответствии с размером компонента `Image1` (в результате служебные компоненты `SpeedButton5` и `SpeedButton6`, а также компоненты `MediaPlayer1`, `Timer1` и `PoupupMenu1` окажутся за границей формы и в окне дизайнера формы не будут отображаться).

Таблица 10.6. *Значения свойств формы*

Свойство	Значение
`BorderStyle`	`bsNone`
`Color`	`clFuchsia`
`TransparentColor`	`clFuchsia`

Таблица 10.7. *Значения свойств компонентов*

Компонент	Свойство	Значение
`Image1`	`Height`	`193`
	`Width`	`214`
	`Picture`	
	`Stretch`	`True`
`SpeedButton1`	`NumGlyphs`	`2`
	`Glyph`	
	`Flat`	`True`
	`Enabled`	`False`
`SpeedButton2`	`NumGlyphs`	`2`
	`Glyph`	
	`Flat`	`True`
	`Enabled`	`False`

Таблица 10.7 *(окончание)*

Компонент	Свойство	Значение
SpeedButton3	NumGlyphs	2
	Glyph	
	Flat	True
	Enabled	False
SpeedButton4	NumGlyphs	1
	Glyph	
SpeedButton5	NumGlyphs	2
	Glyph	
SpeedButton6	NumGlyphs	2
	Glyph	
SpeedButton7	NumGlyphs	1
	Glyph	
Panel1	Width	24
	Height	81
	BevelOuter	bvNone
	Color	clSilver
TrackBar1	Orientation	trVertical
	Width	24
	Height	81
	TickMarks	tmBoth
	TumbLegth	10
	Hint	Громкость
	Showint	True
PoupupMenu1.Items[0]	Name	N1
	Caption	Закрыть
	Bitmap	
PoupupMenu1.Items[1]	Name	N2
	Caption	Свернуть
	Bitmap	
Timer1	Interval	1000

После настройки компонентов следует установить размер формы в соответствии с размером компонента `Image1` так, чтобы компоненты `MediaPlayer1`, `SpeedButton5` и `SpeedButton6` оказались за границей формы. В результате форма должна выглядеть так, как показано на рис. 10.17.

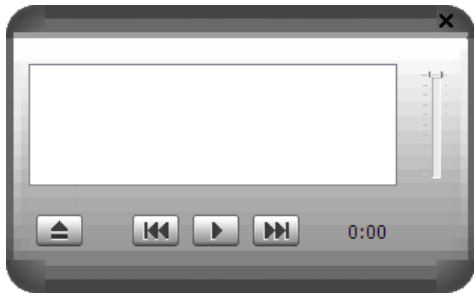


Рис. 10.17. Окончательный вид формы программы "MP3-плеер"

Регулятор громкости

Задать необходимую громкость воспроизведения MP3-файла можно с помощью API-функции `waveOutSetVolume`. Для того чтобы эта функция стала доступной, в текст программы надо поместить ссылку на модуль `MMSystem` (указать имя модуля в директиве `uses`).

Инструкция вызова функции в общем виде выглядит так:

```
r = waveOutSetVolume(ИдентификаторУстройства, Громкость)
```

Параметр *`ИдентификаторУстройства`* задает устройство воспроизведения (точнее, звуковой канал), громкость которого надо установить. При регулировке громкости воспроизведения MP3-файла значение этого параметра должно быть равно `WAVE_MAPPER` (константа `WAVE_MAPPER` определена в модуле `MMSystem`).

Параметр *`Громкость`* (двойное слово) 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 `$FFFFFFFF` (в Delphi при записи шестнадцатеричных констант используется префикс `$`). Уровню громкости 50% соответствует константа `$7FFF7FFF`.

Необходимо обратить внимание, что функция `waveOutSetVolume` регулирует громкость воспроизведения *звукового канала*, а не общий уровень звука.

Изменение громкости осуществляется с помощью компонента `TrackBar1`. Следует обратить внимание, что при вертикальном расположении компонента верхнему положению движка соответствует нулевое значение свойства `Position`, нижнему — значение, заданное свойством `Max`. Поэтому уровень громкости, соответствующий положению движка, вычисляется как разница между текущим и макси-

мально возможным значением свойства `Position` (это значение задает свойство `Max`), умноженная на `$FFFF`.

Непосредственное изменение громкости осуществляет процедура обработки события `Change` (листинг 10.14), регулятора громкости (компонента `TrackBar1`), которое происходит в результате перемещения движка мышью или клавишами перемещения курсора. Сначала она вычисляет значение громкости для левого канала, затем к полученному значению добавляет сдвинутое на 16 разрядов это же значение (в результате в старшем и младшем словах находятся одинаковые значения), и полученное таким образом значение передается в качестве параметра функции `waveOutSetVolume`.

Листинг 10.14. Обработка события `Change` компонента `TrackBar1`

```
// пользователь изменил положение регулятора громкости
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  volume := $FFFF * (TrackBar1.Max - TrackBar1.Position);
  volume := volume +   (volume shl 16);
  waveOutSetVolume(WAVE_MAPPER,volume);
end;
```

Перемещение окна

Значение свойства `BorderStyle` формы равно `bsNone`, поэтому в окне программы заголовок не отображается (см. рис. 10.5), и поэтому, на первый взгляд, пользователь, вроде бы, лишен возможности перемещения окна по экрану привычным для себя способом. Тем не менее окно программы все-таки переместить можно. Для этого надо установить указатель мыши в свободную (не занятую компонентами) точку окна, нажать левую кнопку мыши и, удерживая ее нажатой, перетащить окно в нужную точку экрана (такой способ весьма распространен). Описанный способ перемещения окна обеспечивает процедура обработки события `MouseDown` в поле компонента `Image1` (листинг 10.15). Процедура "обманывает" операционную систему, сообщает ей (путем посылки соответствующего сообщения), что пользователь нажал кнопку мыши в заголовке окна, т. е. выполнил действие, требующее перемещения окна.

Листинг 10.15. Обработка события `MouseDown` в поле компонента `Image1`

```
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
                                 Shift: TShiftState; X, Y: Integer);
begin
  // Заголовка у окна нет. Обманем Windows. Пусть ОС думает,
  // что кнопка нажата, и удерживается в заголовке окна.
```

```
  // В этом случае пользователь может перемещать окно обычным образом
  ReleaseCapture;
  SendMessage(Form1.Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0)
end;
```

Текст программы

Полный текст программы "MP3-плеер" приведен в листинге 10.16.

