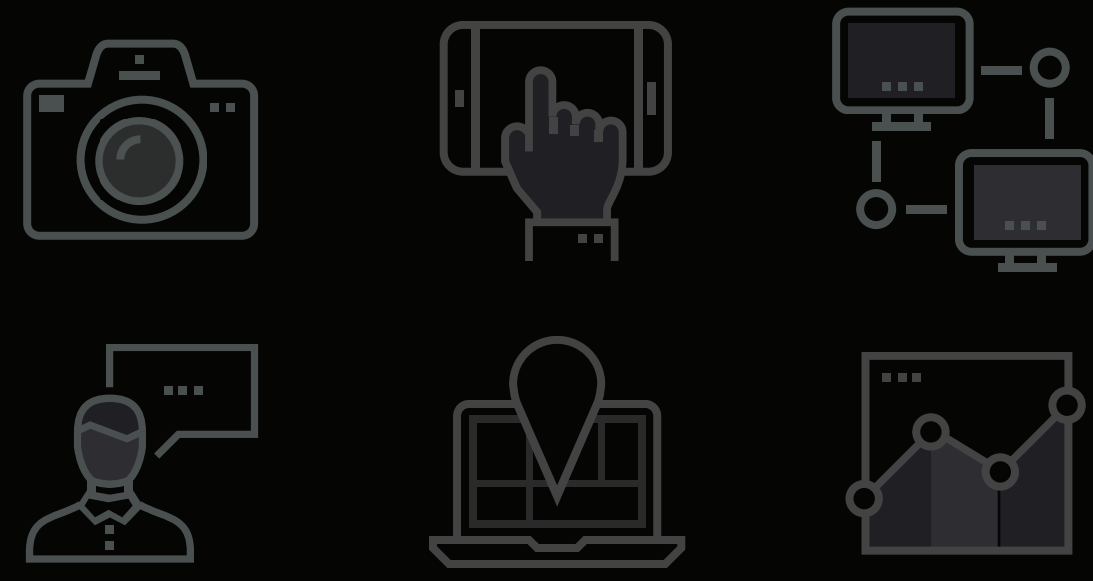
ОСНОВЫ ПРОГРАММИРОВАНИЯ В DELPHI

НИКИТА КУЛЬТИН



Никита Культин

Основы программирования в Embarcadero Delphi

Книга представляет собой практическое руководство по программированию в Embarcadero Delphi. В ней представлена технология визуального проектирования и событийного программирования, подробно рассмотрен процесс создания VCL приложений, показано назначение базовых компонентов, рассмотрены вопросы программирования графики, работа с базами данных Microsoft Access. Уделено внимание разработке Multi-Device приложений, в том числе для платформы Android, использованию компонентов FireMonkey, использованию компонентов FireDAC для работы с интегрированными базами данных SQLite, 3D графика, Float и Path анимация, работа с датчиками и сенсорами.

Для начинающих программистов

Содержание

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 этой технологии заключается в том, что среда разработки берет на себя большую часть рутины, оставляя программисту работу по созданию интерфейса при помощи специализированного графического редактора (визуальное проектирование) и процедур обработки событий (событийное программирование).

Изначально Delphi была ориентирована на разработку приложений для Windows. Теперь в Delphi можно создавать приложения как для Windows, так и для Mac, iOS и Android. Благодаря поддержке кроссплатформенности, программа, написанная для одной платформы легко может быть развернута на другую платформу.

В настоящий момент программистам стала доступна очередная версия Delphi — Embarcadero Delphi XE7. Она существует в трех вариантах: Professional, Enterprise и Architect. Каждый комплект состоит из набора средств и компонентов, обеспечивающих разработку высокоэффективных приложений различного назначения. Чем выше уровень пакета, тем больше возможностей он предоставляет программисту.

Embarcadero Delphi XE7 может работать в среде операционных систем Microsoft Windows 7 (Home или Professional) и в Microsoft Windows 8. Особых требований, по современным меркам, к ресурсам компьютера среда не предъявляет.

Книга, которую вы читаете — это не описание среды разработки или языка программирования, это практическое руководство по программированию в Embarcadero Delphi XE7. В ней представлена технология визуального проектирования и событийного программирования, подробно рассмотрен процесс создания приложений, приведено описание и показано назначение базовых компонентов, рассмотрены вопросы программирования графики, работе с базами данных. Уделено внимание кроссплатформенной разработке на основе FireMonkey, использованию компонентов FireDAC для работы с базами данных, работе с датчиками.

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

Научиться программировать можно только программируя, решая конкретные задачи. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте листинги, вводите программы в компьютер, вносите в них изменения, экспериментируйте. Чем больше вы сделаете самостоятельно, тем большему научитесь!

Глава 1. Первый проект

Процесс разработки *приложения* (от англ. application – прикладная программа) рассмотрим на примере программы расчета стоимости покупки, состоящей из нескольких одинаковых предметов. Вид окна программы после ввода исходных данных в поля редактирования и нажатия кнопки Расчет приведен на рис. 1.1.

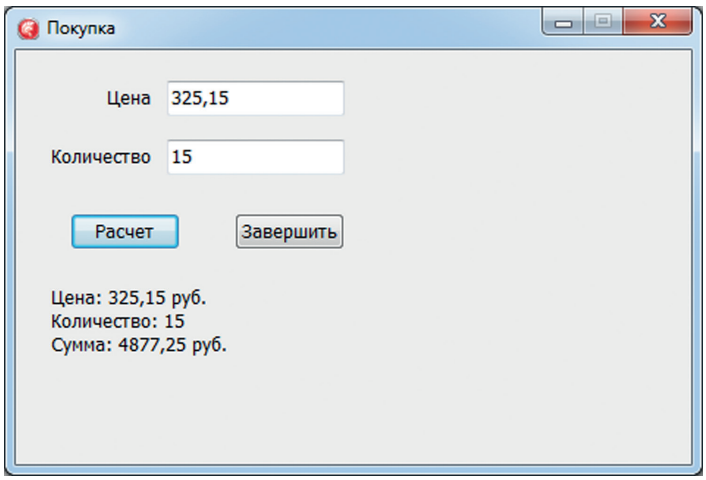


Рисунок 1.1. Окно программы Покупка

Начало работы над новым приложением

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

Вид окна Delphi в начале работы над новым проектом приведен на рис. 1.2.

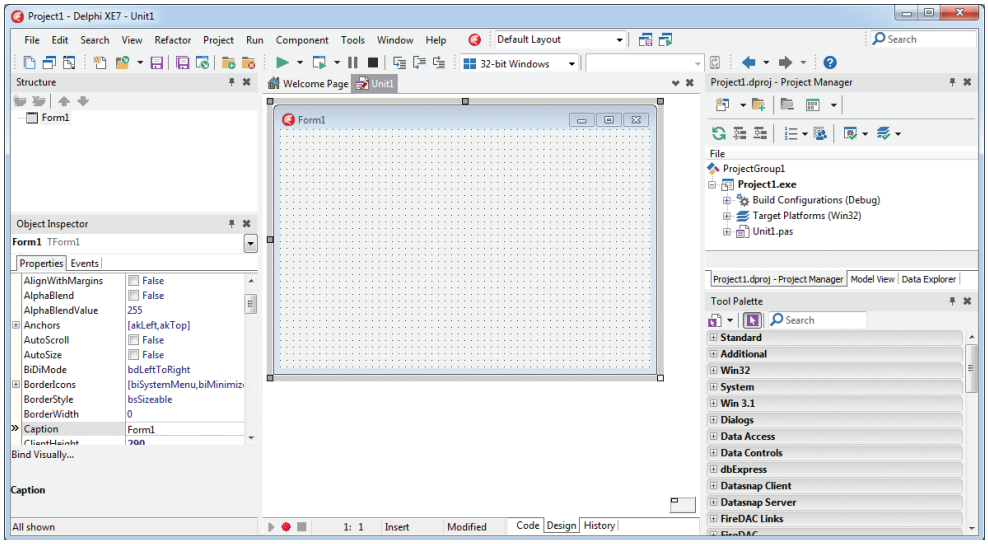


Рисунок 1.2. Окно Delphi в начале работы над новым проектом

Центральную часть окна занимает окно конструктора формы (Design). В нем находится *форма* — заготовка окна разрабатываемого приложения.

За окном конструктора формы располагается окно редактора текста программы или *кода* (рис. 1.3), доступ к нему можно получить, если сделать щелчок кнопкой мыши на ярлыке **Code** или нажать клавишу F12. В начале работы над новой программой в окне редактора кода находится, сформированный средой разработки, шаблон (заготовка) приложения.

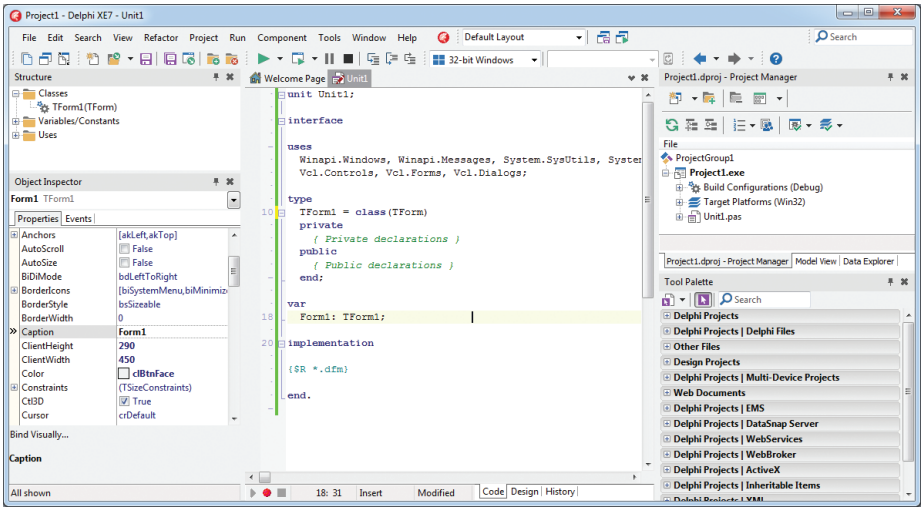


Рисунок 1.3. В редакторе кода отображается текст программы

В нижней левой части окна Delphi находится окно **Object** Inspector. На вкладке **Properties** этого окна отображаются *свойства* (properties) выбранного в данный момент *объекта* (рис. 1.4).

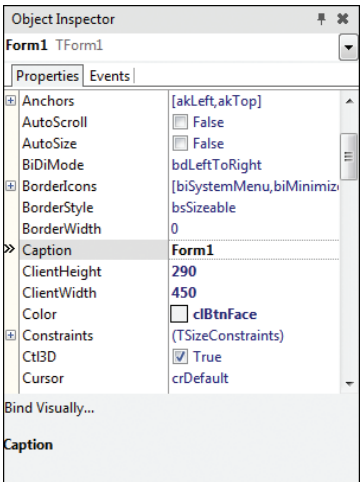


Рисунок 1.4. Вкладке Properties окна Object Inspector

В начале работы над проектом на вкладке **Properties** окна **Object** Inspector отображаются свойства формы приложения – объекта Form1.

Свойства определяют вид объекта, его положение на экране или на поверхности формы. Например, свойство Caption формы определяет текст заголовка формы, свойства Width и Height — размеры формы, а свойство Position — положение формы перед началом работы программы, сразу после ее запуска. Справа от названий свойств указаны их значения.

В правой верхней части окна Delphi располагается окно **Project Manager** (рис. 1.5). В нем отображается структура проекта, над которым в данный момент идет работа.

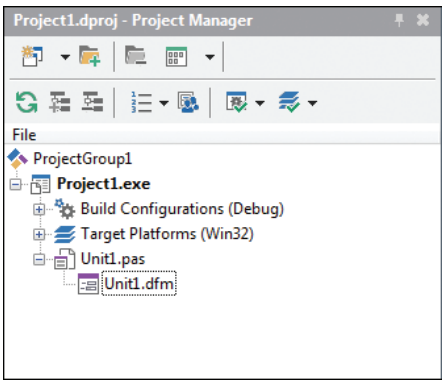


Рисунок 1.5. Окно Project Manager

А в правой нижней части окна Delphi воспроизводится окно **Tool Palette** (рис. 1.6) — палитра компонентов. На вкладках палитры компонентов находятся *компоненты* — элементы пользовательского интерфейса и другие объекты, реализующие 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, обеспечивающие взаимодействие с базами данных.

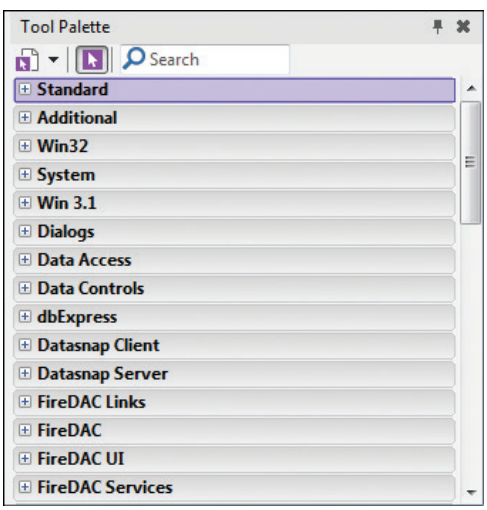


Рисунок 1.6. Окно Tool Palette - палитра компонентов

Форма

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

Настройка формы (впрочем, как и других компонентов) осуществляется путем изменения значений *свойств*.

Основные свойства формы (объекта TForm) приведены в табл. 1.1.

Таблица 1.1. 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 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
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 и его тип (класс) отображается в верхней части окна **Object Inspector**

Чтобы в заголовке формы вместо текста Form1 появилось название программы, в нашем случае — текст Покупка, надо изменить значение свойства Caption. Для этого нужно на вкладке **Properties** окна **Object Inspector** щелкнуть левой кнопкой мыши в строке свойства Caption (в результате будет выделено текущее значение свойства и появится курсор), ввести текст Покупка и нажать клавишу Enter (рис. 1.7).

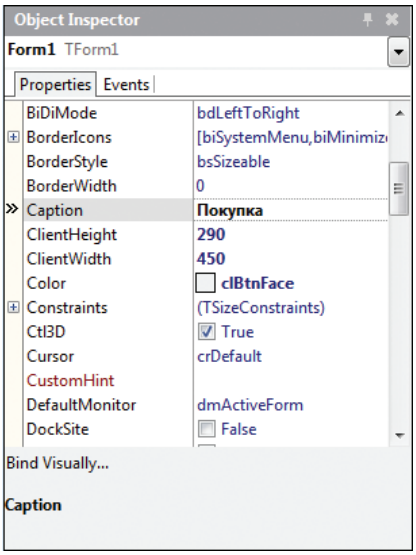


Рисунок 1.7. Изменение значения свойства Caption путем ввода нового значения

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 310 и 466, соответственно. Следует обратить внимание, размер формы и других компонентов можно изменить и при помощи мыши, путем перемещения границы объекта. По окончании перемещения границы значения свойств Height и Width будут соответствовать установленному размеру формы.

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. 1.8).