Листинг 10.16. MP3-плеер

```
{  MP3-плеер с регулятором громкости.
  (c) Культин Н.Б., 2003-2010 }

unit MainForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, MPlayer, ComCtrls,
  MMSYSTEM, FileCtrl, Menus; // эти ссылки вставлены вручную

type
  TForm1 = class(TForm)
     MediaPlayer1: TMediaPlayer;

    // кнопки
    SpeedButton1: TSpeedButton; // предыдущая композиция
    SpeedButton2: TSpeedButton; // Play/Stop
    SpeedButton3: TSpeedButton; // следующая композиция
    SpeedButton4: TSpeedButton; // выбор папки

    // невидимые кнопки SpeedButton5 и SpeedButton6 обеспечивают
    // хранение картинок Play и Stop
    SpeedButton5: TSpeedButton;
    SpeedButton6: TSpeedButton;

    ListBox1: TListBox;  // список композиций

    Timer1: TTimer;
```

```
    Label1: TLabel; // воспроизводимая композиция
    Label2: TLabel; // время воспроизведения

    Image1: TImage; // фон

    PopupMenu1: TPopupMenu; // контекстное меню
    N1: TMenuItem; // "Закрыть"
    N2: TMenuItem; // "Свернуть"

    // регулятор громкости
    Panel1: TPanel;       // панель
    TrackBar1: TTrackBar;

    procedure Image1MouseDown(Sender: TObject; Button: TMouseButton;
                              Shift: TShiftState; X, Y: Integer);
    procedure N2Click(Sender: TObject);
    procedure N1Click(Sender: TObject);

    procedure FormCreate(Sender: TObject);
    procedure ListBox1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure TrackBar1Change(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);

    // эти объявления вставлены сюда вручную
    procedure Play; // воспроизведение
    procedure PlayList(Path: string); // формирует список MP3-файлов

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}
var
  SoundPath: string[255];  // каталог, в котором находятся MP3-файлы

  min,sec: integer;  // время воспроизведения

  volume: LongWord; // громкость воспроизведения:
                    //    старшее слово — правый канал,
                    //    младшее слово — левый

procedure TForm1.FormCreate(Sender: TObject);
begin
   PlayList(''); // сформировать список MP3-файлов
   ListBox1.ItemIndex := 0;
   Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];

   // установить уровень громкости
   TrackBar1.Position := 7;

   // старшее слово переменной volume — правый канал,
   // младшее — левый
   volume := (TrackBar1.Position - TrackBar1.Max+1)* $FFFF;
   volume := volume +   (volume shl 16);
   waveOutSetVolume(WAVE_MAPPER,volume); // уровень громкости
end;

// формирует список MP3-файлов
procedure TForm1.PlayList(Path: string);
var
  SearchRec: TSearchRec; // структура SearchRec содержит информацию
                         // о файле, удовлетворяющем условию поиска
begin
  ListBox1.Clear;
  // сформировать список MP3-файлов
  if FindFirst(Path + '*.mp3', faAnyFile, SearchRec) = 0 then
     begin
       // В каталоге есть файл с расширением mp3.
       // Добавим имя этого файла в список
       ListBox1.Items.Add(SearchRec.Name);
```

```
        // Есть еще MP3-файлы?
        while (FindNext(SearchRec) = 0) do
               ListBox1.Items.Add(SearchRec.Name);
      end;
   ListBox1.ItemIndex := 0;
end;

// щелчок на названии произведения
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  if SpeedButton2.Tag = 0 then
     // вывести в поле метки Label1 имя выбранного файла
     Label1.Caption:=ListBox1.Items[ListBox1.itemIndex]
  else
     Form1.Play; // активизировать процесс воспроизведения
end;

// щелчок на кнопке "Воспроизведение"
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  // свойство Tag хранит информацию о состоянии
  // плеера: 0 — стоп; 1 — воспроизведение
  if SpeedButton2.Tag = 0 then
    begin  // начать воспроизведение
      Form1.Play;
    end
  else
    // если кнопка "Воспроизведение" нажата,
    // то повторное нажатие останавливает воспроизведение
    begin
      SpeedButton2.Tag := 0;
      MediaPlayer1.Stop;
      SpeedButton2.Glyph := SpeedButton5.Glyph;
      Timer1.Enabled := False;
      SPeedButton2.Hint := 'Play';
      Label2.Caption := '0:00';
    end;
end;

// кнопка "Предыдущий трек"
procedure TForm1.SpeedButton1Click(Sender: TObject);
```

```
begin
  if ListBox1.ItemIndex > 0 then
        ListBox1.ItemIndex := ListBox1.ItemIndex - 1;
  if SpeedButton2.Tag = 1 then
     Play;
end;

// кнопка "Следующий трек"
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  if ListBox1.ItemIndex < ListBox1.Count then
     ListBox1.ItemIndex := ListBox1.ItemIndex + 1;
  if SpeedButton2.Tag = 1 then
     Play;
end;

// пользователь изменил положение регулятора громкости
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  volume := $FFFF * (TrackBar1.Max - TrackBar1.Position);
  volume := volume +   (volume shl 16);
  waveOutSetVolume(WAVE_MAPPER,volume);
end;

// воспроизвести композицию, название которой выделено в списке ListBox1
procedure TForm1.Play;
begin
  Timer1.Enabled := False;
  Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];
  MediaPlayer1.FileName := SoundPath + ListBox1.Items[ListBox1.itemIndex];

  try
    Mediaplayer1.Open;
  except
    on EMCIDeviceError do
      begin
        ShowMessage('Ошибка обращения к файлу '+
                    ListBox1.Items[ListBox1.itemIndex]);
        exit;
      end;
  end;
```

```
  MediaPlayer1.Play;
  min :=0;
  sec :=0;
  Timer1.Enabled := True;
  SpeedButton2.Hint := 'Stop';
  SpeedButton2.Tag := 1;
end;

// сигнал таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // изменить счетчик времени
  if sec < 59
    then inc(sec)
    else begin
      sec :=0;
      inc(min);
    end;

  // вывести время воспроизведения
  Label2.Caption := IntToStr(min)+':';
  if sec < 10
    then Label2.Caption := Label2.Caption +'0'+ IntToStr(sec)
    else Label2.Caption := Label2.Caption + IntToStr(sec);

  // если воспроизведение текущей композиции не завершено
  if MediaPlayer1.Position < MediaPlayer1.Length
    then exit;

  // воспроизведение текущей композиции закончено
  Timer1.Enabled := False; // остановить таймер
  MediaPlayer1.Stop;       // остановить плеер

  if ListBox1.ItemIndex < ListBox1.Count // список не исчерпан
  then begin
        ListBox1.ItemIndex := ListBox1.ItemIndex + 1;
        Play;  // активизировать воспроизведение MP3-файла
       end
end;

// Щелчок на кнопке "Папка".
```

```
// Выбрать папку, в которой находятся MP3-файлы
procedure TForm1.SpeedButton4Click(Sender: TObject);
var
  Root: string;       // корневой каталог
  pwRoot : PWideChar;
  Dir: string;
begin
  Root := ''; // корневой каталог — папка Рабочий стол
  GetMem(pwRoot, (Length(Root)+1) * 2);
  pwRoot := StringToWideChar(Root,pwRoot,MAX_PATH*2);
  if not SelectDirectory('Выберите папку, в которой находятся MP3-файлы',
                         pwRoot, Dir)
     then Dir :=''
     else Dir := Dir+'\';

  // каталог, в котором находятся MP3-файлы, выбран
  SoundPath := Dir;
  PlayList(SoundPath);
end;

// нажатие кнопки мыши в поле компонента Image1
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;
                                 Shift: TShiftState; X, Y: Integer);
begin
  // Заголовка у окна нет. Обманем Windows. Пусть ОС думает,
  // что кнопка нажата, и удерживается в заголовке окна.
  // В этом случае пользователь может перемещать окно обычным образом
  ReleaseCapture;
  SendMessage(Form1.Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0)
end;

// выбор команды "Закрыть" в контекстном меню
procedure TForm1.N1Click(Sender: TObject);
begin
  Form1.Close;
end;

end.
```

ПРИЛОЖЕНИЯ

Приложение 1

Справочник

В данном *приложении* приведено описание базовых компонентов, а также часто используемых функций. Описание других компонентов можно найти в справочной системе среды разработки.

Форма

Форма (объект типа `TForm`) является основой программы. Свойства формы (табл. П1.1) определяют вид окна программы.

Таблица П1.1. *Свойства формы (объекта* `TForm`*)*

Свойство	Описание
`Name`	Имя (идентификатор) формы. Используется для доступа к форме, ее свойствам и методам, а также для доступа к компонентам формы
`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
`Height`	Высота формы
`Position`	Положение окна в момент первого его появления на экране (`poCenterScreen` — в центре экрана; `poOwnerFormCenter` — в центре родительского окна; `poDesigned` — положение окна определяют значения свойств `Top` и `Left`)
`ClientWidth`	Ширина рабочей (клиентской) области формы, т. е. без учета ширины левой и правой границ
`ClientHeight`	Высота рабочей (клиентской) области формы, т. е. без учета высоты заголовка и ширины нижней границы формы

Таблица П1.1 *(окончание)*

Свойство	Описание
BorderStyle	Вид границы. Граница может быть обычной (bsSizeable), тонкой (bsSingle) или вообще отсутствовать (bsNone). Если у окна обычная граница, то во время работы программы пользователь может с помощью мыши изменить размер окна. Изменить размер окна с тонкой границей нельзя. Если граница отсутствует, то на экран во время работы программы будет выведено окно без заголовка. Положение и размер такого окна во время работы программы изменить нельзя
BorderIcons	Кнопки управления окном. Значение свойства определяет, какие кнопки управления окном будут доступны пользователю во время работы программы. Значение свойства задается путем присвоения значений уточняющим свойствам biSystemMenu, biMinimize, biMaximize и biHelp. Свойство biSystemMenu определяет доступность кнопки системного меню, biMinimize — кнопки **Свернуть**, biMaximize — кнопки **Развернуть**, biHelp — кнопки вывода справочной информации
Icon	Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню
Color	Цвет фона. Цвет можно задать, указав название цвета или привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой, выбранным компонентом привязки и меняется при изменении цветовой схемы операционной системы
Font	Шрифт, используемый "по умолчанию" компонентами, находящимися на поверхности формы. Изменение свойства Font формы приводит к автоматическому изменению свойства Font компонента, располагающегося на поверхности формы. То есть компоненты наследуют свойство Font от формы (имеется возможность запретить наследование)
Canvas	Поверхность, на которую можно вывести графику

Базовые компоненты

В этом разделе приведено краткое описание базовых компонентов. Подробное описание этих и других компонентов можно найти в справочной системе.

Label

Компонент Label (рис. П1.1) предназначен для вывода текста на поверхность формы. Свойства компонента (табл. П1.2) определяют вид и расположение текста.

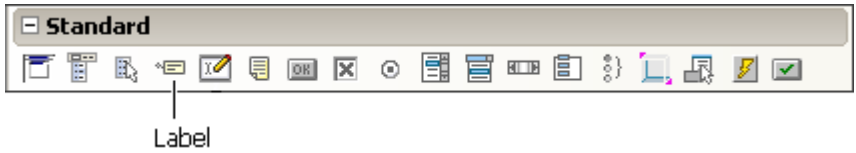


Рис. П1.1. Компонент `Label`

Таблица П1.2. *Свойства компонента* `Label`

Свойство	**Описание**
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст, отображаемый в поле компонента
`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 (значение свойства `AutoSize` должно быть `False`)
`Alignment`	Задает способ выравнивания текста внутри поля. Текст может быть выровнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`LayOut`	Задает способ размещения текста по вертикали. Текст может быть прижат к верхней границе поля компонента (`tlTop`), к нижней границе (`tlBottom`) или размещен по центру (`tlCenter`)
`Font`	Шрифт, используемый для отображения текста. Уточняющие свойства определяют шрифт (`Name`), размер (`Size`), стиль (`Style`) и цвет символов (`Color`)
`ParentFont`	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно `True`, то текст выводится шрифтом, установленным для формы
`Color`	Цвет фона области вывода текста
`Transparent`	Управляет отображением фона области вывода текста. Значение `True` делает область вывода текста прозрачной (область вывода не закрашивается цветом, заданным свойством `Color`)
`Visible`	Позволяет скрыть текст (`False`) или сделать его видимым (`True`)

Edit

Компонент `Edit` (рис. П1.2) представляет собой поле ввода/редактирования строки символов. Свойства компонента приведены в табл. П1.3.

Рис. П1.2. Компонент `Edit`

Таблица П1.3. *Свойства компонента* `Edit`

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 ввода и 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 вводимого текста
`ParentFont`	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно `True`, то при изменении свойства `Font` формы автоматически меняется значение свойства `Font` компонента
`Enabled`	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно `False`, то текст в поле редактирования изменить нельзя
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Button

Компонент `Button` (рис. П1.3) представляет собой командную кнопку. Свойства компонента приведены в табл. П1.4.