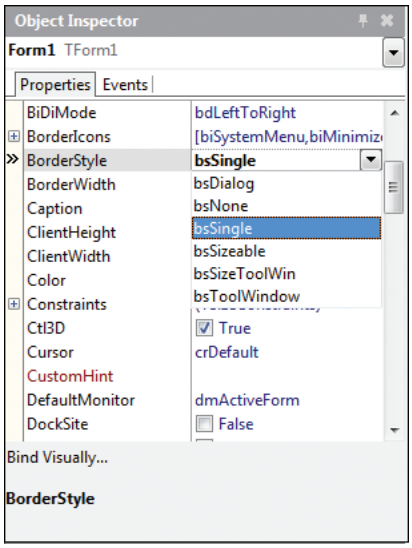


Рисунок 1.8. Установка значения свойства путем выбора значения из списка

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 определяется 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 отображается значок «+», в результате щелчка на котором раскрывается список уточняющих свойств (рис. 1.9). Значение уточняющего свойства можно задать обычным образом (ввести значение в поле редактирования или выбрать в списке).

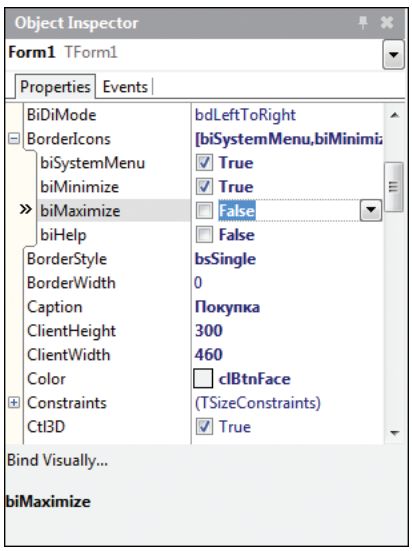


Рисунок 1.9. Изменение значения уточняющего свойства

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 окном **Шрифт**, которое появится в результате щелчка на кнопке с тремя точками (рис. 1.10).

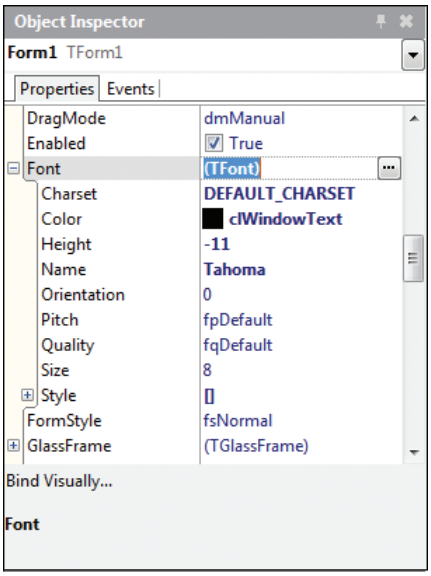


Рисунок 1.10. Чтобы задать свойства шрифта, щелкните на кнопке с тремя точками

В табл. 1.2 приведены значения свойств формы программы Покупка, которые были изменены в процессе настройки формы. 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.

Таблица 1.2. Значения свойств стартовой формы

Свойство	Значение	Комментарий
Caption	Покупка	
Height	310	
Width	466	
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 центе 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. 1.11.

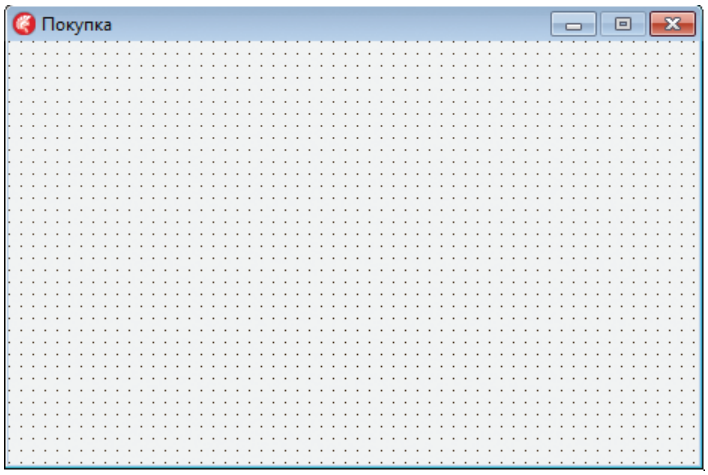


Рисунок 1.11. Форма после изменения значений ее свойств

Теперь на форму надо добавить *компоненты*.

Компоненты

Поля ввода-редактирования текста, поля отображения текста, кнопки, списки, переключатели и другие элементы, обеспечивающие взаимодействие пользователя с программой (или программы с пользователем, кому как больше нравится), называют *компонентами пользовательского интерфейса* или просто *компонентами*.

Компоненты находятся в палитре компонентов (окно **Tool Palette**).

Программа Покупка для расчета стоимости покупки должна получить от пользователя цену предметов (предполагается, что покупка состоит из предметов одного наименования) и их количество. Для ввода данных с клавиатуры предназначен компонент Edit. Поэтому на форму разрабатываемого приложения нужно поместить два компонента Edit.

Чтобы добавить компонент Edit на форму, надо:

1. В палитре компонентов (окно **Tool Palette**) раскрыть вкладку **Standard**.
2. Сделать щелчок на значке компонента Edit (рис. 1.12). Следует обратить внимание, что в палитре компонентов, рядом со значком указывается тип компонента, а не его название.
3. Установить указатель мыши примерно в ту точку формы, в которой должен быть левый верхний угол компонента.
4. Сделать щелчок левой кнопкой мыши.

В результате выполнения описанных действий на форме появляется поле редактирования — компонент Edit (рис. 1.13).

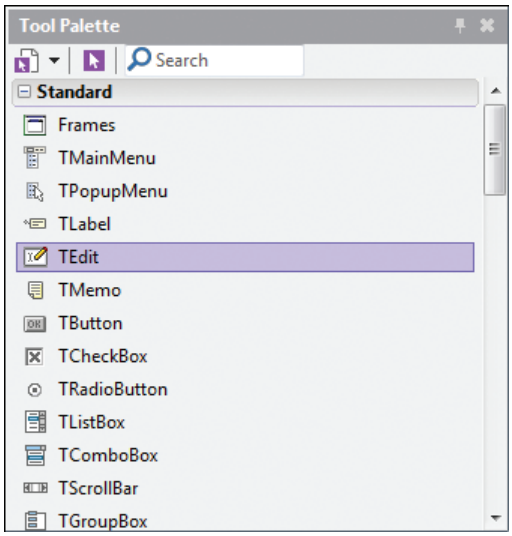


Рисунок 1.12. Компонент Edit — поле редактирования

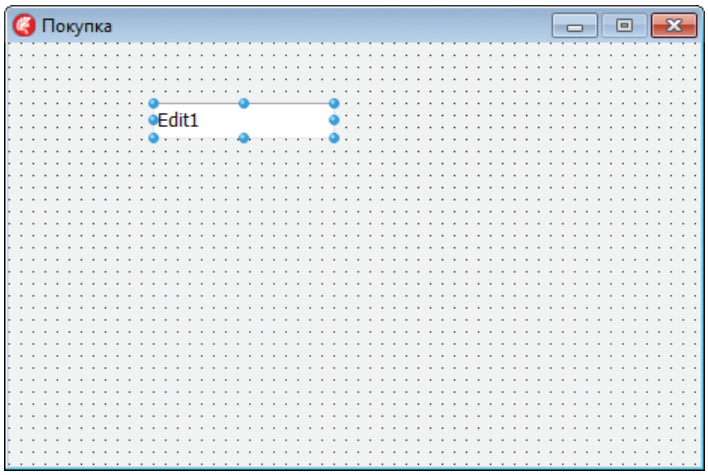


Рисунок 1.13. Результат добавления на форму компонента Edit

Каждому добавленному на форму компоненту, среда разработки присваивает имя, которое формируется из названия компонента и его порядкового номера. Например, первый компонент Edit, получает имя Edit1, второй — Edit2.

Основные свойства компонента Edit приведены в табл. 1.3.

Таблица 1.3. Свойства компонента Edit (объект типа TEdit)

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту
Text	Текст, находящийся в поле редактирования
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы

Свойство	Описание
Height	Высота компонента
Width	Ширина компонента
Font	Шрифт, используемый для отображения текста в поле компонента
ParentFont	Признак наследования шрифта от формы. Если значения 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. 1.14 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 имя нужного компонента в раскрывающемся списке, который находится в верхней части окна **Object Inspector**.

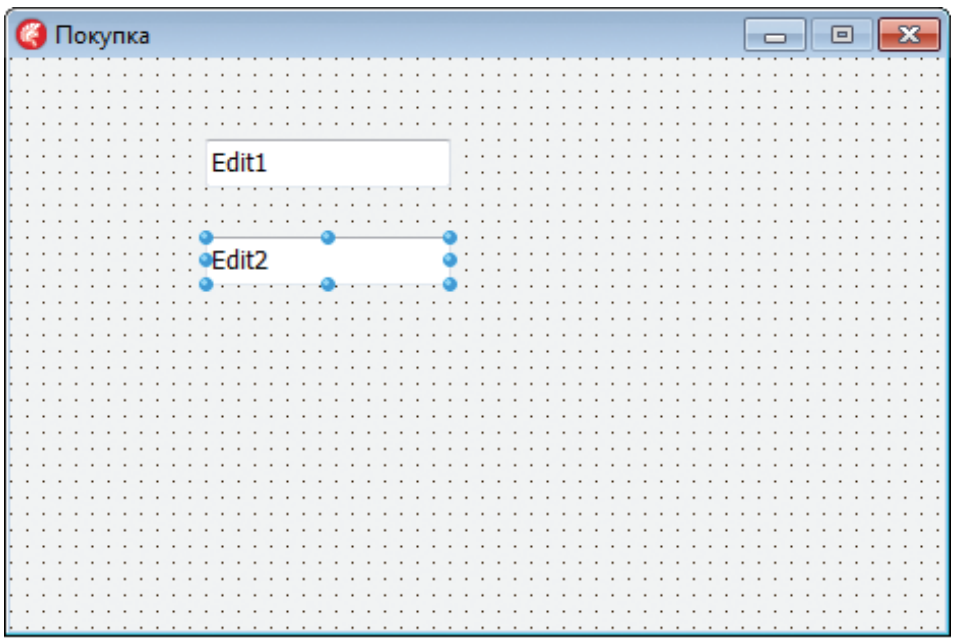


Рисунок 1.14. Форма с двумя компонентами

Значения свойств компонента, 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.

Для того чтобы изменить размер компонента, необходимо сделать щелчок на изображении

компонента (в результате этого компонент будет выбран), установить указатель мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить границу компонента в нужном направлении.

В табл. 1.4 приведены значения свойств компонентов Edit1 и Edit2 (пустое поле в троке свойства Text показывает, что значением свойства является пустая строка). Значения остальных свойств компонентов Edit оставлены без изменения, и поэтому в таблице не показаны.

Таблица 1.4. Значения свойств компонентов Edit

Компонент	Свойство	Значение
Edit1	Top	21
	Left	103
	Text	
	TabOder	0
Edit2	Top	61
	Left	103
	Text	
	TabOder	1

Обратите внимание, значения свойства Font компонентов Edit не указаны явно, поэтому текст в полях редактирования отображается шрифтом, заданным свойством Font 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 нужно, 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.

Чтобы пользователь знал, какую информацию надо вводить в поля редактирования, слева от каждого поля редактирования следует поместить текст, поясняющий назначение поля.

Отображение текста на поверхности формы обеспечивает компонент Label (рис. 1.15).

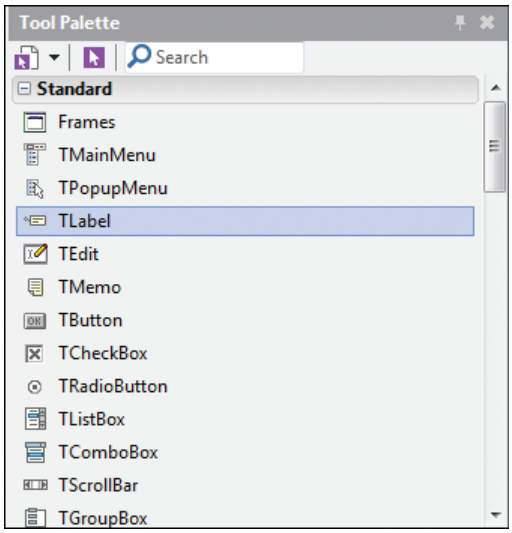


Рисунок 1.15. Компонент Label — поле отображения текста

В рассматриваемой программе два поля редактирования, поэтому в форму надо добавить два компонента Label для отображения пояснительного текста о назначении полей ввода и еще один компонент Label для отображения результата расчета.

Добавляются компоненты Label на форму точно так же, как и поля редактирования.

Основные свойства компонента Label перечислены в табл. 1.5.

Таблица 1.5. Свойства компонента Label

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

Следует обратить внимание, если поле Label должно содержать несколько строк текста, то перед тем как изменить значение свойства Caption, сначала надо свойству AutoSize

присвоить значение False, а свойству WordWrap — True. Затем нужно установить требуемый размер поля (с помощью мыши или вводом значений свойств Width и Height) и только после этого ввести значение свойства Caption.

Как было сказано раньше, на форму разрабатываемого приложения надо добавить три компонента Label. В полях Label1 и Label2 отображается информации о назначении полей ввода, поле Label3 — используется для вывода результата расчета. Значения свойств компонентов Label приведены в табл. 1.6.

Таблица 1.6. Значения свойств компонентов Label

Компонент	Свойство	Значение
Label1	Left	62
	Top	24
	Caption	Цена
Label2	Left	24
	Top	64
	Caption	Количество
Label3	Left	24
	Top	160
	AutoSize	False
	Width	361
	Height	49
	Caption	

После настройки компонентов Label форма разрабатываемого приложения должна выглядеть так, как показано на рис. 1.16.

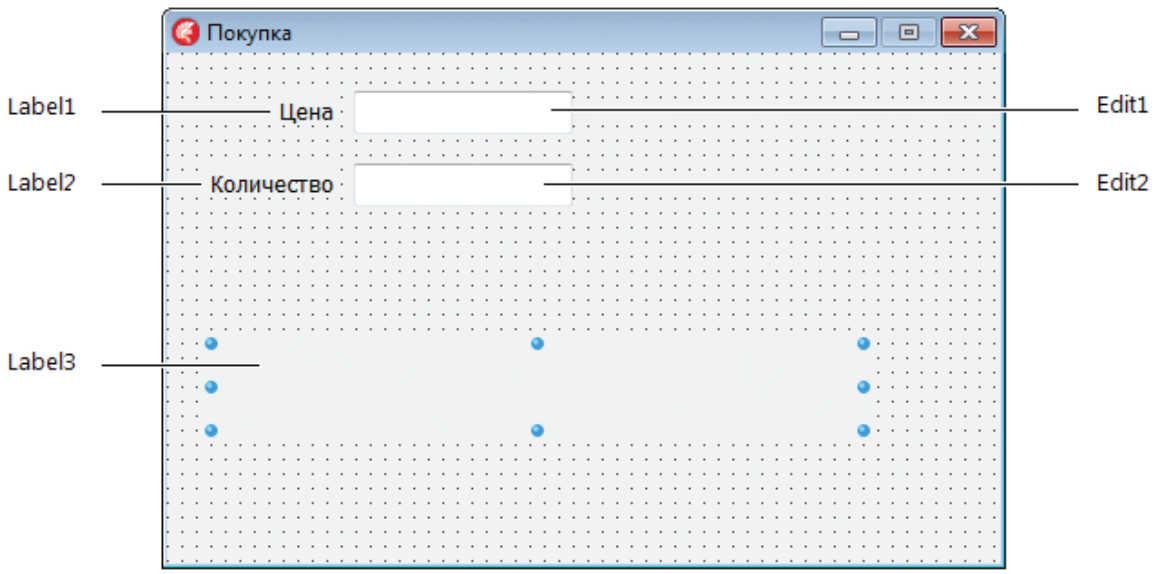


Рисунок 1.16. Вид формы после настройки полей отображения текста

Последнее, что надо сделать на этапе создания формы — добавить на форму две кнопки: **Расчет** и **Завершить**. Кнопка это — компонент Button, который находится на вкладке **Standard** (рис. 1.17).

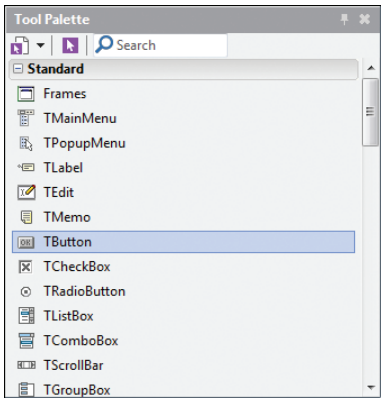


Рисунок 1.17. Командная кнопка — компонент Button

Основные свойства компонента Button приведены в табл. 1.7.

Таблица 1.7. 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
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

После того как на форму будут добавлены кнопки, нужно выполнить их настройку. Значения свойств компонентов Button приведены в табл. 1.8, окончательный вид формы показан на рис. 1.18.

Таблица 1.8. Значения свойств компонентов Button

Свойство	Компонент	
	Button1	Button2
Left	32	149
Top	109	109
Width	75	75

Свойство	Компонент	
	Button1	Button2
Height	25	25
Caption	Расчет	Завершить

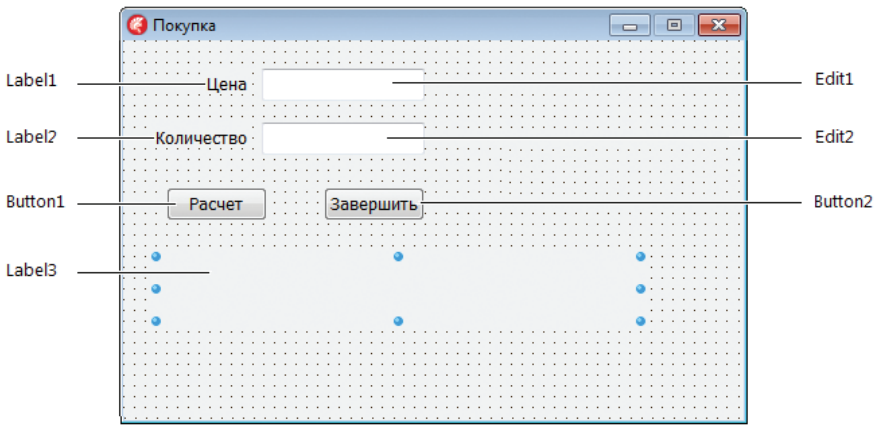


Рисунок 1.18. Окончательный вид формы программы Покупка

Завершив работу над формой, можно приступить к программированию — созданию процедур обработки *событий*.

Событие

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

Событие (Event) — это то, что происходит во время работы программы. Источником события могут быть действия пользователя, например, нажатие кнопки мыши или кнопки на клавиатуре, а также возможны события, генерируемые системой, например, сигнал от таймера.

У каждого события есть имя. Например, щелчок кнопкой мыши — это событие Click, двойной щелчок мышью — событие DblClick.

В табл. 1.9 приведены некоторые события, возникающие в результате действий пользователя.

Таблица 1.9. События

Событие	Описание
Click	Щелчок кнопкой мыши
DblClick	Двойной щелчок кнопкой мыши
MouseDown	Нажатие кнопки мыши

Событие	Описание
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 разными объектами, вызывают разные события. Например, щелчок (событие Click) на кнопке Расчет и щелчок на кнопке Завершить — это два разных события.

Процедура обработки события

Реакцией программы 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 **Расчет**.

Чтобы создать процедуру 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. Или выбрать компонент 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. 1.19) перечислены события, которые может воспринимать выбранный компонент, в правой — указаны имена процедур обработки событий. 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. Если поле рядом с именем события пустое, то это означает, что процедура обработки события не определена и, следовательно, компонент не будет реагировать на это событие.

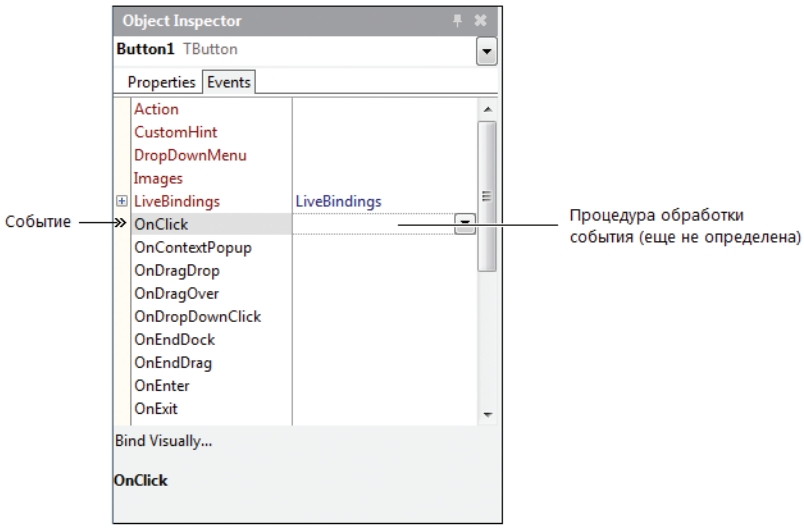


Рисунок 1.19. На вкладке Events перечислены события, которые может воспринимать компонент

Для того чтобы создать процедуру обработки события, нужно на вкладке Evens выбрать событие (сделать щелчок мышью на его имени), в ставшее доступным поле редактирования ввести имя процедуры обработки события (рис. 1.20) и нажать клавишу Enter. Следует обратить внимание на то, что согласно принятому в среде программистов соглашению, имя процедуры обработки должно содержать информацию о компоненте, реагирующей на событие, и событие. Например, Button1Click это «стандартное» имя процедуры обработки события Click на компоненте Button1.

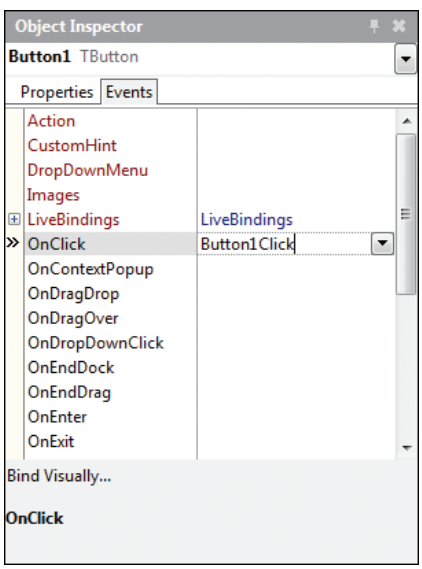


Рисунок 1.20. Рядом с именем события надо ввести имя процедуры обработки этого события

В результате этих действий в модуль формы будет добавлен шаблон процедуры обработки события и станет доступным окно редактора кода (рис. 1.21), в котором можно набирать инструкции процедуры обработки события.

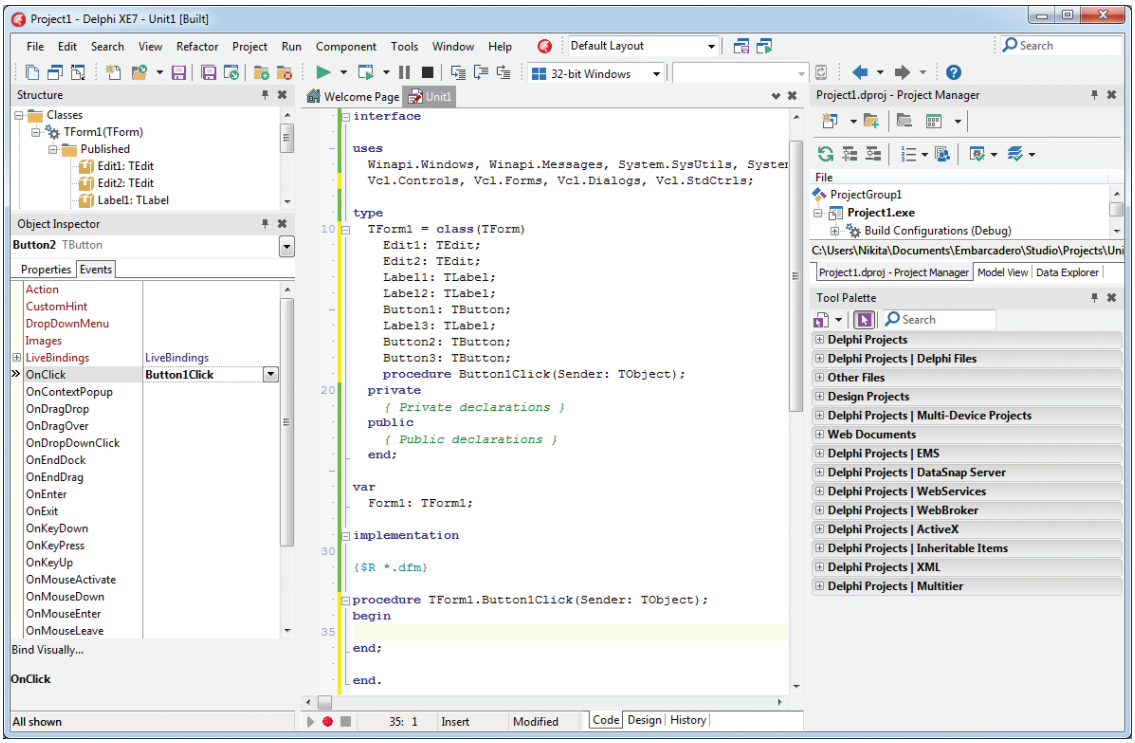


Рисунок 1.21. Шаблон процедуры обработки события

Обратите внимание, что в модуль формы среда разработки добавляет шаблон процедуры обработки события, также в описание класса формы добавляется строка:

```
procedure Button1Click(Sender: TObject);
```

- инструкция объявление процедуры обработки события.

Существует и другой способ создания процедуры обработки события. После выбора события, надо сделать двойной щелчок левой кнопкой мыши в поле редактирования, которое находится справа от имени события. В этом случае имя процедуры обработки события будет сформировано средой разработки, путем объединения имени компонента, для которого создается процедура обработки события, и имени события.

Процедура обработки события Click на кнопке **Расчет** приведена в листинге 1.1.

Листинг 1.1. Обработка события Click на кнопке Расчет

```
procedure TForm1.Button1Click(Sender: TObject);
var
  cena: double;    // цена
  kol: integer;    // количество
  sum: double;     // сумма
begin
  cena := StrToFloat(Edit1.Text);
  kol := StrToInt(Edit2.Text);
  sum := cena * kol;
```

```
  Label3.Caption :=
         'Цена: ' + FloatToStrF(cena,ffFixed,9,2) + ' руб.' + #13 +
         'Количество: ' + IntToStr(kol) + '' + #13 +
         'Сумма: ' + FloatToStrF(sum,ffFixed,9,2) +' руб.';
end;
```

Процедура TForm1.Button1Click рассчитывает цену покупки и выводит результат в поле Label3. Исходные данные (цена и количество) 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, а переменные cena и kol, соответственно, типа double и integer. Поэтому для преобразования строк в числа соответствующего типа используются функции StrToFloat и StrToInt. Следует обратить внимание, функция StrToFloat возвращает результат (число double) только в том случае, если строка, переданная функции в качестве параметра (находящаяся в поле редактирования Edit1), действительно является изображением числа в правильном формате. Что предполагает использование запятой в качестве разделителя целой и дробной частей числа, при стандартной для России настройке операционной системы. Если число введено неверно, например, вместо запятой введена точка, то возникает исключение (ошибка). Аналогичным образом работает функция StrToInt. Вычисленное значение суммы, а также исходные данные выводятся в поле Label3 путем присваивания значения свойству Caption. Свойство Caption строкового типа, поэтому число надо преобразовать в строку. Для преобразования чисел в строки используются функции FloatToStrF и IntToStr. У функции FloatToStrF четыре параметра: первый — значение, которое надо преобразовать в строку; второй — формат; третий и четвертый задают, соответственно, количество цифр целой и дробных частей.

В листинге 1.2 приведена процедура обработки события Click на кнопке **Завершить** (создается она точно так же, как и процедура обработки события Click для кнопки **Расчет**). В результате щелчка на кнопке **Завершить** программа должна завершить работу. Чтобы это произошло, надо закрыть окно программы (форму). Делает это метод Close.

Листинг 1.2. 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**).

Часто во время разработки программы 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 и открывающей скобки редактор кода выводит список параметров. Параметр, который в данный момент вводит программист, в подсказке выделяется полужирным шрифтом. Например, если набрать слово FloatToStr, которое является именем функции преобразования дробного числа в строку символов, и открывающую скобку, то на экране появится окно, в котором будут перечислены параметры функции (рис. 1.22).

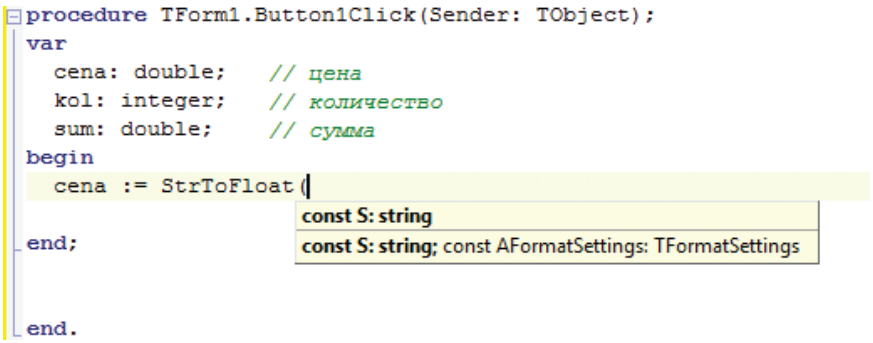


Рисунок 1.22. Пример подсказки – список параметров функции

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. 1.23). 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.

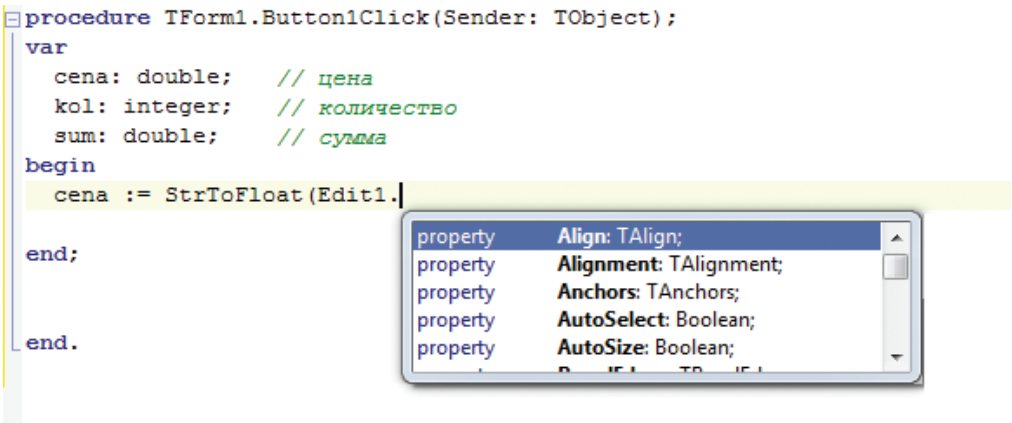


Рисунок 1.23. Редактор кода автоматически выводит список свойств и методов текущего объекта

Сохранение проекта

Проект — это набор файлов, используя которые, компилятор создает выполняемую программу (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).