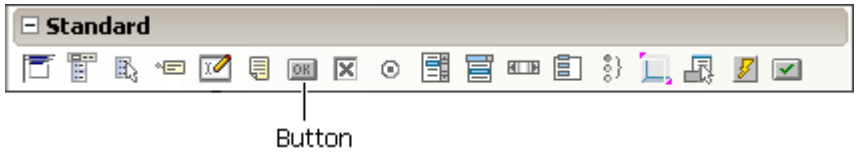


Рис. П1.3. Компонент `Button`

***Таблица П1.4.** Свойства компонента* `Button`

Свойство	Описание
`Name`	Имя компонента. Используется 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
`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, например в результате щелчка на кнопке событие `Click` не возникает
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Hint`	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, значение свойства `ShowHint` должно быть `True`)
`ShowHint`	Разрешает (`True`) или запрещает (`False`) отображение подсказки при позиционировании указателя на кнопке

Memo

Компонент `Memo` (рис. П1.4) представляет собой элемент редактирования текста, который может состоять из нескольких строк. Свойства компонента приведены в табл. П1.5.

Рис. П1.4. Компонент `Memo`

Таблица П1.5. *Свойства компонента* `Memo`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Text`	Текст, находящийся в поле **Memo** (свойство доступно только во время работы программы)
`Lines`	Массив строк, соответствующий содержимому поля. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
`Lines.Count`	Количество строк текста в поле **Memo** (количество элементов в массиве `Lines`)
`Left`	Расстояние от левой границы поля до левой границы формы
`Top`	Расстояние от верхней границы поля до верхней границы формы
`Height`	Высота поля
`Width`	Ширина поля
`Font`	Шрифт, используемый для отображения вводимого текста
`ParentFont`	Признак наследования свойств шрифта родительской формы

RadioButton

Компонент `RadioButton` (рис. П1.5) представляет зависимую кнопку (переключатель), состояние которой определяется состоянием других кнопок группы. Свойства компонента приведены в табл. П1.6.

Если в диалоговом окне надо организовать несколько групп переключателей, то каждую группу следует представить компонентом `RadioGroup`.

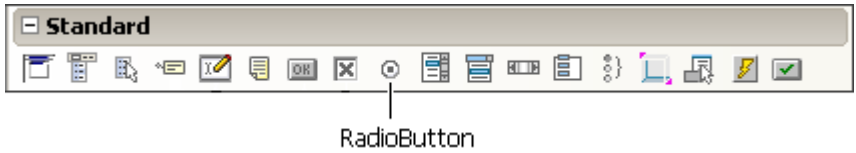


Рис. П1.5. Компонент `RadioButton`

Таблица П1.6. *Свойства компонента* `RadioButton`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Caption`	Текст, который находится справа от кнопки

Таблица П1.6 *(окончание)*

Свойство	Описание
`Checked`	Состояние, внешний вид кнопки: если кнопка выбрана, то `Checked = True`; если кнопка не выбрана, то `Checked = False`
`Left`	Расстояние от левой границы переключателя до левой границы формы
`Top`	Расстояние от верхней границы переключателя до верхней границы формы
`Height`	Высота поля вывода поясняющего текста
`Width`	Ширина поля вывода поясняющего текста
`Font`	Шрифт, используемый для отображения поясняющего текста
`ParentFont`	Признак наследования характеристик шрифта родительской формы

CheckBox

Компонент `CheckBox` (рис. П1.6) представляет собой независимую кнопку (флажок). Свойства компонента приведены в табл. П1.7.

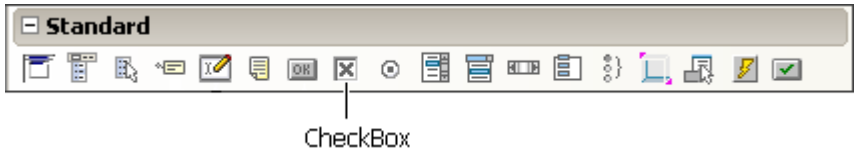


Рис. П1.6. Компонент `CheckBox`

Таблица П1.7. *Свойства компонента* `CheckBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Caption`	Текст, который находится справа от флажка
`Checked`	Состояние, внешний вид флажка: если флажок установлен (в квадратике есть "галочка"), то `Checked = True`; если флажок сброшен (нет "галочки"), то `Checked = False`
`State`	Состояние флажка. В отличие от свойства `Checked`, позволяет различать установленное, сброшенное и промежуточное состояния. Состояние флажка определяет одна из констант: `cbChecked` (установлен); `cbGrayed` (серый, неопределенное состояние); `cbUnChecked` (сброшен)

Таблица П1.7 *(окончание)*

Свойство	Описание
`AllowGrayed`	Свойство определяет, может ли флажок быть в промежуточном состоянии: если `AllowGrayed = False`, то флажок может быть только установленным или сброшенным; если значение `AllowGrayed = True`, то допустимо промежуточное состояние
`Left`	Расстояние от левой границы флажка до левой границы формы
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота поля вывода поясняющего текста
`Width`	Ширина поля вывода поясняющего текста
`Font`	Шрифт, используемый для отображения поясняющего текста
`ParentFont`	Признак наследования характеристик шрифта родительской формы

ListBox

Компонент `ListBox` (рис. П1.7) представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. П1.8.

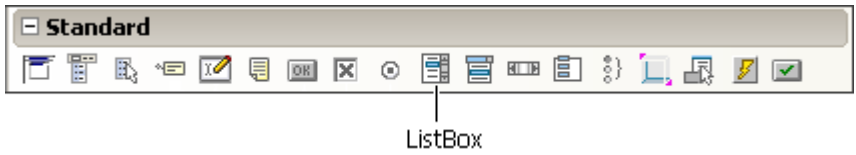


Рис. П1.7. Компонент `ListBox`

Таблица П1.8. *Свойства компонента* `ListBox`

Свойство	Описание
`Name`	Имя компонента. В программе используется для доступа к компоненту и его свойствам
`Items`	Элементы списка — массив строк
`Items.Count`	Количество элементов списка
`ItemIndex`	Номер выбранного элемента (элементы списка нумеруются с нуля). Если в списке ни один из элементов не выбран, то значение свойства равно –1 (минус один)
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента

***Таблица П1.8** (окончание)*

Свойство	Описание
`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
`ParentFont`	Признак наследования свойств шрифта родительской формы

ComboBox

Компонент `ComboBox` (рис. П1.8) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или выбора из списка. Свойства компонента приведены в табл. П1.9.

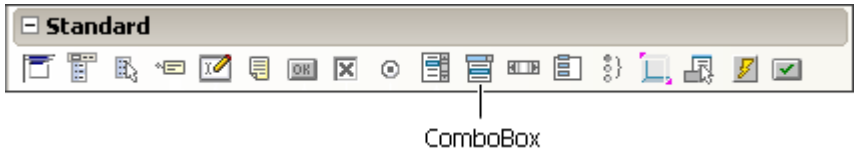


Рис. П1.8. Компонент `ComboBox`

***Таблица П1.9.** Свойства компонента `ComboBox`*

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Text`	Текст, находящийся в поле ввода/редактирования
`Items`	Элементы списка — массив строк
`Count`	Количество элементов списка
`ItemIndex`	Номер элемента, выбранного в списке. Если ни один из элементов списка не был выбран, то значение свойства равно –1 (минус один)
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента
`DropDownCount`	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше, чем `DropDownCount`, то появляется вертикальная полоса прокрутки

Таблица П1.9 *(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 (поля ввода/редактирования)
`Width`	Ширина компонента
`Font`	Шрифт, используемый для отображения элементов списка
`ParentFont`	Признак наследования свойств шрифта родительской формы

StringGrid

Компонент `StringGrid` (рис. П1.9) представляет собой таблицу, ячейки которой содержат строки символов. Свойства компонента приведены в табл. П1.10.

Рис. П1.9. Компонент `StringGrid`

Таблица П1.10. *Свойства компонента* `StringGrid`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`ColCount`	Количество колонок таблицы
`RowCount`	Количество строк таблицы
`DefaultColWidth`	Ширина колонок таблицы
`DefaultRowHeight`	Высота строк таблицы
`FixedCols`	Количество зафиксированных слева колонок таблицы. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте
`FixedRows`	Количество зафиксированных сверху строк таблицы. Зафиксированные строки выделяются цветом и при вертикальной прокрутке таблицы остаются на месте

Таблица П1.10 *(окончание)*

Свойство	Описание
`Cells`	Соответствующий таблице двумерный массив. Ячейке таблицы, находящейся на пересечении столбца с номером `col` и строки с номером `row`, соответствует элемент `cells[col][row]`
`GridLineWidth`	Ширина линий, ограничивающих ячейки таблицы
`Left`	Расстояние от левой границы поля таблицы до левой границы формы
`Top`	Расстояние от верхней границы поля таблицы до верхней границы формы
`Height`	Высота поля таблицы
`Width`	Ширина поля таблицы
`Options.goEditing`	Признак допустимости редактирования содержимого ячеек таблицы. `True` — редактирование разрешено, `False` — запрещено
`Options.goTab`	Разрешает (`True`) или запрещает (`False`) использование клавиши <Tab> для перемещения курсора в следующую ячейку таблицы
`Options.goAlwaysShowEditor`	Признак нахождения компонента в режиме редактирования. Если значение свойства `False`, то для того, чтобы в ячейке появился курсор, надо начать набирать текст, нажать клавишу <F2> или сделать щелчок мышью
`Font`	Шрифт, используемый для отображения содержимого ячеек таблицы
`ParentFont`	Признак наследования характеристик шрифта формы

Image

Компонент `Image` (рис. П1.10) обеспечивает вывод на поверхность формы иллюстраций, представленных в BMP-формате (чтобы компонент можно было использовать для отображения иллюстраций в формате JPEG, надо подключить модуль `JPEG` — включить в текст программы директиву `uses JPEG`). Свойства компонента `Image` приведены в табл. П1.11.