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. 1.24) надо выбрать папку, предназначенную для проектов Delphi (по умолчанию это C:\Users*UserName*\Documents\Embarcadero\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 (можно оставить сформированное средой разработки имя модуля – слово Unit и порядковый номер модуля) и нажать кнопку **Сохранить**. Затем в появившемся окне **Save Project As** (рис. 1.25) надо ввести имя проекта. Обратите внимание на то, что имя проекта определяет имя exe-файла exe, который будет создан компилятором.

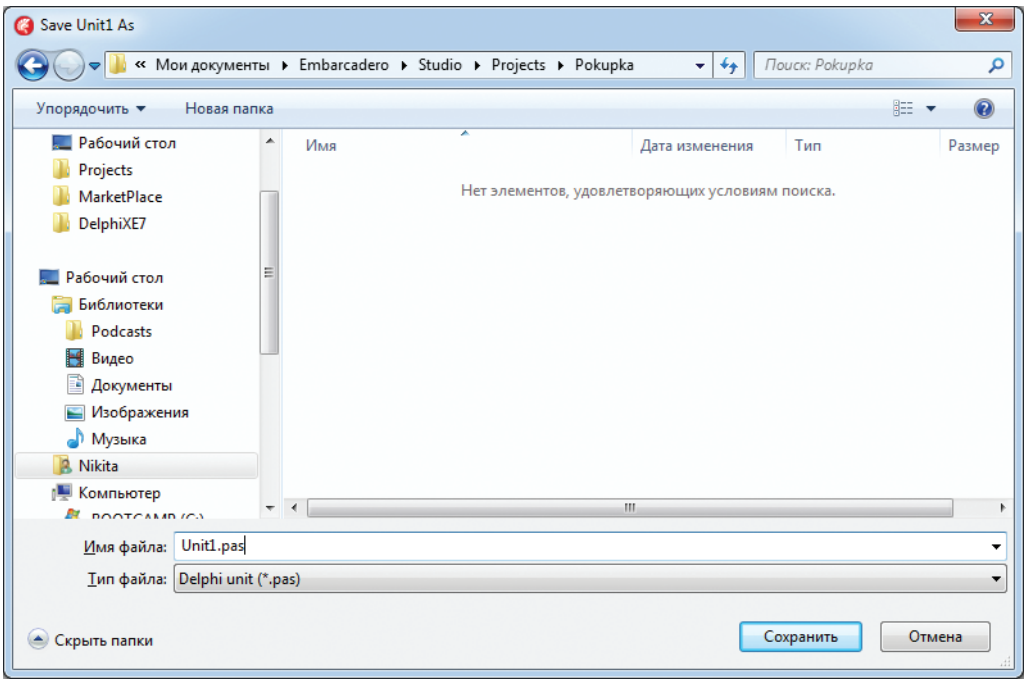


Рисунок 1.24. Сохранение модуля формы

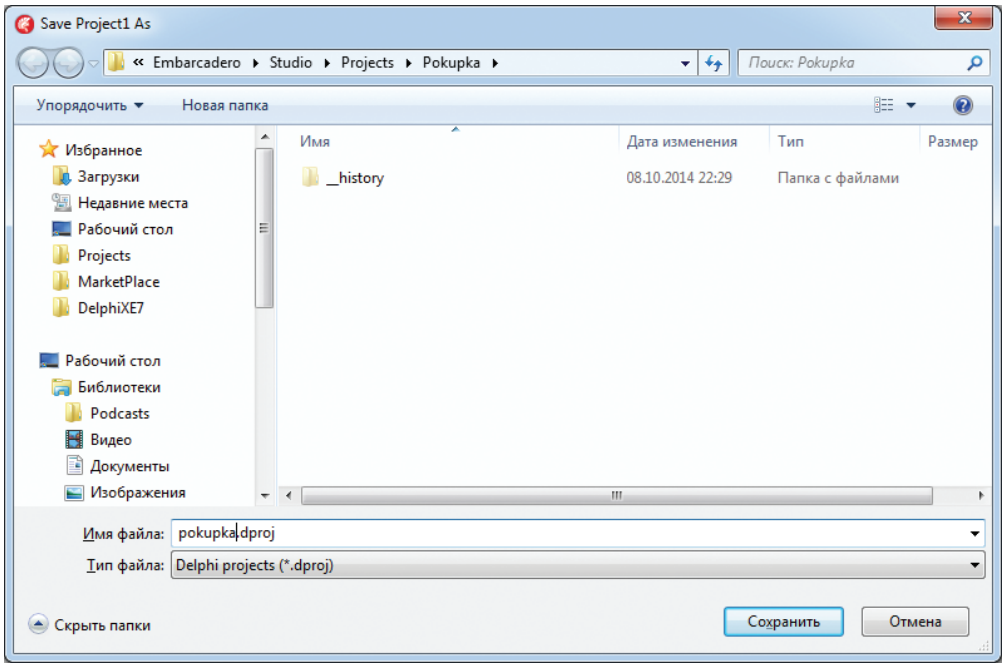


Рисунок 1.25. Сохранение проекта

Структура проекта

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 используется файл ресурсов (.res файл), файлы описания форм (.dfm отдельный 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 1.3 приведен главный модуль приложения Конвертер.

Листинг 1.3. Главный модуль приложения Конвертер (converter.dpr)

```
program pokupka;

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

{$R *.res}

begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  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 1.4 приведен модуль стартовой формы программы Покупка. Модуль содержит объявление класса TForm1. Обратите внимание, что инструкция Application.CreateForm(TForm1, Form1), которая находится в главном модуле приложения, создает объект класса TForm1, т. е. стартовую форму приложения Покупка.

Листинг 1.4. Модуль стартовой формы программы Конвертер (unit1.pas)

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.
Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Label3: TLabel;
    Button2: TButton;
    Button3: 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
 cena: double;    // цена
```

```
  kol: integer;    // количество
  sum: double;     // сумма
begin
  cena := StrToFloat(Edit1.Text);
  kol := StrToInt(Edit2.Text);

  sum := cena * kol;

  Label3.Caption := 'Цена: ' + FloatToStrF(cena,ffFixed,3,2) + ' руб.' + #13 +
                    'Количество: ' + IntToStr(kol) + '' + #13 +
                    'Сумма: ' + FloatToStrF(sum,ffFixed,3,2) +' руб.';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Form1.Close;
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.

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 (файл dpr), 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*. Он 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 в результате запуска программы из среды разработки (команда **Run** или **Run Without Debugging** меню **Run**), если с момента последней компиляции в программу были внесены изменения.

Процесс и результат компиляции отражаются в окне **Compile**. Если в программе синтаксических ошибок нет, то по завершении компиляции в поле **Compiling** отображается сообщение **Done** (рис. 1.26).

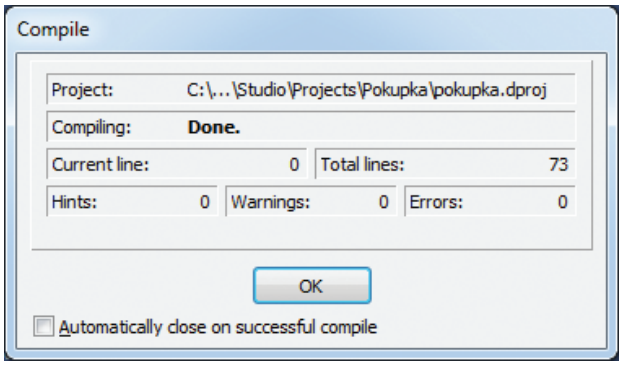


Рисунок 1.26. Результат компиляции (в программе ошибок нет)

Если в программе есть ошибки и (или) неточности, то в поле Compiling выводится сообщение **There are errors** (рис. 1.27) и отображается информация о количестве синтаксических (Errors) и семантических (Warnings) ошибок, а также о количестве неточностей (Hints).

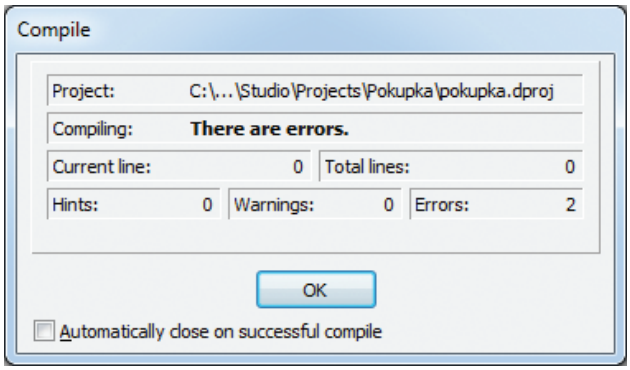


Рисунок 1.27. Результат компиляции (в программе есть ошибки)

Сами же сообщения об ошибках, предупреждения и подсказки отображаются в окне **Messages** (рис. 1.28). 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. 1.29).

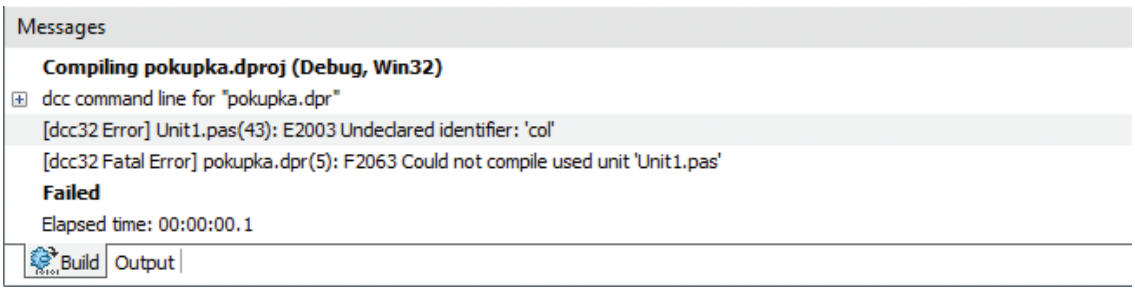


Рисунок 1.28. Сообщения об обнаруженных ошибках

```
30  implementation

    {$R *.dfm}

    procedure TForm1.Button1Click(Sender: TObject);
    var
      cena: double;    // цена
      kol: integer;    // количество
      sum: double;     // сумма
    begin
40    cena := StrToFloat(Edit1.Text);
      kol := StrToInt(Edit2.Text);

43    sum := cena * col;

      Label3.Caption := 'Цена: ' + FloatToStrF(cena,ffFixed,3,2) + ' руб.' + #13
                        'Количество: ' + IntToStr(kol) + '' + #13 +
                        'Сумма: ' + FloatToStrF(sum,ffFixed,3,2) +' руб.';
    end;

50  procedure TForm1.Button2Click(Sender: TObject);
    begin
      Form1.Close;
    end;

    end.
```

Рисунок 1.29. Строка программы, содержащая ошибку – необъявленный идентификатор

Ошибки

Компилятор генерирует выполняемую программу (файл exe) 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 не объявленные переменные. После очередного внесения изменений в текст программы выполняется повторная компиляция.

В табл. 1.10 приведены сообщения компилятора о типичных ошибках.

Таблица 1.10. 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
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

Сообщение	Вероятная причина
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 Run4

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, наиболее 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. 1.11 приведены наиболее часто выводимые компилятором предупреждения и подсказки о типичных неточностях в программе.

Таблица 1.11. 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

Чтобы запустить программу, надо в меню **Run** выбрать команду **Run** или **Run Without Debugging**. Можно также сделать щелчок на кнопке **Run** (рис. 1.30) или нажать клавишу F9.

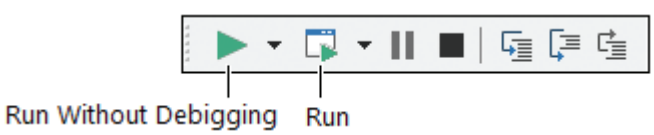


Рисунок 1.30. Чтобы запустить программу, сделайте щелчок на кнопке **Run**

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 **Цена** программы **Покупка** ввести строку 23.5 и сделать щелчок на кнопке **Расчет**, то на экране появится сообщение: '23.5' is not valid floating point value (рис. 1.31).

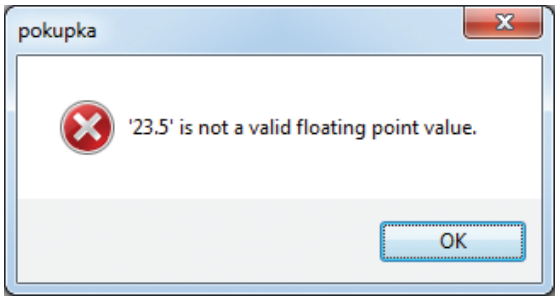


Рисунок 1.31. Пример сообщения об ошибке (исключении)

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 операционной системы Windows предполагает использование запятой в качестве десятичного разделителя. В рассматриваемом примере строка 23.5 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. 1.32). 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.

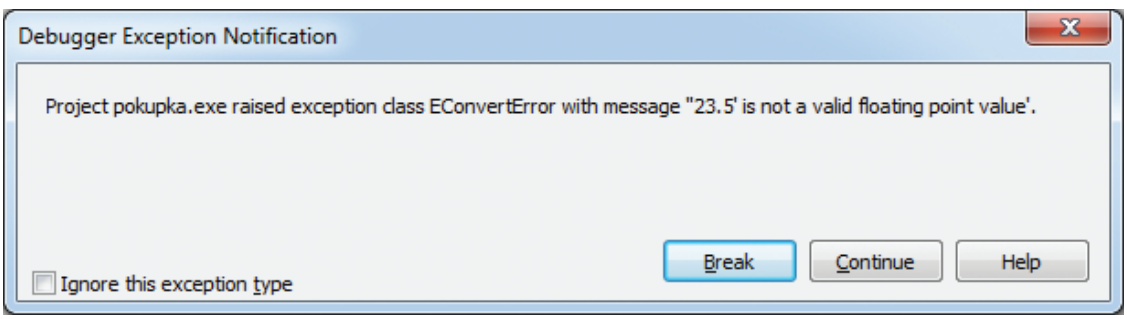


Рисунок 1.32. Пример сообщения о возникновении исключения

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. 1.12 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 1.12. 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)

В качестве примера использования инструкции try в листинге 1.5 приведена процедура обработки события Click на кнопке **Расчет** окна программы Покупка. При возникновении исключения EConvertError программа определяет причину (незаполненное поле или неверный формат данных) и выводит соответствующее сообщение (рис. 1.33).

Листинг 1.5. Щелчок на кнопке Расчет (с обработкой исключения)

```
procedure TForm1.Button2Click(Sender: TObject);
var
  cena: double;    // цена
  kol: integer;    // количество
  sum: double;     // сумма
begin
  try
    cena := StrToFloat(Edit1.Text);
    kol := StrToInt(Edit2.Text);

    sum := cena * kol;