Рис. П1.10. Компонент `Image`

Таблица П1.11. *Свойства компонента* `Image`

Свойство	Описание
`Picture`	Иллюстрация, которая отображается в поле компонента
`Width`, `Height`	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств `AutoSize`, `Stretch` и `Proportional` равно `False`, то отображается часть иллюстрации
`Proportional`	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства `AutoSize` должно быть `False`
`Stretch`	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена
`AutoSize`	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
`Center`	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно `False`, то картинка прижата к правой границе компонента, если `True` — то картинка располагается по центру
`Visible`	Отображается ли компонент и, соответственно, иллюстрация на поверхности формы
`Canvas`	Поверхность, на которую можно вывести графику

Timer

Компонент `Timer` (рис. П1.11) обеспечивает генерацию последовательности событий `Timer`. Свойства компонента приведены в табл. П1.12.

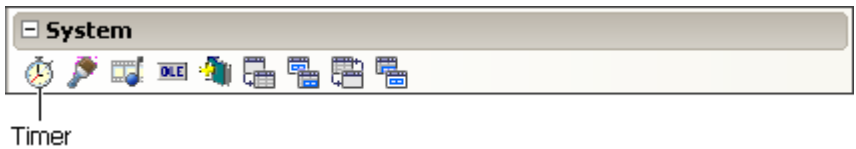


Рис. П1.11. Компонент `Timer`

Таблица П1.12. *Свойства компонента* `Timer`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту
`Interval`	Период генерации события `Timer`. Задается в миллисекундах

Таблица П1.12 *(окончание)*

Свойство	Описание
`Enabled`	Разрешение работы. Разрешает (значение `True`) или запрещает (значение `False`) генерацию события `Timer`

SpeedButton

Компонент `SpeedButton` (рис. П1.12) представляет собой кнопку, на поверхности которой находится картинка. Свойства компонента приведены в табл. П1.13.

Рис. П1.12. Компонент `SpeedButton`

Таблица П1.13. *Свойства компонента* `SpeedButton`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Glyph`	Битовый образ, в котором находятся картинки для каждого из состояний кнопки. В битовом образе может быть до четырех изображений кнопки (рис. П1.13)
`NumGlyphs`	Количество картинок в битовом образе `Glyph`
`Flat`	Свойство `Flat` определяет вид кнопки (наличие границы). Если значение свойства равно `True`, то граница кнопки появляется только при позиционировании указателя мыши на кнопке
`GroupIndex`	Идентификатор группы кнопок. Кнопки, имеющие одинаковый идентификатор группы, работают подобно переключателям: нажатие одной из кнопок группы вызывает срабатывание других кнопок этой группы. Чтобы кнопку можно было зафиксировать, значение свойства `GroupIndex` не должно быть равно нулю
`Down`	Идентификатор состояния кнопки. Изменить значение свойства можно, если значение свойства `GroupIndex` не равно 0
`AllowAllUp`	Свойство определяет возможность отжатия кнопки. Если кнопка нажата и значение свойства равно `True`, то кнопку можно отжать
`Left`	Расстояние от левой границы кнопки до левой границы формы

Таблица П1.13 *(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
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Hint`	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, значение свойства `ShowHint` должно быть `True`)
`ShowHint`	Разрешает (`True`) или запрещает (`False`) отображение подсказки при позиционировании указателя на кнопке

Рис. П1.13. Структура и пример битового образа `Glyph`: картинки, соответствующие состоянию кнопки

UpDown

Компонент `UpDown` (рис. П1.14) представляет собой две кнопки, используя которые можно изменить значение внутренней переменной-счетчика на определенную величину. Увеличение или уменьшение значения происходит при каждом щелчке на одной из кнопок. Свойства компонента приведены в табл. П1.14.

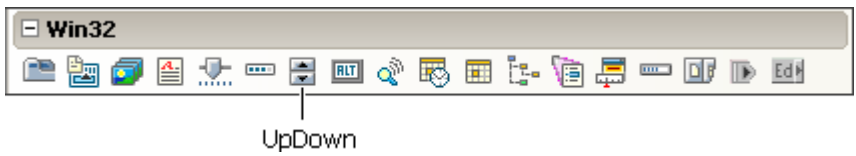


Рис. П1.14. Компонент `UpDown`

Таблица П1.14. *Свойства компонента* `UpDown`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Position`	Счетчик. Значение свойства изменяется в результате щелчка на кнопке `Up` (увеличивается) или `Down` (уменьшается). Диапазон изменения определяют свойства `Min` и `Max`, величину изменения — свойство `Increment`
`Min`	Нижняя граница диапазона изменения свойства `Position`
`Max`	Верхняя граница диапазона изменения свойства `Position`
`Increment`	Величина, на которую изменяется значение свойства `Position` в результате щелчка на одной из кнопок компонента
`Associate`	Определяет компонент (`Edit` — поле ввода/редактирования), используемый в качестве индикатора значения свойства `Position`. Если значение свойства задано, то при изменении содержимого поля редактирования автоматически меняется значение свойства `Position`
`Orientation`	Задает ориентацию кнопок компонента. Кнопки могут быть ориентированы вертикально (`udVertical`) или горизонтально (`udHorizontal`)

OpenDialog

Компонент `OpenDialog` (рис. П1.15) представляет собой диалог **Открыть**. Свойства компонента приведены в табл. П1.15. Отображение диалога обеспечивает метод `Execute`, значение которого позволяет определить, щелчком на какой кнопке, **Открыть** или **Отмена**, пользователь закрыл диалог.

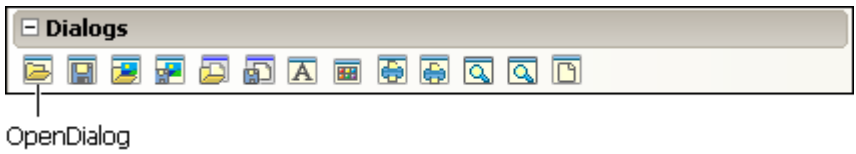


Рис. П1.15. Компонент `OpenDialog`

Таблица П1.15. *Свойства компонента* `OpenDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Открыть**

Таблица П1.15 *(окончание)*

Свойство	Описание
`Filter`	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют выбранному (текущему) фильтру. Во время отображения диалога пользователь может выбрать фильтр в списке **Тип файлов**. Каждый фильтр задается строкой вида *описание*`|`*маска*, например `Текст|*.txt`
`FilterIndex`	Если в списке `Filter` несколько элементов (например, `Текст|*.txt|Все файлы|*.*`), то значение свойства задает фильтр, который используется в момент появления диалога на экране
`InitialDir`	Каталог, содержимое которого отображается при появлении диалога на экране. Если значение свойства не указано, то в окне диалога отображается содержимое папки Мои документы
`FileNane`	Имя файла, выбранного пользователем

SaveDialog

Компонент `SaveDialog` (рис. П1.16) представляет собой диалог **Сохранить**. Свойства компонента приведены в табл. П1.16. Отображение диалога обеспечивает метод `Execute`, значение которого позволяет определить, щелчком на какой кнопке, **Сохранить** или **Отмена**, пользователь закрыл диалог.

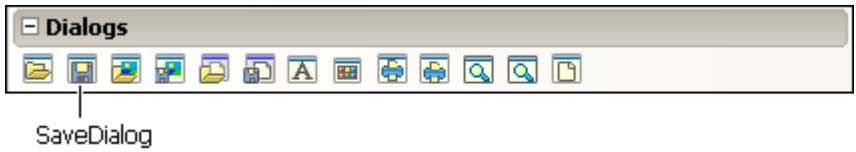


Рис. П1.16. Компонент `SaveDialog`

Таблица П1.16. *Свойства компонента* `SaveDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Сохранить как**
`Filter`	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют выбранному (текущему) фильтру. Во время отображения диалога пользователь может выбрать фильтр в списке **Тип файлов**. Каждый фильтр задается строкой вида *описание*`|`*маска*, например `Текст|*.txt`

Таблица П1.16 *(окончание)*

Свойство	Описание
`FilterIndex`	Если в списке `Filter` несколько элементов (например, `Текст|*.txt|Все файлы|*.*`), то значение свойства задает фильтр, который используется в момент появления диалога на экране
`InitialDir`	Каталог, содержимое которого отображается при появлении диалога на экране. Если значение свойства не указано, то в окне диалога отображается содержимое папки Мои документы
`FileNane`	Имя файла, введенное пользователем в поле **Имя файла**
`DefaultExt`	Расширение, которое будет добавлено к имени файла, если в поле **Имя файла** пользователь не задаст расширение файла

Animate

Компонент `Animate` (рис. П1.17) позволяет воспроизводить простую, не сопровождаемую звуком анимацию, кадры которой находятся в AVI-файле. Свойства компонента приведены в табл. П1.17.

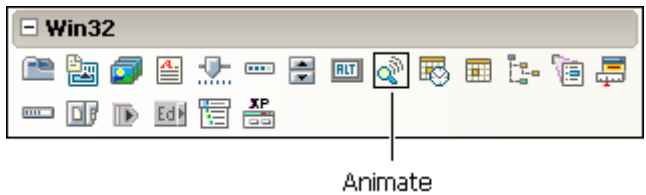


Рис. П1.17. Компонент `Animate`

Таблица П1.17. *Свойства компонента* `Animate`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента и управления его поведением
`FileName`	Имя AVI-файла, в котором находится анимация, отображаемая с помощью компонента
`StartFrame`	Номер кадра, с которого начинается отображение анимации
`StopFrame`	Номер кадра, на котором заканчивается отображение анимации
`Activate`	Признак активизации процесса отображения кадров анимации
`Color`	Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация

Таблица П1.17 *(окончание)*

Свойство	Описание
`Transparent`	Режим использования "прозрачного" цвета при отображении анимации
`Repetitions`	Количество повторов отображения анимации

MediaPlayer

Компонент `MediaPlayer` (рис. П1.18) позволяет воспроизвести видеоролик, звук и сопровождаемую звуком анимацию. Свойства компонента приведены в табл. П1.18.

Рис. П1.18. Компонент `MediaPlayer`

Таблица П1.18. *Свойства компонента* `MediaPlayer`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента и управления работой плеера
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент `MediaPlayer`. Тип устройства задается именованной константой: `dtAutoSelect` — тип устройства определяется автоматически; `dtWaveAudio` — проигрыватель звука; `dtAVIVideo` — видеопроигрыватель; `dtCDAudio` — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
`AutoOpen`	Признак автоматического открытия файла видеоролика или звукового фрагмента сразу после запуска программы
`Display`	Определяет компонент, на поверхности которого воспроизводится видеоролик (обычно в качестве экрана для отображения видео используют компонент `Panel`)
`VisibleButtons`	Составное свойство. Определяет видимые кнопки компонента. Позволяет сделать невидимыми некоторые кнопки

Компоненты доступа/манипулирования данными

ADOConnection

Компонент `ADOConnection` (рис. П1.19) обеспечивает соединение с базой данных. Свойства компонента приведены в табл. П1.19.

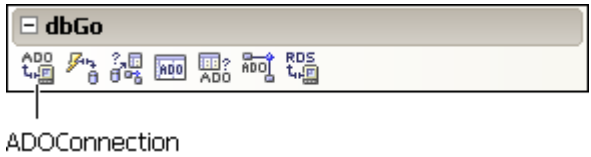


Рис. П1.19. Компонент `ADOConnection`

Таблица П1.19. *Свойства компонента* `ADOConnection`

Свойство	Описание
`ConnectionString`	Строка соединения. Содержит информацию, необходимую для подключения к базе данных
`LoginPrompt`	Признак необходимости запросить имя и пароль пользователя в момент подключения к базе данных. Если значение свойства равно `False`, то окно **Login** в момент подключения к базе данных не отображается
`Mode`	Режим соединения. Соединение с базой данных может быть открыто для чтения (`cmRead`), записи (`cmWrite`), чтения/записи (`cmReadWrite`)
`Connected`	Признак того, что соединение установлено

ADOTable

Компонент `ADOTable` (рис. П1.20) представляет собой таблицу (все столбцы) базы данных (обеспечивает доступ к таблице). Свойства компонента приведены в табл. П1.20.

Рис. П1.20. Компонент `ADOTable`

Таблица П1.20. *Свойства компонента* `ADOTable`

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`ConnectionString`	Строка соединения. Содержит информацию о базе данных, элементом которой является таблица, подключение к которой обеспечивает компонент
`TableName`	Таблица базы данных
`Filter`	Фильтр — условие отбора записей из таблицы. Позволяет скрыть записи, не удовлетворяющие критерию отбора
`Filtered`	Признак фильтрации записей. Если значение свойства равно `True`, то записи, не удовлетворяющие критерию запроса, не отображаются
`ReadOnly`	Запрещает (`True`) или разрешает (`False`) изменение данных таблицы
`Active`	Признак того, что соединение с таблицей активно

ADODataSet

Компонент `ADODataSet` (рис. П1.21) хранит данные, полученные из базы данных в результате выполнения сервером SQL-команды. В отличие от компонента `ADOTable`, который представляет собой всю таблицу (все столбцы), компонент позволяет прочитать только нужные столбцы или может быть заполнен данными из разных таблиц. Свойства компонента `ADODataSet` приведены в табл. П1.21.

Рис. П1.21. Компонент `ADODataSet`

Таблица П1.21. *Свойства компонента* `ADODataSet`

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`CommandText`	Команда, которая направляется серверу

Таблица П1.21 *(окончание)*

Свойство	Описание
`Parameters`	Параметры команды
`Filter`	Фильтр. Позволяет скрыть записи, не удовлетворяющие критерию отбора
`Filtered`	Признак использования фильтра
`Activate`	Открывает или делает недоступным набор данных

ADOQuery

Компонент `ADOQuery` (рис. П1.22) позволяет направить запрос серверу базы данных. Обычно он используется, если данные, которые надо получить из базы данных, распределены по нескольким таблицам (если данные находятся в одной таблице, то рекомендуется использовать компонент `ADOTable`). Свойства компонента приведены в табл. П1.22.

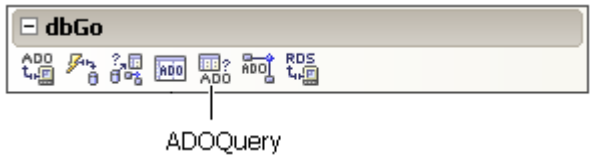


Рис. П1.22. Компонент `ADOQuery`

Таблица П1.22. Свойства компонента *`ADOQuery`*

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`ConnectionString`	Строка соединения. Содержит информацию о базе данных, которой направляется запрос (SQL-команда)
`SQL`	SQL-команда (запрос), которая направляется серверу
`Filter`	Фильтр — условие отбора записей из таблицы. Позволяет скрыть записи, не удовлетворяющие критерию отбора
`Filtered`	Признак фильтрации записей. Если значение свойства равно `True`, то записи, не удовлетворяющие критерию запроса, не отображаются
`Active`	Признак того, что соединение с таблицей активно

DataSource

Компонент DataSource (рис. П1.23) обеспечивает связь между данными, представленными, например компонентом ADODataSet, ADOTable или ADOQuery и компонентом, обеспечивающим отображение данных, например DBGrid, DBEdit, DBMemo или DBText. Свойства компонента приведены в табл. П1.23.

Рис. П1.23. Компонент DataSource

Таблица П1.23. *Свойства компонента* DataSource

Свойство	Определяет
Name	Имя компонента. Используется компонентом отображения данных для доступа к компоненту DataSource и, следовательно, к данным, связь с которыми он обеспечивает
DataSet	Компонент, представляющий собой данные (например, ADODataSet, ADOTable или ADOQuery)

DBEdit, DBMemo, DBText

Компоненты DBEdit и DBMemo (рис. П1.24) обеспечивают просмотр и редактирование полей записи базы данных, компонент DBText — только просмотр. Свойства компонентов приведены в табл. П1.24.

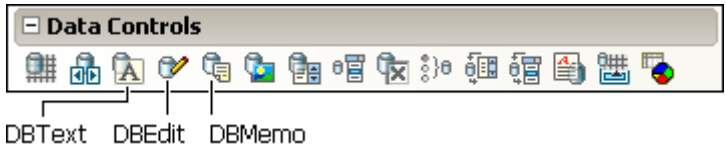


Рис. П1.24. Компоненты просмотра и редактирования полей БД

Таблица П1.24. *Свойства компонентов* DBText, DBEdit *и* DBMemo

Свойство	Определяет
Name	Имя компонента. Используется для доступа к свойствам компонента
DataSource	Компонент-источник данных

Таблица П1.24 *(окончание)*

Свойство	Определяет
`DataField`	Поле базы данных, для отображения или редактирования которого используется компонент

DBGrid

Компонент `DBGrid` (рис. П1.25) используется для просмотра и редактирования базы данных в режиме таблицы. Свойства компонента приведены в табл. П1.25.

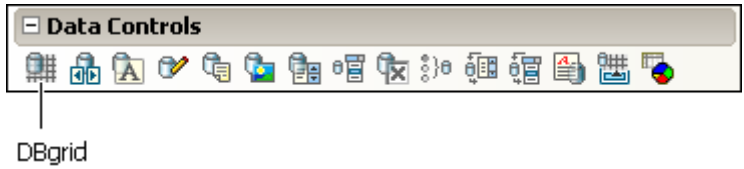


Рис. П1.25. Компонент `DBGrid`

Таблица П1.25. *Свойства компонента* `DBGrid`

Свойство	Описание
`Name`	Имя компонента
`DataSource`	Источник отображаемых в таблице данных (компонент `DataSource`)
`Columns`	Свойство `Columns` представляет собой массив объектов типа `TColumn`, каждый из которых определяет колонку таблицы и отображаемую в ней информацию (см. табл. П1.26)
`Options`	Определяет настройку компонента
`Options.dgTitles`	Разрешает вывод строки заголовка столбцов
`Options.dgIndicator`	Разрешает вывод колонки индикатора. Во время работы с базой данных текущая запись помечается в колонке индикатора треугольником, новая запись — звездочкой, редактируемая — специальным значком
`Options.dgColumnResize`	Разрешает менять во время работы программы ширину колонок таблицы
`Options.dgColLines`	Разрешает выводить линии, разделяющие колонки таблицы
`Options.dgRowLines`	Разрешает выводить линии, разделяющие строки таблицы

Таблица П1.26. *Свойства объекта* `TColumn`

Свойство	Определяет
`FieldName`	Поле записи, содержимое которого выводится в колонке
`Width`	Ширина колонки в пикселах
`Font`	Шрифт, используемый для вывода текста в ячейках колонки
`Color`	Цвет фона колонки
`Alignment`	Способ выравнивания текста в ячейках колонки. Текст может быть выровнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Title.Caption`	Заголовок колонки. Значением по умолчанию является имя поля записи
`Title.Alignment`	Способ выравнивания заголовка колонки. Заголовок может быть выровнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Title.Color`	Цвет фона заголовка колонки
`Title.Font`	Шрифт заголовка колонки

DBNavigator

Компонент `DBNavigator` (рис. П1.26 и П1.27) обеспечивает перемещение указателя текущей записи, активизацию режима редактирования, добавление и удаление записей.