    Label3.Caption := 'Цена: ' + FloatToStrF(cena,ffFixed,3,2) + ' руб.' + #13 +
                      'Количество: ' + IntToStr(kol) + '' + #13 +
                      'Сумма: ' + FloatToStrF(sum,ffFixed,3,2) +' 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;
```

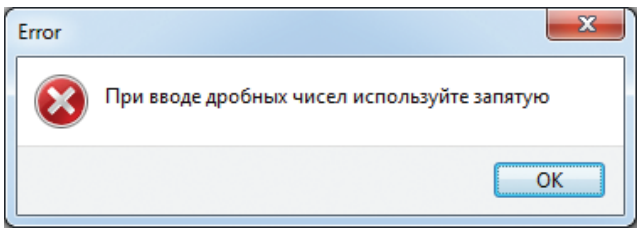


Рисунок 1.33. Пример сообщения об ошибке (диалог MessageDlg)

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. 1.13).

- *Кнопки* — список кнопок, отображаемых в окне сообщения (табл. 1.14).
- *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. 1.15), 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 1.13. Сообщения

Сообщение	Тип сообщения	Значок
Warning(Внимание)	mtWarning	
Error (Ошибка)	mtError	
Information (Информа-ция)	mtInformation	
Confirmation (Подтверж-дение)	mtConfirmation	

Таблица 1.14. Идентификаторы кнопок

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

Таблица 1.15. 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. 1.34 приведен пример окна сообщения — результат выполнения инструкции

```
ShowMessage('Надо ввести данные в оба поля')
```

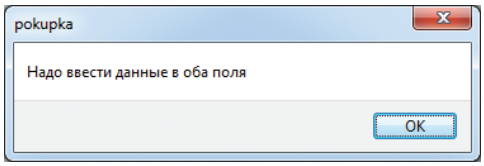


Рисунок 1.34. Сообщение, выведенное процедурой ShowMessage

Обратите внимание, 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 Покупка можно усовершенствовать. Например, сделать так, чтобы в поля редактирования пользователь мог ввести только правильную информацию (в поле Цена – дробное, в поле Количество - целое), и чтобы в результате нажатия клавиши Enter в поле **Цена** курсор переходил в поле **Количество**, а при нажатии этой же клавиши в поле **Количество** становилась активной кнопка **Расчет**. Можно сделать так, чтобы кнопка **Расчет** была доступной только в том случае, если данные введены в оба поля редактирования.

Чтобы внести изменения в программу, нужно открыть соответствующий проект. Сделать это можно обычным способом, выбрав в меню **File** команду **Open Project** (рис. 1.35). 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**.

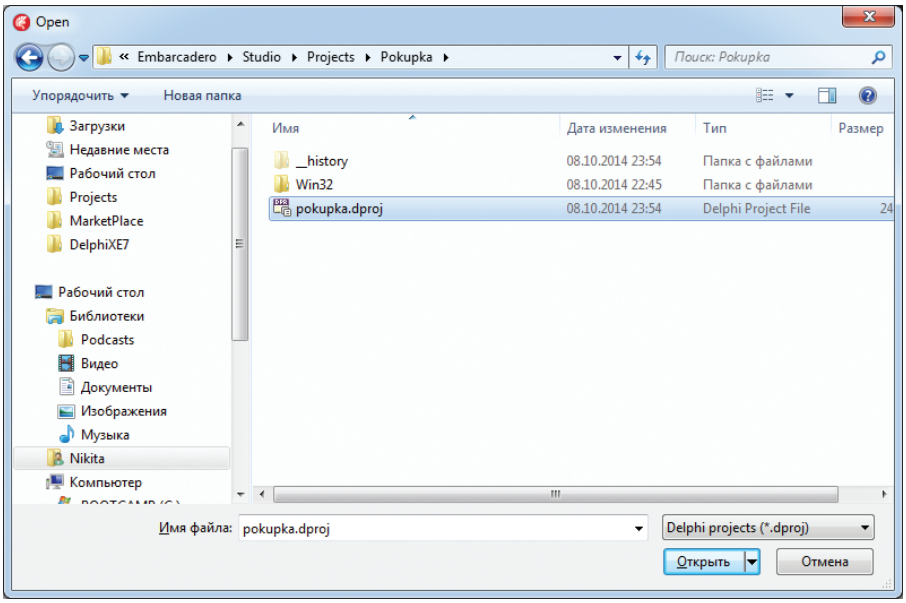


Рисунок 1.35. Загрузка проекта

Чтобы программа Покупка 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 «запрещенный», заменяет его на так называемый нуль-символ, который в поле редактирования не отображается, в результате чего у пользователя создается впечатление, что программа не реагирует на нажатие клавиши.

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 **Расчет**. Она проверяет, есть ли данные в полях редактирования и, если в каком-либо из полей данных нет, присваивает свойству Enabled кнопки Button1 значение False и тем самым делает кнопку недоступной. Следует обратить внимание на то, что действие, которое надо выполнить, если изменилось содержимое поля Edit1, ничем не отличается от действия, которое надо выполнить, при изменении содержимого 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, затем — указать имя этой процедуры в качестве имени процедуры обработки события другого компонента.

В листинге 1.6 приведен модуль формы усовершенствованной программы Покупка. Обратите внимание, в процедуре обработки события Click кнопки **Расчет** нет инструкций обработки исключений. Исключения просто не могут возникнуть, т. к. пользователь просто не сможет ввести в поля редактирования неверные данные.

Листинг 1.6. Модуль формы программы Покупка (Unit1.pas)

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.
Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Button1: TButton;
    Button2: TButton;
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit2KeyPress(Sender: TObject; var Key: Char);
```

```
    procedure Edit1Change(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
  cena: double;    // цена
  kol: integer;    // количество
  sum: double;     // сумма
begin
    cena := StrToFloat(Edit1.Text);
    kol := StrToInt(Edit2.Text);

    sum := cena * kol;

    Label3.Caption := 'Цена: ' + FloatToStrF(cena,ffFixed,3,2) + ' руб.' + #13 +
                      'Количество: ' + IntToStr(kol) + '' + #13 +
                      'Сумма: ' + FloatToStrF(sum,ffFixed,3,2) +' руб.';
end;

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

// нажатие клавиши в поле Цена
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) на символ DecimalSeparator - правильный символ,
         который должен использоваться при записи дробных чисел.
        }
        '.',',':
            begin
               Key :=  FormatSettings.DecimalSeparator;
               // проверим, введен ли уже в поле Edit
               // десятичный разделитель
               if pos(Key,Edit1.Text) <> 0
                  then Key := #0;
```

```
            end;
         #13: Edit2.SetFocus; // <Enter> - переместить курсор в поле Edit2
         else Key := #0; // остальные символы запрещены
     end;
end;

// нажатие клавиши в поле количество
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
      case Key of
        '0'..'9', #8: ;  // цифры и <Backspace>

         #13: Button1.SetFocus; // <Enter> - фокус на кнопку Button1

         else Key := #0; // остальные символы запрещены
     end;
end;

// изменилось содержимое поля редактирования
// процедура Edit1Change обрабатывает событие Change ОБОИХ компонентов Edit
procedure TForm1.Edit1Change(Sender: TObject);
begin
    // т. к. данные в поле редактирования изменились,
    // то информация в Label3 уже не соответствует данным
    // находящимся в поле редактирования
    label3.Caption := '';

    if ((Edit1.Text = '') or (Edit2.Text = '')) then
        // в одном из полей нет данных
        Button1.Enabled := False
    else
        Button1.Enabled := True;
end;

end.
```

Настройка приложения

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

Настройка приложения выполняется в окне **Project Options** (рис. 1.36), которое становится доступным в результате выбора в меню **Project** соответствующей команды.

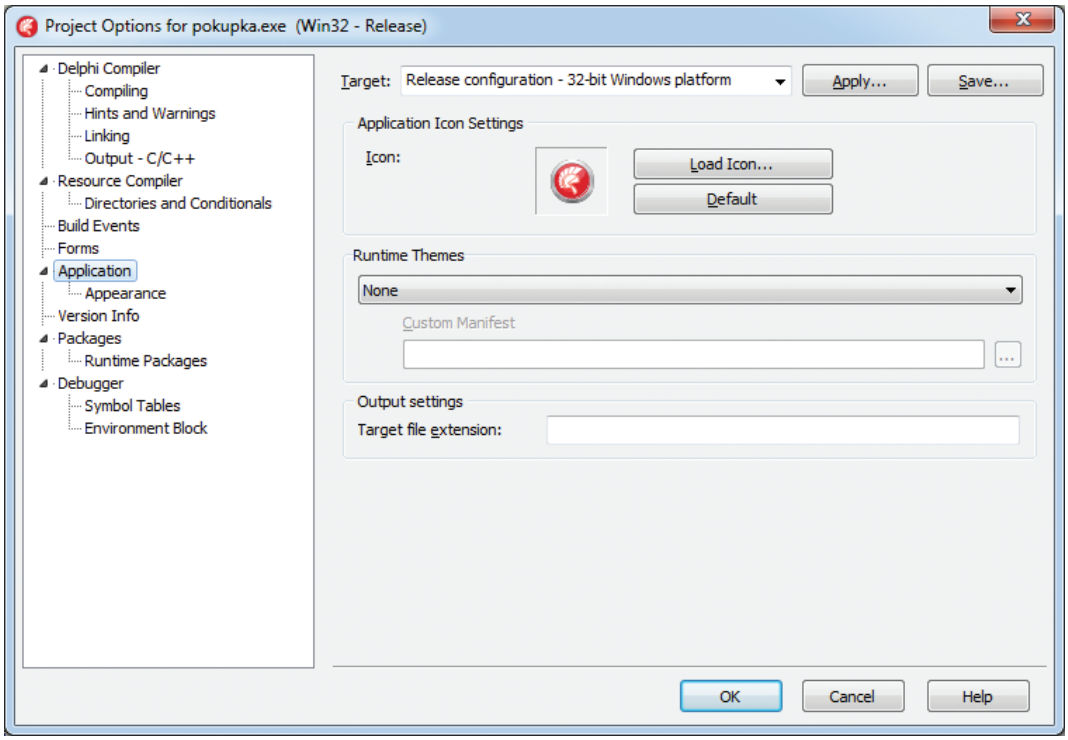


Рисунок 1.36. Настройка приложения

Сначала надо установить режим компиляции Release – в списке Target выбрать Release configuration – 32-bit Windows platform. Затем на вкладке **Application** можно задать значок приложения. Чтобы это сделать, надо щелкнуть на кнопке **Load Icon** и, используя стандартное окно просмотра папок, указать подходящий значок (.ico файл). На вкладке **Application / Appearance** надо задать название приложения. 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).

После изменения параметров проекта следует выполнить окончательную сборку приложения – двойным щелчком мыши на слове Release в окне **Project Manager** установить режим построения Release (рис. 1.37) и в меню **Project** выбрать команду **Build**.

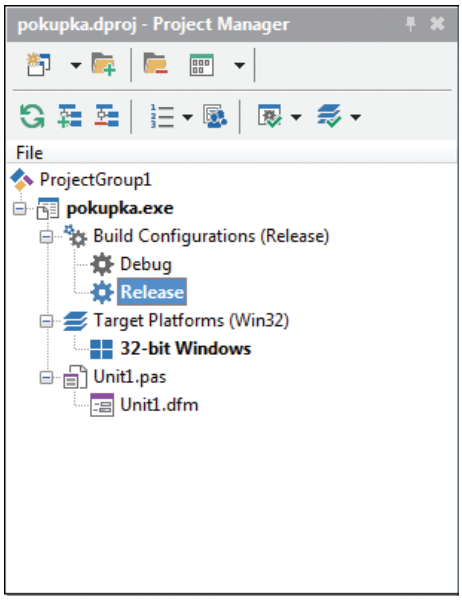


Рисунок 1.37. Окно **Project Manager**

Установка приложения на другой компьютер

Приложение, созданное в Delphi XE7, можно перенести на другой компьютер, например, с помощью USB Flash-накопителя (exe файл программы после компиляции в режиме Release находится в каталоге Win32\Release папки проекта). Однако, следует обратить внимание на то, что по умолчанию программа компилируется в режиме использования динамических пакетов (runtime packages) и поэтому она будет работать на другом компьютере только в том случае, если на нем есть необходимые библиотеки (пакеты). 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 rtl210.bpl) и библиотека визуальных компонентов (Visual Components Library, файл vcl210.bpl). Указанные файлы следует поместить в папку приложения или, если библиотеки будут использоваться несколькими программами, в папку C:\Windows\System32.

Замечание. Имена библиотек (пакетов), 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, 140 это библиотека Delphi 2010, 210 – Delphi XE7.

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 построение программы, в окне **Project Options** раскрыть вкладку **Packages** и сбросить флажок **Build with runtime packages** (рис. 1.38).

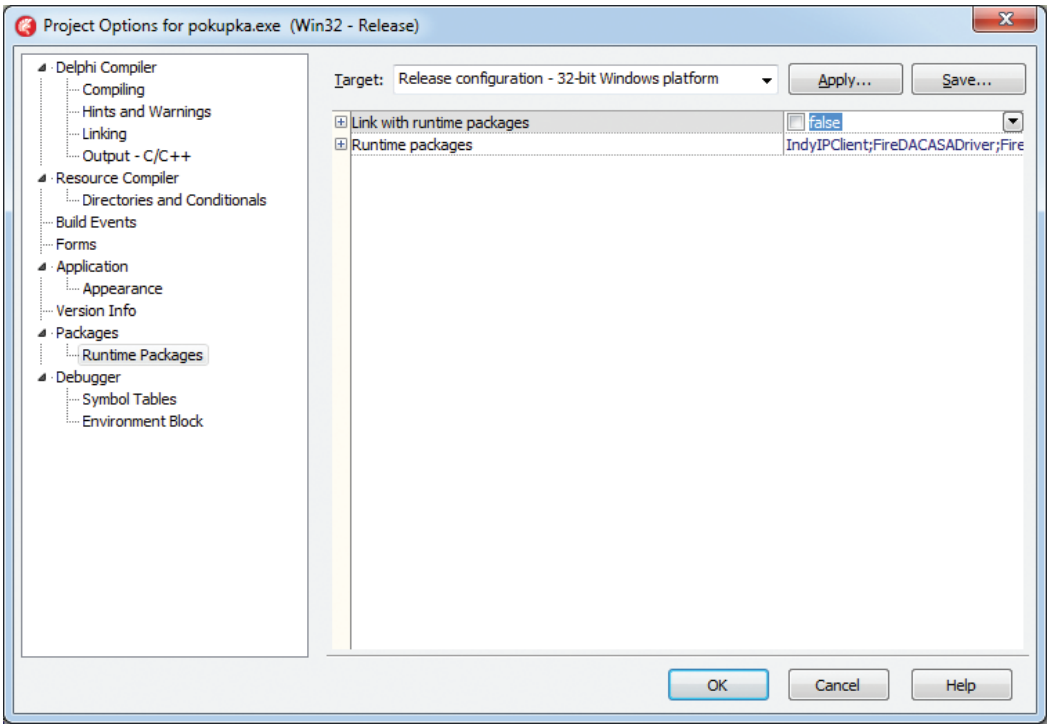


Рисунок 1.38. Чтобы программа не использовала динамические библиотеки надо сбросить флажок **Link with runtime packages**

Для справки: размер exe файла простой программы, типа Покупка, скомпилированной в режиме использования динамических пакетов, как правило, не превышает 100 КБ. А если программа не использует динамические пакеты, то размер exe файла увеличивается до нескольких мегабайт.

Глава 2. VCL Компоненты

В этой главе приведено описание и примеры использования базовых, наиболее часто используемых, VCL компонентов. К базовым можно отнести компоненты, обеспечивающие взаимодействие с пользователем (Edit, Label, Button, CheckBox, RadioButton, ListBox), компонент Image, предназначенный для отображения иллюстраций, и некоторые другие, например, Timer. Базовые компоненты находятся на вкладках **Standard**, **Additional**, **Win32** и **System**.