Рис. П1.26. Значок компонента `DBNavigator`

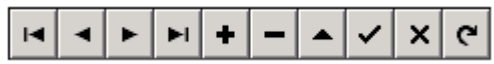


Рис. П1.27. Компонент `DBNavigator`

Компонент представляет собой совокупность командных кнопок (табл. П1.27). Свойства компонента приведены в табл. П1.28.

Таблица П1.27. *Кнопки компонента* `DBNavigator`

Кнопка		Обозначение	Действие
	**К первой**	`nbFirst`	Указатель текущей записи перемещается к первой записи файла данных
	**К предыдущей**	`nbPrior`	Указатель текущей записи перемещается к предыдущей записи файла данных
	**К следующей**	`nbNext`	Указатель текущей записи перемещается к следующей записи файла данных
	**К последней**	`nbLast`	Указатель текущей записи перемещается к последней записи файла данных
	**Добавить**	`nbInsert`	В файл данных добавляется новая запись
	**Удалить**	`nbDelete`	Удаляется текущая запись файла данных
	**Редактирование**	`nbEdit`	Устанавливает режим редактирования текущей записи
	**Сохранить**	`nbPost`	Изменения, внесенные в текущую запись, записываются в файл данных
	**Отменить**	`Cancel`	Отменяет внесенные в текущую запись изменения
	**Обновить**	`nbRefresh`	Записывает внесенные изменения в файл

Таблица П1.28. *Свойства компонента* `DBNavigator`

Свойство	Определяет
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`DataSource`	Имя компонента, являющегося источником данных. В качестве источника данных может выступать база данных (компонент `Database`), таблица (компонент `Table`) или результат выполнения запроса (компонент `Query`)
`VisibleButtons`	Видимые командные кнопки

Графика

PaintBox

Компонент `PaintBox` (рис. П1.28) представляет собой область, в которую программа может вывести графику. Вывод графики обеспечивают методы свойства `Canvas`.

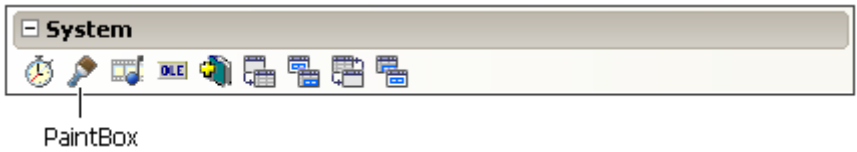


Рис. П1.28. Компонент `PaintBox`

Canvas

`Canvas` — это объект `TCanvas`, поверхность (формы, компонента `PaintBox` или компонента `Image`), на которую программа может вывести графику.

Вывод графики обеспечивают методы объекта `TCanvas` (табл. П1.29), свойства объекта `TCanvas` (табл. П1.30) определяют вид графических элементов.

Таблица П1.29. *Методы объекта* `TCanvas`

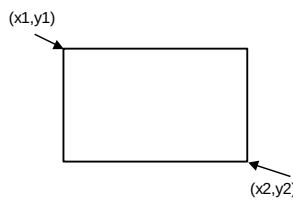
Метод	Описание
`TextOut(x,y,s)`	Выводит строку `s` от точки с координатами (`x`, `y`). Шрифт определяет свойство `Font` поверхности (`Canvas`), на которую выводится тест, цвет закраски области вывода текста — свойство `Brush`
`Draw(x,y,b)`	Выводит от точки с координатами (`x`, `y`) битовый образ `b`. Если значение свойства `Transparent` поверхности, на которую выполняется вывод, равно `True`, то точки, цвет которых совпадает с цветом левой нижней точки битового образа, не отображаются
`LineTo(x,y)`	Вычерчивает линию из текущей точки в точку с указанными координатами. Чтобы начертить линию из точки (`x1`, `y1`) в точку (`x2`, `y2`), надо сначала вызвать метод `MoveTo(x1,y2)`, затем — `LineTo(x2,y2)`. Вид линии определяет свойство `Pen`
`MoveTo(x,y)`	Перемещает указатель текущей точки (карандаш) в точку с указанными координатами
`Rectangle(x1,y1, x2,y2)`	Вычерчивает прямоугольник. Параметры `x1`, `y1`, `x2` и `y2` задают координаты левого верхнего и правого нижнего углов. Вид линии определяет свойство `Pen`, цвет и способ закраски внутренней области — свойство `Brush` 

Таблица П1.29 *(окончание)*

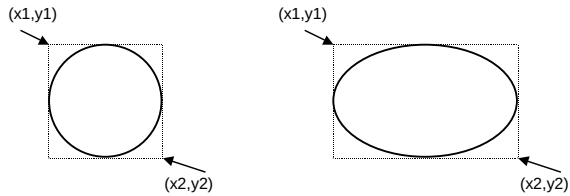
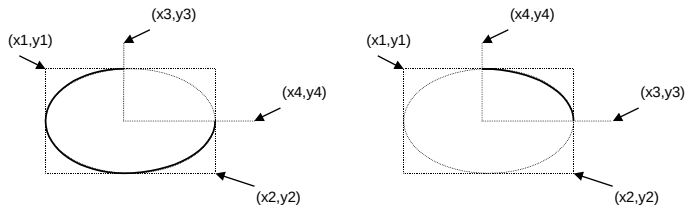
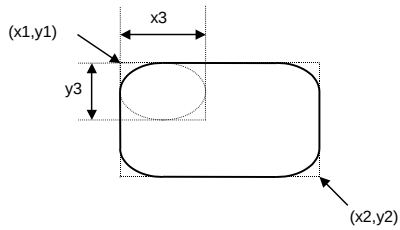
Метод	Описание
`Ellipse(x1,y1, x2,y2)`	Вычерчивает эллипс, окружность или круг. Параметры `x1`, `y1`, `x2` и `y2` задают размер прямоугольника, в который вписывается эллипс. Вид линии определяет свойство `Pen` 
`Arc(x1,y1,x2,y2, x3,y3,x4,y4)`	Вычерчивает дугу. Параметры `x1`, `y1`, `x2`, `y2` определяют эллипс, из которого вырезается дуга, параметры `x2`, `y2`, `x3` и `y4` — координаты концов дуги. Дуга вычерчивается против часовой стрелки от точки (`x3`, `y3`) к точке (`x4`, `y4`). Вид линии (границы) определяет свойство `Pen`, цвет и способ закраски внутренней области — свойство `Brush` 
`RoundRec(x1,y1, x2,y2, x3,y3)`	Вычерчивает прямоугольник со скругленными углами. Параметры `x1`, `y1`, `x2` и `y2` задают координаты левого верхнего и правого нижнего углов, `x3` и `y3` — радиус скругления. Вид линии определяет свойство `Pen`, цвет и способ закраски внутренней области — свойство `Brush` 
`PolyLine(pl)`	Вычерчивает ломаную линию. Координаты точек перегиба задает параметр `pl` — массив структур типа `TPoint`. Если первый и последний элементы массива одинаковые, то будет вычерчен замкнутый контур. Вид линии определяет свойство `Pen`
`Polygon(pl)`	Вычерчивает и закрашивает многоугольник. Координаты углов задает параметр `pl` — массив структур типа `TPoint`. Первый и последний элементы массива должны быть одинаковыми. Вид границы определяет свойство `Pen`, цвет и стиль закраски внутренней области — свойство `Brush`

Таблица П1.30. *Свойства объекта* `TCanvas`

Свойство	Описание
`Pen`	Свойство `Pen` представляет собой объект `TPen` (см. табл. П1.31), свойства которого определяют цвет, толщину и стиль линий, вычерчиваемых методами вывода графических примитивов
`Brush`	Свойство `Brush` представляет собой объект `TBrush` (см. табл. П1.32), свойства которого определяют цвет и стиль закраски областей, вычерчиваемых методами вывода графических примитивов
`Font`	Свойство `Font` представляет собой объект, уточняющие свойства которого определяют шрифт (название, размер, цвет, способ оформления), используемый для вывода на поверхность холста текста
`Transparent`	Признак использования "прозрачного" цвета при выводе битового образа методом `Draw`. Если значение свойства равно `True`, то точки, цвет которых совпадает с цветом левой нижней точки битового образа, не отображаются

Pen

Объект `Pen` (карандаш) является свойством объекта `Canvas`. Свойства объекта `Pen` (табл. П1.31) определяют цвет, стиль и толщину линий, вычерчиваемых методами вывода графических примитивов.

Таблица П1.31. *Свойства объекта* `TPen`

Свойство	Описание
`Color`	Цвет линии (`clBlack` — черный; `clMaroon` — каштановый; `clGreen` — зеленый; `clOlive` — оливковый; `clNavy` — темно-синий; `clPurple` — фиолетовый; `clTeal` — зелено-голубой; `clGray` — серый; `clSilver` — серебристый; `clRed` — красный; `clLime` — салатовый; `clBlue` — синий; `clFuchsia` — ярко-розовый; `clAqua` — бирюзовый; `clWhite` — белый)
`Style`	Стиль (вид) линии: `psSolid` — сплошная; `psDash` — пунктирная (длинные штрихи); `psDot` — пунктирная (короткие штрихи); `psDashDot` — пунктирная (чередование длинного и короткого штрихов); `psDashDotDot` — пунктирная (чередование одного длинного и двух коротких штрихов); `psClear` — не отображается (используется, если не надо изображать границу, например, прямоугольника)
`Width`	Толщина линии задается в пикселах. Толщина пунктирной линии не может быть больше 1

Brush

Объект `Brush` (Кисть) является свойством объекта `Canvas`. Свойства объекта `Brush` (табл. П1.32) определяют цвет, стиль закраски внутренних областей контуров, вычерчиваемых методами вывода графических примитивов.