Label

Компонент Label, его значок (рис. 2.1) находится на вкладке **Standard**, предназначен для отображения текста.

Свойства компонента приведены в табл. 2.1.

Рисунок 2.1. Значок компонента Label

Таблица 2.1. Свойства компонента Label

Свойство	Описание
Name	Имя (идентификатор) компонента
Caption	Отображаемый текст
Left	Расстояние от левой границы поля вывода до левой границы формы
Top	Расстояние от верхней границы поля вывода до верхней границы формы
Height	Высота поля вывода
Width	Ширина поля вывода
AutoSize	Признак того, что размер поля определяется его содержимым
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (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)

Следующая программа (ее форма приведена на рис. 2.2, а текст процедуры обработки события Click на кнопке **Ok** — в листинге 2.1) демонстрирует возможности компонента Label. 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 отображается 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. 2.2.

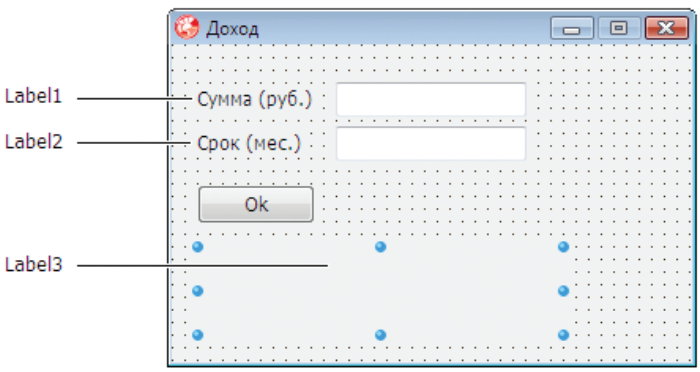


Рисунок 2.2. Форма программы Доход

Таблица 2.2. 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 2.1. Доход (компонент Label)

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var  sum, pr: real;       // сумма, процентная ставка
  srok: integer;        // срок вклада в месяцах
  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. 2.3) находится на вкладке **Standard**, предназначен для ввода и редактирование текста. Свойства компонента приведены в табл. 2.3.

Рисунок 2.3. Значок компонента Edit

Таблица 2.3. 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
Top	Расстояние от верхней границы компонента до верхней границы формы
Height	Высота

Свойство	Описание
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. Следующая программа (ее форма приведена на рис. 2.4, а текст — в листинге 2.2) демонстрирует использование компонента Edit для ввода данных различного типа. 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).

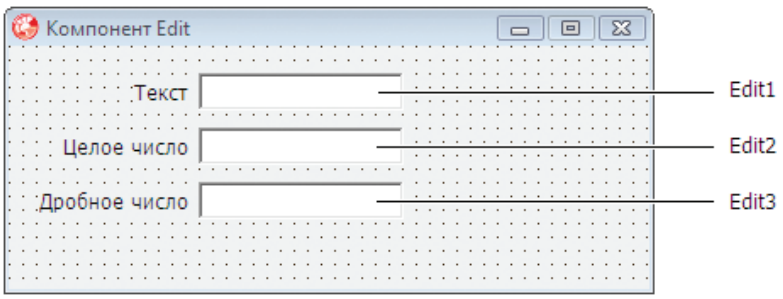


Рисунок 2.4. Форма программы Компонент Edit

Листинг 2.2. 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. 2.5) находится на вкладке **Standard**, представляет собой командную кнопку. Свойства компонента приведены в табл. 2.4.

Рисунок 2.5. Значок компонента Button

Таблица 2.4. 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)

Свойство	Описание
ShowHint	Разрешает (True) или запрещает (False) отображение подсказки при позиционировании указателя на кнопке

Следующая программа (ее форма приведена на рис. 2.6, а текст — в листинге 2.3) демонстрирует использование компонента Button. 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 OK 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.

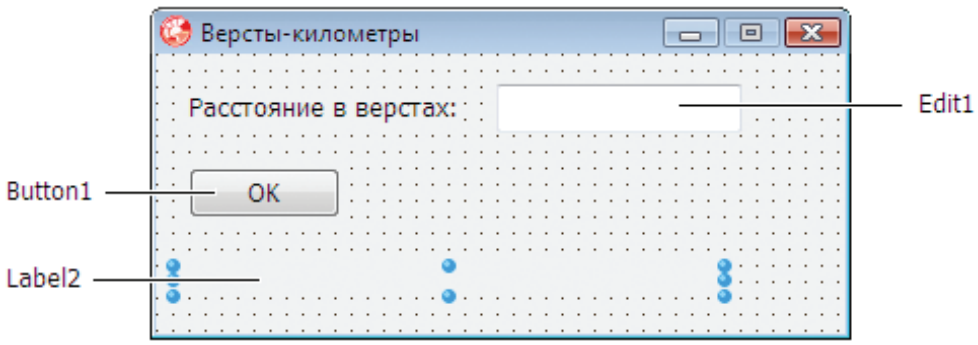


Рисунок 2.6. Форма программы Версты-километры

Листинг 2.3. Версты-километры (компонент Button)

```
// 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
             1// 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. 2.7) находится на вкладке **Standard**, представляет собой переключатель, который может находиться в одном из двух состояний: выбранном или невыбранном. Часто вместо «выбранный» говорят «установленный», а вместо «невыбранный» — «сброшенный». Рядом с переключателем обычно находится поясняющий текст. Свойства компонента CheckBox приведены в табл. 2.5.

Рисунок 2.7. Значок компонента CheckBox

Таблица 2.5. 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

Свойство	Описание
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

Следующая программа (ее форма приведена на рис. 2.8, а текст — в листинге 2.4) демонстрирует использование компонентов CheckBox. Программа позволяет посчитать цену автомобиля в комплектации, выбранной пользователем. Значения свойств компонентов CheckBox приведены в табл. 2.6.

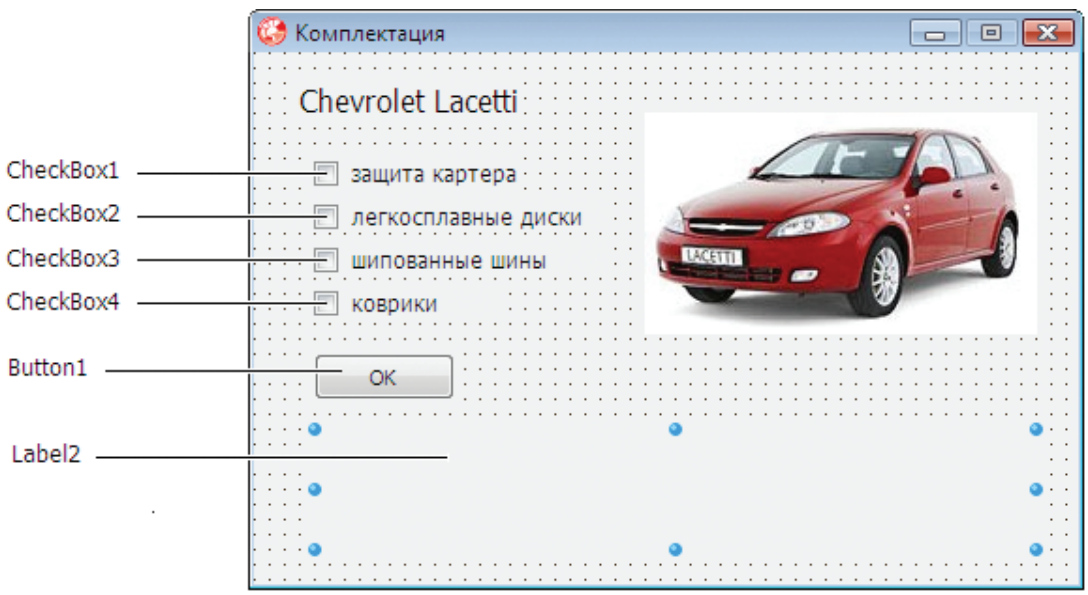


Рисунок 2.8. Форма программы Комплектация

Таблица 2.6. 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 2.4. Комплектация (компонент CheckBox)

```
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. 2.9) 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. 2.7.

Рисунок 2.9. Значок компонента RadioButton

Таблица 2.7. 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

Следующая программа (ее форма приведена на рис. 2.10, а текст — в листинге 2.5) демонстрирует использование компонента RadioButton. Программа вычисляет стоимость печати фотографий. Значения свойств компонентов RadioButton приведены в табл. 2.8.

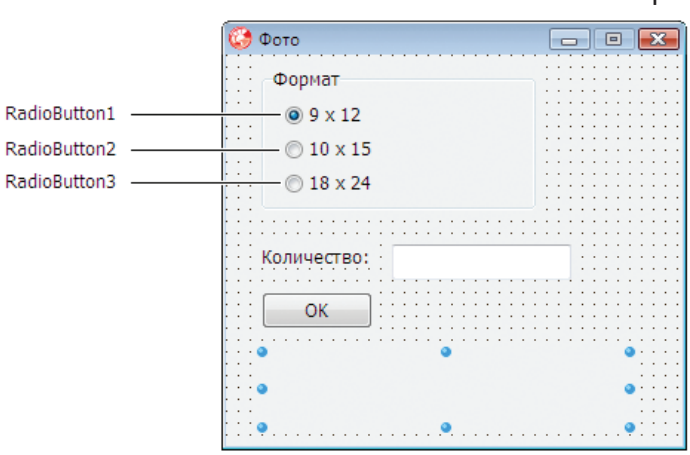


Рисунок 2.10. Форма программы **Фото**

Таблица 2.8. Значения свойств компонентов RadioButton

Компонент	Свойство	Значение
RadioButton1	Caption	9x12
	Checked	True
RadioButton2	Caption	10x15
	Checked	False
RadioButton3	Caption	18x24
	Checked	False

Листинг 2.5. Фото (компонент RadioButton)

```
// 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. 2.11) находится на вкладке **Standard**, представляет собой комбинацию поля редактирования и списка, что дает возможность ввести данные путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 2.9.

Рисунок 2.11. Значок компонента ComboBox

Таблица 2.9. 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

Свойство	Описание
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. 2.12).

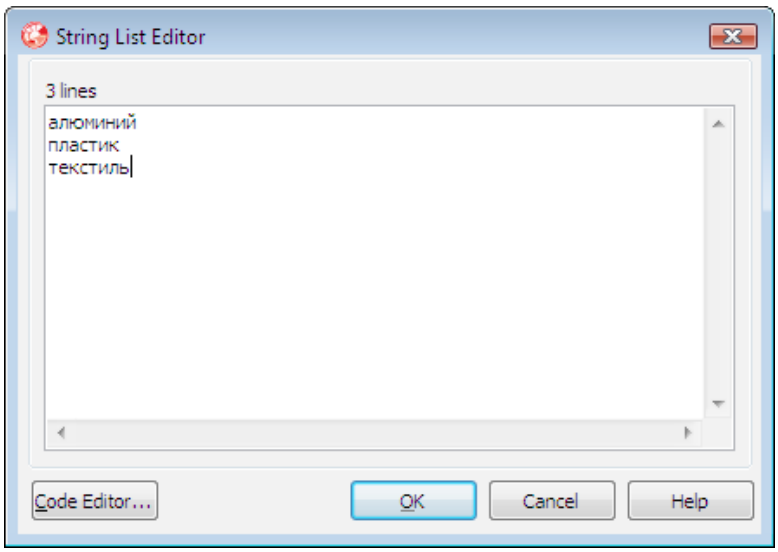


Рисунок 2.12. Формирование списка компонента во время разработки формы

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');
```

Следующая программа, ее форма приведена на рис. 2.13, демонстрирует использование компонента ComboBox. 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 2.6.