Таблица П1.32. *Свойства объекта* `TBrush`

Свойство	Определяет
`Color`	Цвет закраски области. (`clBlack` — черный; `clMaroon` — каштановый; `clGreen` — зеленый; `clOlive` — оливковый; `clNavy` — темно-синий; `clPurple` — фиолетовый; `clTeal` — зелено-голубой; `clGray` — серый; `clSilver` — серебристый; `clRed` — красный; `clLime` — салатовый; `clBlue` — синий; `clFuchsia` — ярко-розовый; `clAqua` — бирюзовый; `clWhite` — белый)
`Style`	Стиль (тип) заполнения области: `bsSolid` — сплошная заливка; `bsClear` — область не закрашивается; `bsHorizontal` — горизонтальная штриховка; `bsVertical` — вертикальная штриховка; `bsFDiagonal` — диагональная штриховка с наклоном линий вперед; `bsBDiagonal` — диагональная штриховка с наклоном линий назад; `bsCross` — горизонтально-вертикальная штриховка, в клетку; `bsDiagCross` — диагональная штриховка, в клетку

Цвет

Цвет линии (свойство `Pen.Color`) или цвет закраски области (свойство `Brush.Color`) можно задать, указав в качестве значения свойства именованную константу типа `TColor` (табл. П1.33). Вместо конкретного цвета можно указать цвет элемента графического интерфейса, тем самым "привязать" цвет к цветовой схеме операционной системы.

Таблица П1.33. *Константы* `TColor`

Константа	Цвет	Константа	Цвет
`clBlack`	Черный	`clSilver`	Серебристый
`clMaroon`	Каштановый	`clRed`	Красный
`clGreen`	Зеленый	`clLime`	Салатовый
`clOlive`	Оливковый	`clBlue`	Синий
`clNavy`	Темно-синий	`clFuchsia`	Ярко-розовый
`clPurple`	Фиолетовый	`clAqua`	Бирюзовый
`clTeal`	Зелено-голубой	`clWhite`	Белый
`clGray`	Серый		

Функции

В этом разделе приведено краткое описание часто используемых функций (табл. П1.34—П1.37). Подробное их описание можно найти в справочной системе.

Функции ввода и вывода

Таблица П1.34. *Функции ввода и вывода*

Функция	Описание

`InputBox(`*`Заголовок`*`,`
`  `*`Подсказка`*`,`
`  `*`Значение`*`)`
	В результате выполнения функции на экране появляется диалоговое окно, в поле которого пользователь может ввести строку символов. Значением функции является введенная строка. Параметр *`Значение`* задает значение функции "по умолчанию", т. е. строку, которая будет в поле редактирования в момент появления окна

`ShowMessage(s)`	Процедура `ShowMessage` выводит окно, в котором находится сообщение `s` и командная кнопка **OK**
`MessageDlg(s,t,b,h)`	Выводит на экран диалоговое окно с сообщением `s` и возвращает код кнопки, щелчком на которой пользователь закрыл окно. Параметр `t` определяет тип окна: `mtWarning` — внимание; `mtError` — ошибка; `mtInformation` — информация; `mtConfirmation` — запрос; `mtCustom` — пользовательское (без значка). Параметр `b` (множество — заключенный в квадратные скобки список констант) задает командные кнопки диалогового окна (`mbYes`, `mbNo`, `mbOK`, `mbCancel`, `mbHelp`, `mbAbort`, `mbRetry`, `mbIgnore` и `mbAll`). Параметр `h` задает раздел справочной системы программы, который появится в результате нажатия кнопки **Help** или клавиши <F1>. Если справочная система не используется, значение параметра должно быть равно `0`. Значением функции может быть одна из констант: `mrAbort`, `mrYes`, `mrOk`, `mrRetry`, `mrNo`, `mrCancel`, `mrIgnore` или `mrAll`, обозначающая соответствующую командную кнопку

Математические функции

Таблица П1.35. *Математические функции*

Функция	Значение
`abs(n)`	Абсолютное значение `n`
`sqrt(n)`	Квадратный корень из `n`
`exp(n)`	Экспонента `n`

Таблица П1.35 *(окончание)*

Функция	Значение
`random(n)`	Случайное целое число в диапазоне от 0 до `n-1` (перед первым обращением к функции необходимо вызвать процедуру `randomize`, которая выполнит инициализацию программного генератора случайных чисел)
`sin(α)`	Синус выраженного в радианах угла α
`cos(α)`	Косинус выраженного в радианах угла α
`tan(α)`	Тангенс выраженного в радианах угла α
`asin(n)`, `acos(n)`, `atan(n)`	Угол (в радианах), синус, косинус и тангенс которого равен `n`

Аргумент тригонометрических функций (угол) должен быть выражен в радианах. Для преобразования величины угла из градусов в радианы следует воспользоваться формулой `(a*3.1415256)/180`, где `a` — величина угла в градусах; `3.1415926` — число π. Вместо десятичной константы `3.1415926` можно использовать стандартную именованную константу `PI`.

Функции преобразования

Таблица П1.36. *Функции преобразования*

Функция	Значение функции
`IntToStr(k)`	Строка, являющаяся изображением целого `k`
`FloatToStr(n)`	Строка, являющаяся изображением вещественного `n`
`FloatToStrF(n,f,k,m)`	Строка, являющаяся изображением вещественного `n`. При вызове функции указывают: `f` — формат; `k` — точность; `m` — количество цифр после десятичной точки. Формат определяет способ изображения числа: `ffGeneral` — универсальный; `ffExponent` — научный; `ffFixed` — с фиксированной точкой; `ffNumber` — с разделителями групп разрядов; `ffCurrency` — финансовый. Точность — нужное общее количество цифр: 7 или меньше для значения типа `Single`, 15 или меньше для значения типа `Double` и 18 или меньше для значения типа `Extended`
`StrToInt(s)`	Целое, изображением которого является строка `s`
`StrToFloat(s)`	Вещественное, изображением которого является строка `s`

Функции манипулирования датами и временем

Большинству функций манипулирования датами в качестве параметра передается переменная типа `TDateTime`, которая хранит информацию о дате и времени.

Для того чтобы в программе были доступны функции `DayOf`, `WeekOf`, `MonthOf` и некоторые другие, например `CompareTime`, в ее текст надо включить ссылку на модуль `DateUtils` (указать имя модуля в директиве `uses`).

Таблица П1.37. *Функции манипулирования датами и временем*

Функция	Значение
`Now()`	Системная дата и время — значение типа `TDateTime`
`DateToStr(dt)`	Строка символов, изображающая дату в формате `dd.mm.yyyy`
`TimeToStr(dt)`	Строка символов, изображающая время в формате `hh:mm:ss`
`FormatDateTime(s,dt)`	Строка символов, представляющая собой дату или время. Способ представления задает строка `s`. Например, строка `dd.mm.yyyy` задает, что значением функции является дата, а строка `hh:mm` — время. Помимо символов формата, в строке `s` могут быть и другие символы, например, если необходимо отобразить текущее время, то строка форматирования может выглядеть так: `Сейчас hh:mm`. Форматы: `d` — число, одна или две цифры; `dd` — число, две цифры; `ddd` — сокращенное название дня недели; `dddd` — полное название дня недели; `m` — месяц, одна или две цифры; `mm` — месяц, две цифры; `mmm` — сокращенное название месяца; `mmmm` — полное название месяца
`DayOf(dt)`	День (номер дня в месяце), соответствующий дате, указанной в качестве параметра функции
`MonthOf(dt)`	Номер месяца, соответствующий дате, указанной в качестве параметра функции
`WeekOf(dt)`	Номер недели, соответствующий дате, указанной в качестве параметра функции
`YearOf(dt)`	Год, соответствующий указанной дате
`DayOfWeek(dt)`	Номер дня недели, соответствующий указанной дате: 1 — воскресенье, 2 — понедельник, 3 — вторник и т. д.
`StartOfWeek(w)`	Дата первого дня указанной недели (`w` — номер недели, отсчитываемый от начала года)

Таблица П1.37 *(окончание)*

Функция	Значение
`HourOf(dt)`	Количество часов
`MinuteOf(dt)`	Количество минут
`SecondOf(dt)`	Количество секунд
`DecodeDate(dt,y,m,d)`	Возвращает год, месяц и день, представленные отдельными числами
`DecodeTime(dt,h,m,s,ms)`	Возвращает время (часы, минуты, секунды и миллисекунды), представленное отдельными числами
`EncodeDate(y,m,d)`	Значение типа `TDateTime`, соответствующее указанной дате (`y` — год, `m` — месяц, `d` — день)
`EncodeTime(h,m,s,ms)`	Значение типа `TDateTime`, соответствующее указанному времени (`h` — часы; `m` — минуты; `s` — секунды; `ms` — миллисекунды)
`CompareDate(dt1,dt2)`	Сравнивает две даты (значения типа `TDateTime`). Если даты совпадают, значение функции равно нулю, если `dt1 < dt2` (например, `dt1 = 01.06.2006`, а `dt2 = 05.06.2006`), то значение функции — "минус один", если `dt2 > dt1`, то значение функции — "единица"
`CompareTime(dt1,dt2)`	Сравнивает два временных значения (значения типа `TDateTime`). Если времена совпадают, значение функции равно нулю, если `dt1 < dt2` (например, `dt1` = 10:00, а `dt2` = 10:30), то значение функции — "минус один", если `dt2 > dt1`, то значение функции — "единица"

События

Таблица П1.38. *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

Таблица П1.38 *(окончание)*

Событие	Происходит
KeyDown	При нажатии клавиши клавиатуры. События OnKeyDown и OnKeyPress — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие OnKeyUp)
KeyUp	При отпускании нажатой клавиши клавиатуры
Create	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
Paint	При появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном и в других случаях. Событие сообщает о необходимости обновить (перерисовать) окно
Enter	При получении элементом управления фокуса
Exit	При потере элементом управления фокуса

Исключения

Таблица П1.39. *Типичные исключения*

Тип исключения	Возникает
EConvertError	При выполнении преобразования строки в число, если преобразуемая величина не может быть приведена к требуемому виду. Наиболее часто возникает при преобразовании строки в дробное число, если в качестве разделителя целой и дробной частей указан неверный символ
EZeroDivide	Деление на ноль. При выполнении операции деления, если делитель равен нулю (если и делитель, и делимое равны нулю, то возникает исключение EInvalidOp)
EFOpenError	При обращении к файлу, например при попытке загрузить файл иллюстрации с помощью метода LoadFromFile. Наиболее частой причиной является отсутствие требуемого файла или, в случае использования сменного диска, отсутствие диска в накопителе
EInOutError	При обращении к файлу, например при попытке открыть для чтения несуществующий файл
EOLEException	При выполнении операций с базой данных, например при попытке открыть несуществующую базу данных, если для доступа к базе данных используются ADO-компоненты (чтобы иметь возможность обработки этого исключения, в директиву uses надо добавить ссылку на модуль ComObj)

Приложение 2

Содержимое компакт-диска

Прилагаемый к книге компакт-диск содержит проекты, приведенные в книге. Каждый проект (табл. П2.1) находится в отдельном каталоге. Помимо файлов, образующих проект, в каталоге проекта есть exe-файл, что позволяет (при условии, что среда Delphi XE установлена на компьютере) увидеть, как работает программа, без загрузки проекта в Delphi. Программы работы с базами данных требуют, чтобы файлы баз данных находились в каталоге D:\Database.

Для активной работы с примерами, чтобы иметь возможность вносить в них изменения, необходимо скопировать каталоги проектов на жесткий диск компьютера.

Таблица П2.1. *Проекты*

Проект (каталог)	Краткое описание	Глава
Animate	Демонстрирует использование компонента `Animate` для воспроизведения простой, не сопровождаемой звуком анимации	5
Blackfish_books	Программа работы с базой данных Blackfish SQL "Книги" (books). Подкаталог delploy\database содержит базу данных, подкаталоги deploy\client и deploy\server — соответственно клиентскую и серверную части приложения	6
Button	Пересчет расстояния из верст в километры. Программа демонстрирует использование компонента `Button`	3
CDp	Полнофункциональный проигрыватель компакт-дисков. Демонстрирует использование компонента `MediaPlayer`	5
CheckBox	Расчет цены автомобиля в зависимости от выбранной комплектации. Программа демонстрирует использование компонента `CheckBox`	3
ComboBox	Расчет цены жалюзи. Программа демонстрирует использование компонента `ComboBox`	3

Таблица П2.1 *(продолжение)*

Проект (каталог)	Краткое описание	Глава
Converter	Пересчет цены из долларов в рубли. Демонстрирует фильтрацию символов, вводимых в поле редактирования, вывод информации в поле `Label` в различных форматах (общий, финансовый)	2
Database	Содержит файлы, необходимые для работы программ БД_контакты_1 — БД_контакты_3	
Edit	Демонстрирует использование компонента `TextBox` для ввода данных различного типа	3
Image	Просмотр иллюстраций. Программа демонстрирует использование компонентов `Image` и `SpeedButton`	3
Label	Доход по вкладу. Демонстрирует использование компонента `Label`: вывод информации в поле компонента, изменение цвета отображаемого текста	3
LineTo	Демонстрирует различные стили рисования линий	4
ListBox	Просмотр иллюстраций. Демонстрирует использование компонентов `ListBox`. Программа формирует список jpg-файлов, которые находятся в выбранном пользователем каталоге, и отображает его в поле компонента `ListBox`. Выбор каталога осуществляется в стандартном окне **Обзор папок**	3
MainMenu	MEdit — простой редактор теста. Демонстрирует использование компонентов `MainMenu`, `OpenDialog`, `SaveDialog`, загрузку текста из файла в компонент `Memo`	3
Midi	Игра "Угадай число". Демонстрирует использование компонента `MediaPlayer` для воспроизведения MIDI-музыки. Мелодия воспроизводится "по кругу" до тех пор, пока пользователь не угадает число или не истечет время, отведенное на решение задачи	5
MP3 Player	MP3-плеер. Демонстрирует использование компонентов `MediaPlayer`, `TrackBar`, `SpeedButton`, а также регулировку громкости из окна программы и перемещение окна, у которого нет заголовка	10
NkCtrls	Компонент программиста. Компонент `NkEdit` предназначен для ввода числовой информации (в зависимости от настройки — целого или дробного числа). В каталоге находятся модуль компонента (NkCtrls.pas), программа тестирования модуля компонента (проект TestNkEdit.dproj) и проект создания пакета компонентов (проект NkPackage.dproj)	7
NkEditTest	Программа проверки работоспособности пакета NkPackage (компонента NkEdit)	7

Таблица П2.1 *(продолжение)*

Проект (каталог)	Краткое описание	Глава
NPV	Вычисление чистого приведенного дохода. Демонстрирует различные способы отображения справочной информации в формате HTML Help, а также обработку одной процедурой событий от нескольких компонентов	8
OpenDialog	Программа позволяет просмотреть содержимое txt-файла. Демонстрирует использование компонента `OpenDialog`	3
Polygon	Демонстрирует использование метода `Polygon` — рисует полигон (график) изменения курса доллара. Данные загружаются из файла	4
ProgressBar	Игра "Угадай число". Демонстрирует использование компонента `ProgressBar`	3
RadioButton	Расчет суммы заказа печати фотографий. Демонстрирует использование компонента `RadioButton`	3
Rectangle	Демонстрирует использование метода `Rectangle`	4
Saper	Игра "Сапер". Демонстрирует работу с графикой, вывод окна **О программе**, отображение справочной информации в формате HTML Help. В каталоге CHM находятся файлы проекта создания справочной системы	10
SMP3	Простой MP3-плеер. Демонстрирует использование компонентов `MediaPlayer`	5
StatusBar	Игра "Угадай число". Демонстрирует использование компонента `StatusBar`	3
TextOut	Демонстрирует использование метода `TextOut` для вывода текста на графическую поверхность, показывает, как разместить текст в центре окна	4
Timer	Секундомер. Демонстрирует использование компонента `Timer`	3
UpDown	Программа "Будильник". Демонстрирует использование компонента `UpDown`, а также воспроизведение WAV-звука с помощью функции `PlaySound`	3
Video	Видеоплеер. Демонстрирует использование компонента `MediaPlayer` для воспроизведения небольших видеороликов в AVI-формате	4
Vista\myEdit	Программа для Windows Vista "Редактор текста". Демонстрирует использование компонентов `FileOpenDialog`, `FileSaveDialog` и `TaskDialog`	3

Таблица П2.1 *(продолжение)*

Проект (каталог)	Краткое описание	Глава
Vista\Доход	Программа для Windows Vista "Доход". Демонстрирует использование компонента `TaskDialog`	3
БД_контакты_1	Программа работы с базой данных Microsoft Access "Контакты". Позволяет просматривать, редактировать, добавлять и удалять записи. Демонстрирует использование компонентов `ADOConnection`, `ADODataSet`, `DataSource` и `DataGrid`. Файл базы данных (contacts.mdb) должен находиться в каталоге D:\Database	6
БД_контакты_2	Программа работы с базой данных Microsoft Access "Контакты". Позволяет просматривать, редактировать, добавлять и удалять записи, а также обеспечивает выборку информации содержимому одного из полей. Демонстрирует использование компонентов `ADOConnection`, `ADODataSet`, `DataSource` и `DataGrid`, возможность фильтрации данных. Файл базы данных (contacts.mdb) должен находиться в каталоге D:\Database	6
БД_контакты_3	Программа работы с базой данных Microsoft Access "Контакты" в режиме формы. Демонстрирует использование компонентов `ADOConnection`, `ADODataSet`, `DataSource`, `DBEdit`, `DBMemo` и `DBNavigator`. Файл базы данных (contacts.mdb) должен находиться в каталоге D:\Database, файлы иллюстраций — в каталоге D:\Database\Images	6
Битовый образ	Демонстрирует использование битовых образов для формирования сложных изображений. Битовые образы, из которых формируется картинка, загружаются из файлов	4
График	Строит график изменения курса доллара. Демонстрирует использование методов, обеспечивающих формирование графики на поверхности формы (`LineTo`, `TextOut`, `Ellipse`). Данные загружаются из файла	4
Диаграмма	Строит круговую диаграмму "Источники энергии". Демонстрирует использование методов, обеспечивающих формирование графики на поверхности формы (`Pie`, `Rectangle`, `TextOut`). Данные загружаются из файла	4
Корабль	Демонстрирует принципы реализации мультипликации. Изображение объекта формируется из графических примитивов (элементов)	4
Обработка исключения	Пересчет цены из долларов в рубли. Демонстрирует использование инструкции `try` для обработки исключения `EConvertError`	2

Таблица П2.1 *(окончание)*

Проект (каталог)	**Краткое описание**	**Глава**
ПВО	Игра "ПВО". Демонстрирует принцип реализации интерактивной графики	4
Полет в облаках	Демонстрирует принципы реализации мультипликации. Фрагмент картинки (кадр) формируется на невидимой графической поверхности и затем выводится в нужную точку формы. Изображения фона и объекта загружаются из файлов	4
Фоновый рисунок	Формирует фоновый рисунок, путем многократного вывода битового образа на поверхность формы	4
Экзаменатор	Экзаменатор — универсальная программа тестирования. Файл теста надо указать в командной строке запуска программы	10

Предметный указатель

L

M

N

O

P

R

S

T

U, W, Y

А

Б

В

Д

З

И

К

Л

М

О

П

Р

С

Т

Ф

Ц

Ш

Э