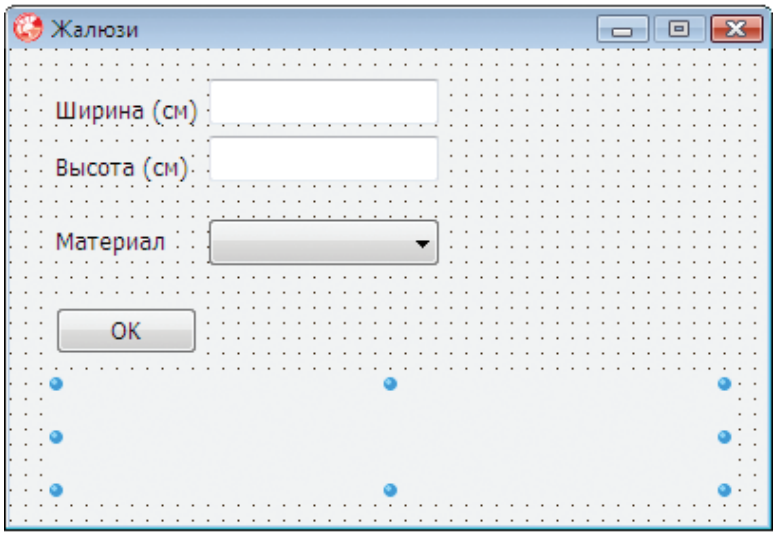


Рисунок 2.13. Форма программы **Жалюзи**

Листинг 2.6. Жалюзи (компонент ComboBox)

```
// 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. 2.14) находится на вкладке **Standard**, представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 2.10.

Рисунок 2.14. Значок компонента ListBox

Таблица 2.10. 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 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. Формирование списка во время работы программы обеспечивает метод Add свойства Items.

Следующая программа, ее окно приведено на рис. 2.15, демонстрирует использование компонента ListBox. Программа позволяет просмотреть фотографии (JPG-файлы).

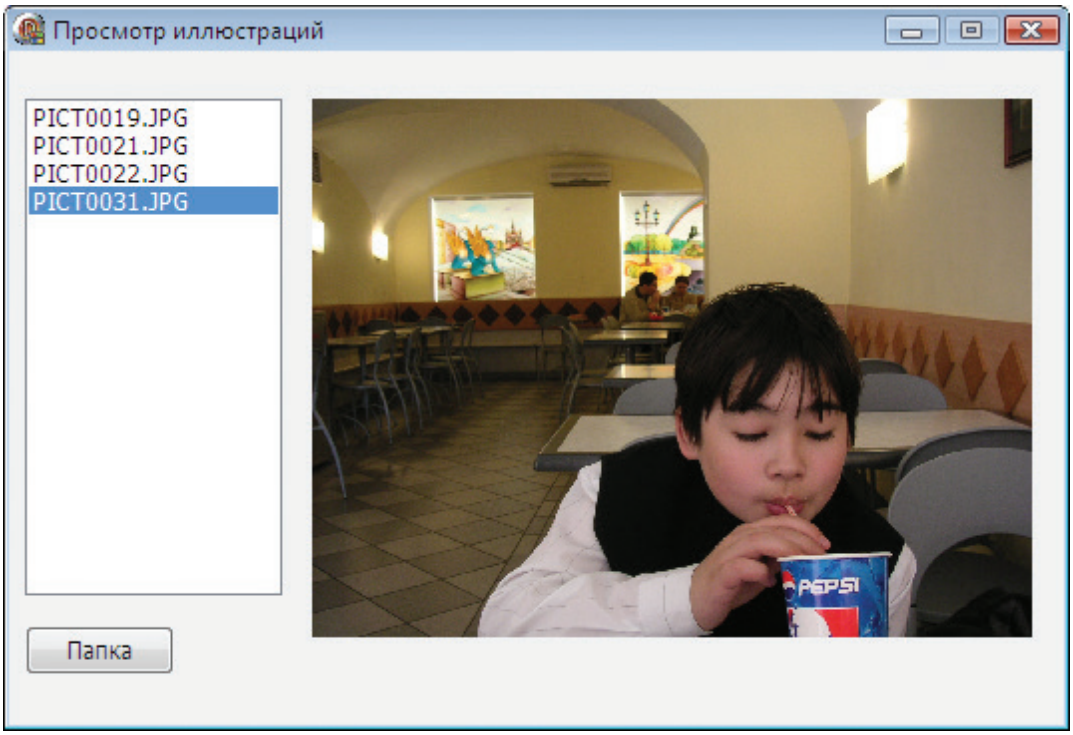


Рисунок 2.15. Окно программы **Просмотр иллюстраций**

Форма программы приведена на рис. 2.16, а текст — в листинге 2.7. 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. 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.

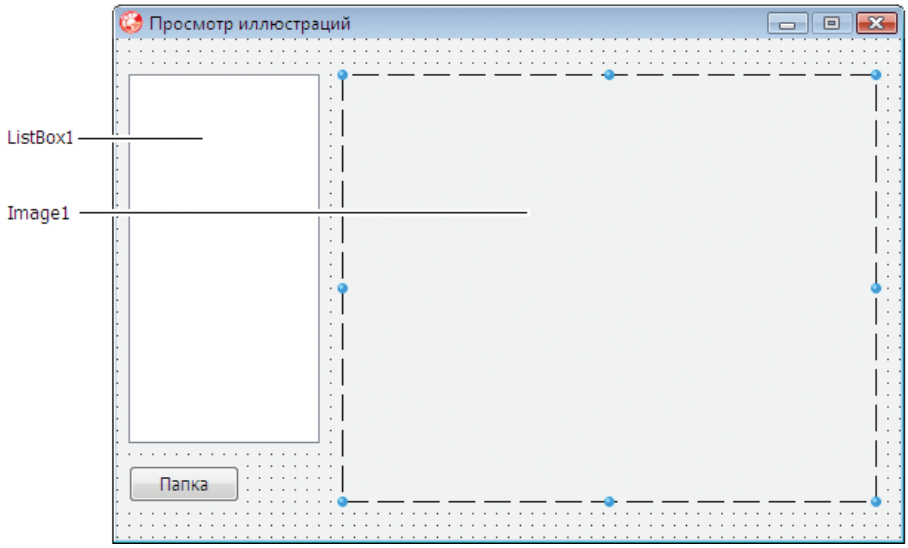


Рисунок 2.16. Форма программы **Просмотр иллюстраций**

Листинг 2.7. Просмотр иллюстраций (компонент ListBox)

```
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 JPG-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 JPG-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 := 'Просмотр иллюстраций - ' + Path;
    FillListBox;
  end;
end;

end.
```

Timer

Компонент Timer, его значок (рис. 2.17) находится на вкладке System, генерирует последовательность событий Timer. Компонент является невизуальным, то есть во время работы программы не отображается на форме. Свойства компонента приведены в табл. 2.11.

Рисунок 2.17. Значок компонента Timer

Таблица 2.11. Свойства компонента Timer

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

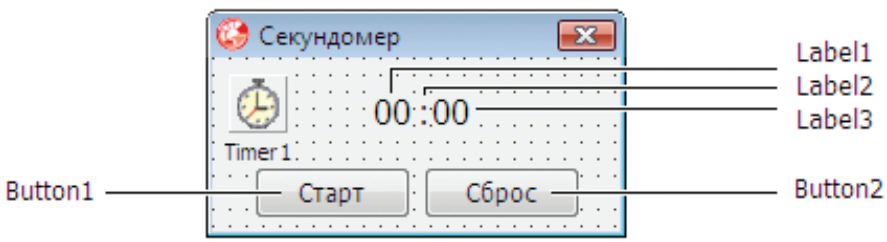


Рисунок 2.18. Форма программы Секундомер

Программа Секундомер (ее форма приведена на рис. 2.18, а текст — в листинге 2.8) демонстрирует использование компонента Timer. 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 сек., инвертирует значение свойства Visible компонента Label2 (в результате 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 2.8. Секундомер (компонент Timer)

```
// 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 от 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;
```

Image

Компонент Image, его значок (рис. 2.19) находится на вкладке **Additional**, обеспечивает отображение графики: иллюстраций, фотографий, рисунков. Свойства компонента приведены в табл. 2.12.

Рисунок 2.19. Значок компонента Image

Таблица 2.12. 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
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 JPG-файла, в директиву uses модуля формы надо добавить ссылку на модуль JPEG.

Следующая программа, ее форма приведена на рис. 2.20, демонстрирует использование компонента Image для просмотра фотографий. Необходимо отметить, что во время создания формы сначала нужно настроить компонент Panel (присвоить значение alBottom свойству Align), а затем компонент Image (значения свойств приведены в табл. 2.13).

В листинге 2.9 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.

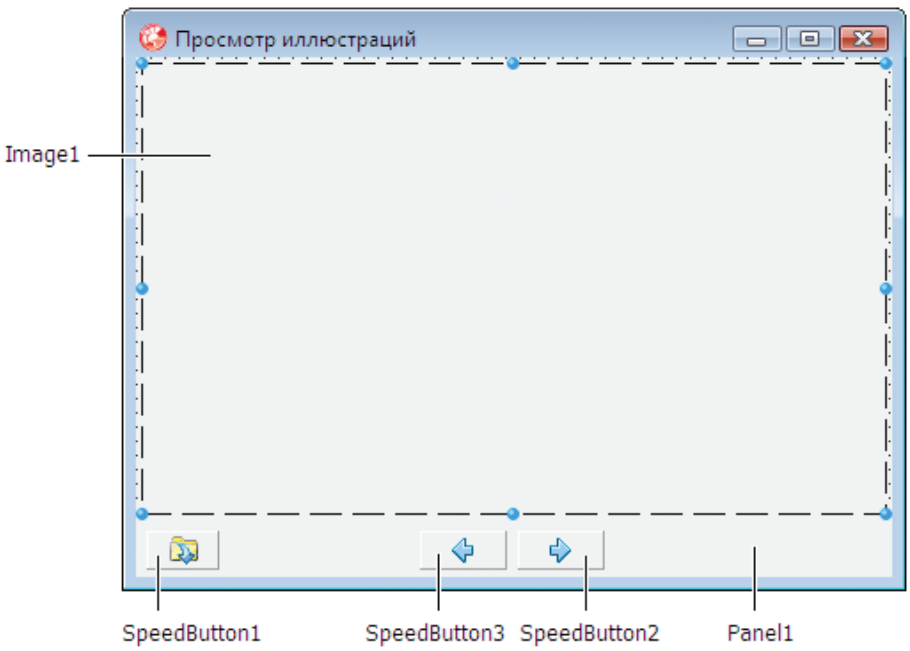


Рисунок 2.20. Форма программы Просмотр иллюстраций

Таблица 2.13. Значения свойств компонента Image

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

Листинг 2.9. Просмотр иллюстраций (компонент Image)

```
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 jpg-файл
        Pictures.Clear; // очистить список иллюстраций
        Pictures.Add(aSearchRec.Name); // добавить имя файла в список иллюстраций

        // получить имена остальных jpg-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 jpg-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.
```

OpenDialog

Компонент OpenDialog, его значок (рис. 2.21) находится на вкладке **Dialogs**, представляет собой диалог **Открыть**. Свойства компонента приведены в табл. 2.14. Отображение диалога обеспечивает метод Execute. Значением метода Execute является идентификатор кнопки, щелчком на которой пользователь закрыл окно.

Рисунок 2.21. Значок компонента OpenDialog

Таблица 2.14. 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

Следующая программа демонстрирует использование компонента OpenDialog для выбора файла, содержимое которого надо отобразить в поле компонента Memo. Форма программы приведена на рис. 2.22. 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 2.10). Значения свойств компонентов программы приведены в табл. 2.15.

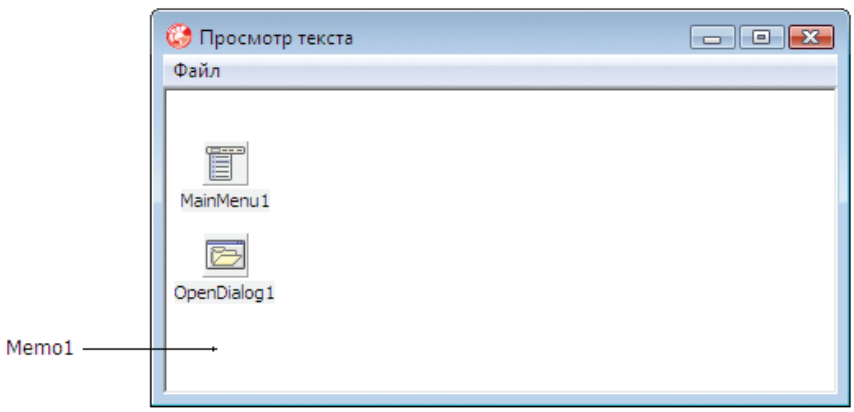


Рисунок 2.22. Форма программы

Таблица 2.15. Значения свойств компонентов

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

Листинг. 2.10. Выбор файла (компонент OpenDialog)

```
// 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. 2.23) находится на вкладке **Dialogs**, представляет собой диалог **Сохранить**. Свойства компонента приведены в табл. 2.16. Отображение диалога обеспечивает метод Execute, значение которого позволяет определить, щелчком на какой кнопке **Открыть** или **Отмена**, пользователь закрыл диалог.

Рисунок 2.23. Значок компонента SaveDialog

Таблица 2.16. 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
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 2.11 приведена процедура обработки события CloseQuery окна простого редактора текста (форма приведена на рис. 2.24, значения свойств компонента SaveDialog — в табл. 2.17).

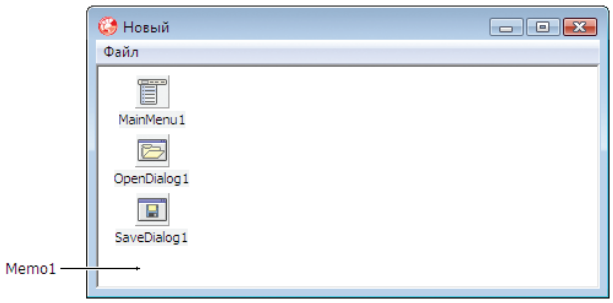


Рисунок 2.24. Форма программы

Таблица 2.17. Значения свойств компонента SaveDialog

Свойство	Значение
DefaultExt	txt
Filter	Текст|*.txt

Листинг 2.11. Обработка события CloseQuery (компонент SaveDialog)

```
// 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;
```

Глава 3. Графика

В этой главе рассказывается, как отобразить в окне программы графику, что надо сделать, чтобы на поверхности формы появилась диаграмма, график или фотография. Также вы познакомитесь с принципами реализации анимации, узнаете, как «оживить» картинку.

Компоненты Image и PainBox

Для отображения графики в окне программы используют компоненты Image и PaintBox. Компонент Image находится на вкладке **Additional** палитры компонентов, он предназначен для отображения иллюстраций, компонент PaintBox (находится на вкладке **System**) используется качестве поверхности, на ней формируется графика из графических элементов во время работы программы.

Графическая поверхность

Графика, в поле компонента PaintBox, формируется на его *графической поверхности*. Доступ к графической поверхности, которая представляет собой объект TCanvas, осуществляется через свойство Canvas.

Методы вывода графических примитивов рассматривают свойство Canvas как поверхность, на которой они могут *рисовать* (canvas с англ. «поверхность», «холст для рисования»). Графическая поверхность представляет собой совокупность точек (пикселей). Положение пикселя на графической поверхности определяется горизонтальной (X) и вертикальной (Y) координатами. Координата X возрастает слева направо, Y – сверху вниз и (рис. 3.1). Таким образом, левый верхний пиксель имеет координаты (0, 0).

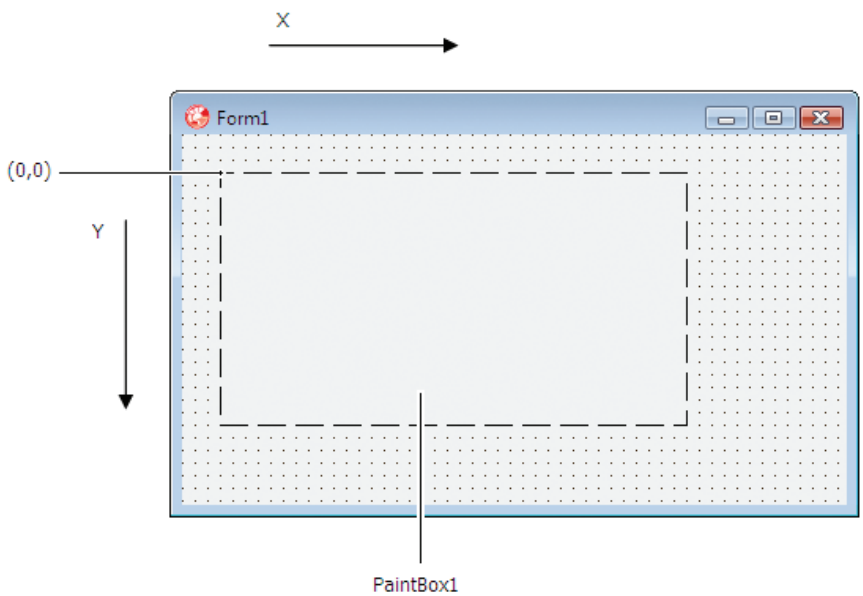


Рисунок 3.1. Координаты точек графической поверхности

Чтобы на графической поверхности появилась линия, прямоугольник, эллипс или другой графический элемент, необходимо вызвать соответствующий метод объекта TCanvas. Например, инструкция

```
Form1.PaintBox1.Canvas.Rectangle(10,20,150,40);
```

рисует в поле компонента PaintBox1 прямоугольник 140x20 пикселей, левый верхний угол которого находится в точке с координатами (10,20).

Методы класса TCanvas, обеспечивающие вывод графики, приведены в табл. 3.1.

Таблица 3.1. Методы класса TCanvas

Метод	Действие
MoveTo(x,y)	Перемещает указатель текущей точки в точку с указанными координатами.
LineTo(x,y)	Рисует линию из текущей точки в точку с указанными координатами (начальную точку линии можно задать при помощи метода 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 точку дуги. Цвет дуги определяет свойство Pen.Color
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 ломанную линию. Points — массив типа TPoint. Каждый элемент массива представляет собой запись, поля x и y которой содержат координаты точки перегиба ломаной. n — количество звеньев ломанной. 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.

Цвет, толщину и вид линий (или границы геометрической фигуры), а также цвет и стиль закраски внутренних областей геометрических фигур, рисуемых соответствующими методами, определяют свойства графической поверхности (табл. 3.2).

Таблица 3.2. Свойства графической поверхности

Свойство	Описание
Pen	Карандаш. Определяет цвет, толщину и стиль линии (границы геометрической фигуры).
Brush	Кисть. Определяет цвет и стиль закраски внутренней области геометрической фигуры.
Font	Шрифт. Определяет шрифт, который используется для вывода текста на графическую поверхность.

Событие Paint

Графику на графической поверхности должна формировать процедура обработки события Paint. Это событие возникает автоматически всякий раз, когда объект появляется на экране, например, после запуска программы или после того как пользователь сдвинет другое окно, частично или полностью перекрывающее окно программы, или развернет окно программы, если оно было свернуто.

В качестве примера в листинге 3.1 приведена процедура обработки события Paint компонента PaintBox, она рисует в поле компонента итальянский флаг.

Листинг 3.1. Вывод графики в компонент PaintBox

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  x,y: integer;    // координаты левого верхнего угла
  w,h: integer;    // размер полосы
begin

  x := 20;
  y := 20;

  w:=40;
  h:=75;

  // зеленая полоса (прямоугольник)
  PaintBox1.Canvas.Brush.Color := clGreen;
  PaintBox1.Canvas.Rectangle(x,y,x+w,y+h);

  // белая полоса
  x:=x+w-1;
  PaintBox1.Canvas.Brush.Color := clWhite;
  PaintBox1.Canvas.Rectangle(x,y,x+w,y+h);

  // красная полоса
  x:=x+w-1;
  PaintBox1.Canvas.Brush.Color := clRed;
  PaintBox1.Canvas.Rectangle(x,y,x+w,y+h)
```

```
  // подпись
  PaintBox1.Canvas.Brush.Style := bsClear;
  PaintBox1.Canvas.Font.Name := 'Tahoma';
  PaintBox1.Canvas.Font.Size := 12;
  // вычислим координату x, чтобы подпись была по центру
  // относительно флага
  x := x-2*w  + ( 3*w - PaintBox1.Canvas.TextWidth('Италия')) div 2;
  PaintBox1.Canvas.TextOut(x, y+h + Font.Size,'Италия');

end;
```

Карандаш и кисть

Вид графического элемента, рисуемого на графической поверхности соответствующим методом, 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. 3.3) определяют цвет, толщину и вид линии, свойства объекта Brush (табл. 3.4) — цвет и стиль закраски внутренних областей геометрических фигур.

Таблица 3.3. Свойства объекта Pen

Свойство	Описание
Color	Цвет линии
Width	Толщина линии (задается в пикселях)
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

Таблица 3.4. Свойства объекта Brush

Свойство	Описание
Color	Цвет закраски 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. 3.5 приведены именованные константы, которые можно использовать, чтобы задать цвет.

Таблица 3.5. Константы TColor

Цвет	Константа
clAqua	Бирюзовый
clBlack	Черный
clBlue	Синий
clFuchsia	Ярко-розовый
clGreen	Зеленый
clLime	Салатный
clMaroon	Каштановый
clNavy	Темно-синий
clOlive	Оливковый
clPurple	Пурпурный
clRed	Красный
clSilver	Серебристый
clTeal	Зелено-голубой
clWhite	Белый

Если необходимо установить цвет отличный от стандартного, то в свойство Color надо записать его RGB-код. 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

График, диаграмма, схема, чертеж это — совокупность графических *элементов* или *примитивов*: линий, прямоугольников, окружностей, дуг, а также букв (текста).

Вычерчивание графических примитивов на графической поверхности выполняют соответствующие методы класса TCanvas.

Инструкция вызова метода, выполняющего рисование графического элемента, в общем виде выглядит так:

Объект.Canvas.*Метод*(*Параметры*)

Параметр Объект определяет объект, на поверхности которого нужно нарисовать графический элемент. В качестве объекта обычно указывается компонент PaintBox. *Метод* — это метод, который рисует нужный графический элемент. Параметры, в большинстве случаев, определяют положение и размер графического элемента на графической поверхности.

Текст

Вывод текста на графическую поверхность объекта обеспечивает метод TextOut. Инструкция вызова метода TextOut в общем виде выглядит так:

Объект.Canvas.TextOut(x,y,*Текст*)

Параметры x и y задают координаты точки графической поверхности, от которой выполняется вывод текста (рис. 3.2). Следует 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 необходимо обратить внимание на размер области вывода текста, он зависит от количества символов (длины текста) и характеристик шрифта, который применяется для отображения текста.

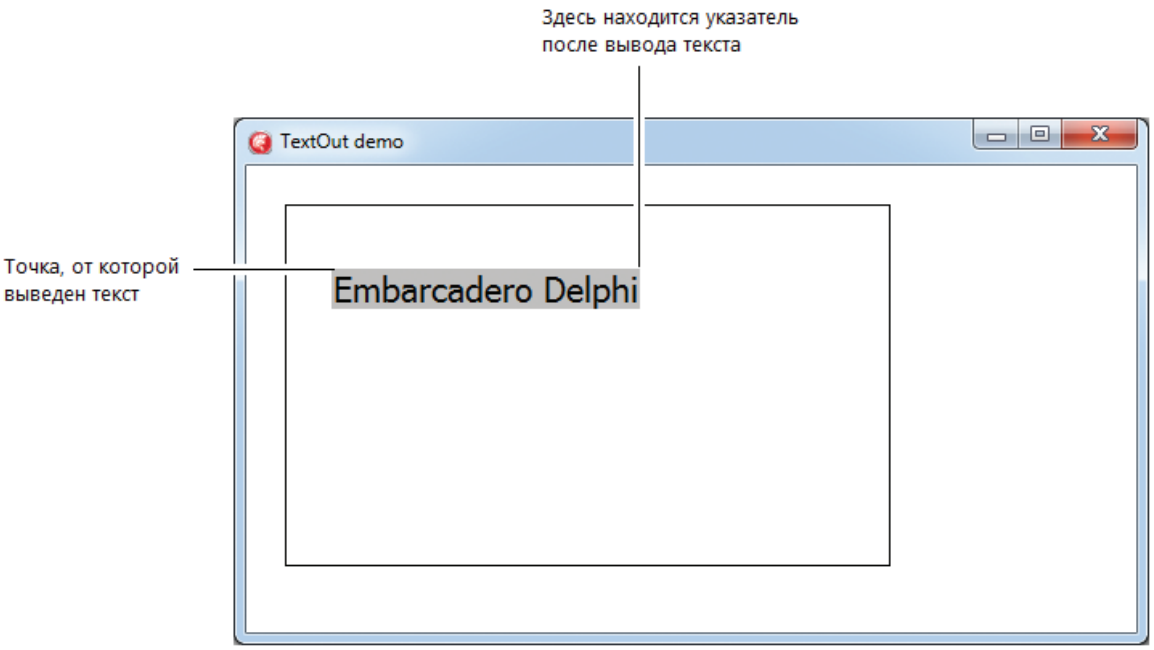


Рисунок 3.2. Координаты области вывода текста

Шрифт, используемый для отображения текста, определяет свойство Font графической поверхности, на которую текст выводится. Свойство Font представляет собой объект типа TFont. В табл. 3.6 перечислены свойства класса TFont.

Таблица 3.6. Свойства класса TFont

Свойство	Определяет
Name	Шрифт, которым отображается текст. В качестве значения следует указать 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.

Следующий фрагмент кода (Листинг 3.2) демонстрирует использование методов TextOut, TextWidth и TextHeight для вывода текста на поверхность формы. Приведенная процедура обработки события Paint выводит в центре (по горизонтали) компонента PaintBox две строки текста шрифтом разного размера (рис. 3.3).

Листинг 3.2. Вывод текста на графическую поверхность

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

var
  x,y: integer;
  st: string;

begin

  // показать границу PaintBox1
  PaintBox1.Canvas.Rectangle(0,0,PaintBox1.Width,PaintBox1.Height);

  st := 'Embarcadero Delphi';
  y :=10;

  PaintBox1.Canvas.Font.Size := 20;

  x:=  (PaintBox1.Width - PaintBox1.Canvas.TextWidth(st)) div 2;
  PaintBox1.Canvas.TextOut(x,y,st);

  // координата нижней границы области вывода первой строки
  y:= Paintbox1.Canvas.PenPos.Y +  PaintBox1.Canvas.TextHeight(st);

  PaintBox1.Canvas.Font.Size := 12;
  x:=  (PaintBox1.Width - PaintBox1.Canvas.TextWidth(st)) div 2;
  PaintBox1.Canvas.TextOut(x,y,st);

end;
```

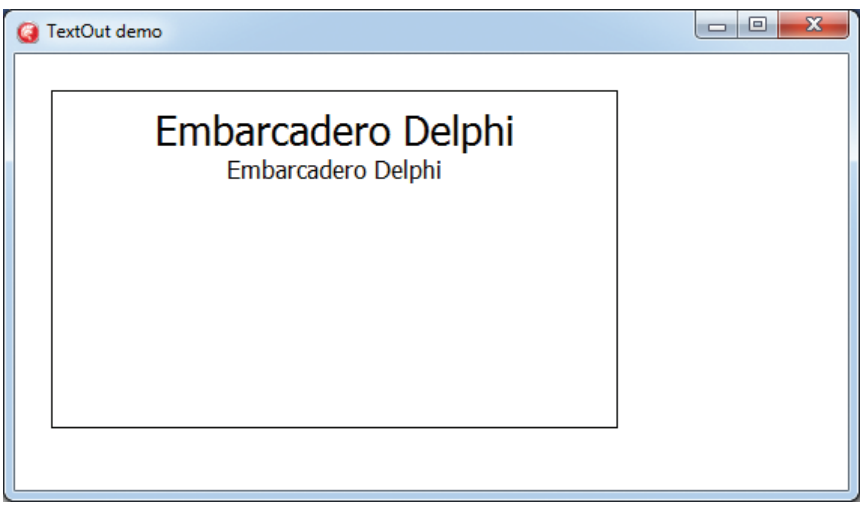


Рисунок 3.3. Вывод текста на графическую поверхность

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 остановится и будет находиться в этой конечной точке.

В качестве примера в листинге 3.3 приведен фрагмент кода, который демонстрирует различные виды линий (рис. 3.4). Необходимо обратить внимание на то, что пунктиром можно нарисовать только линию толщиной в один пиксел.

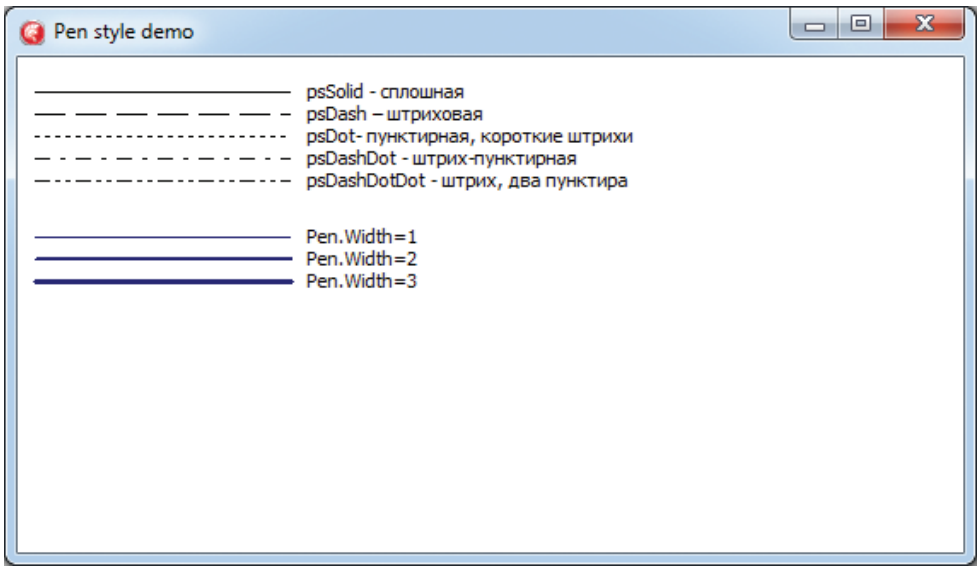


Рисунок 3.4. Вид линии определяет значение свойства Pen.Style

Листинг 3.3. 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 3.4, строит в поле компонента PaintBox график изменения курса доллара (рис. 3.5). Данные загружаются из файла kurs.txt (Листинг 3.5), который должен находится в папке **Мои документы**. Для получения полного имени папки используется функция GetDocumentsPath. Следует обратить внимание на то, что более «свежая» информация находится в начале файла (предполагается, что данные добавляются в начало файла). Поэтому график строится не от первой точки (элемента массива), а от последней. Т.к. разница между минимальным и максимальным значениями ряда данных, отображаемых на графике, незначительна, то 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.

Рисунок 3.5. График

Листинг 3.4. График

```
var
  path: string; // путь к файлу данных
  dataFile: string;
  kurs: array[1..15] of real; // массив данных
  nrec: integer; // количество чисел, прочитанных из файла
procedure TForm1.FormActivate(Sender: TObject);
var
  f: TextFile;
begin
  // Внимание! В директиву uses надо добавить ссылку на
  // модуль System.IOUtils
  path:= System.IOUtils.TPath.GetDocumentsPath();
  dataFile := path + '\kurs.txt';
  AssignFile(f, dataFile);
  try
    reset(f); // открыть файл для чтения
    nrec:=0;
    while (NOT EOF(f)) AND (nrec < 7) do
      begin
        nrec := nrec +1;
        readln(f,kurs[nrec]);
      end;
  except on EInOutError do
    //MessageDlg('Нет файла данных ' + dataFile, mtError,[mbOK],0);
  end;
end;
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    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 := 10;
      PaintBox1.Canvas.TextOut(10,10,'Ошибка! Нет файла данных ' + dataFile);
      // данные из файла не прочитаны
      exit;  // выход из процедуры
    end;
    // строим график
    with PaintBox1.Canvas do begin
      // заголовок
      Font.Name := 'Tahoma';
      Font.Size := 16;
      x := Round( (PaintBox1.Width - TextWidth('Курс доллара')) /2);
      PaintBox1.Canvas.Font.Color := clGreen;
      Brush.Style := bsClear;
      TextOut(x,10,'Курс доллара');
      PaintBox1.Canvas.Font.Color := clBlack;
      // найти минимальное и максимальное значение 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;
        {  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) / (nrec-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 := nrec downto 1 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 <> nrec) 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;
        { т.к. 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 3.5. Файл данных программы График (kurs.txt)

```
41.0450
40.7457
40.9416
40.5304
40.3251
40.2125
39.9800
39.9819
39.7417
39.9820
```

Ломаная линия

Метод Polyline чертит ломаную линию или, если координаты первой и последней точек массива совпадают, контур.

Инструкция вызова метода Polyline 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. 3.6), который можно рассматривать, как ломаную, состоящую из трех звеньев (номера точек ломаной соответствуют индексам элементов массива).

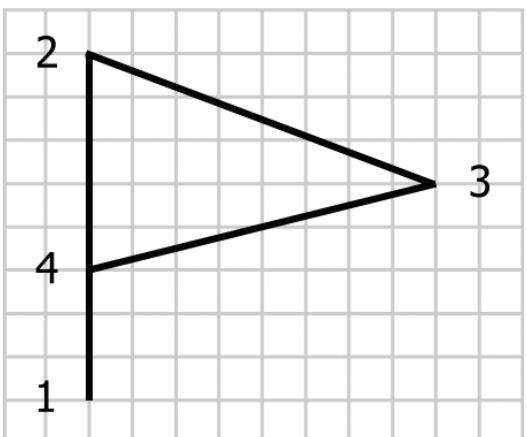


Рисунок 3.6. Пример ломаной линии

```
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 3.6 приведена процедура обработки события Paint, которая на поверхности формы рисует итальянский и французский флаги (рис. 3.7).

Рисунок 3.7. Метод Rectangle рисует прямоугольник

Листинг 3.6. 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 TopLeft 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. Следующая программа, ее текст приведен в листинге 3.7, демонстрирует использование методов FillRect и FrameRect.

Листинг 3.7. 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 RoundRect 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. 3.8).

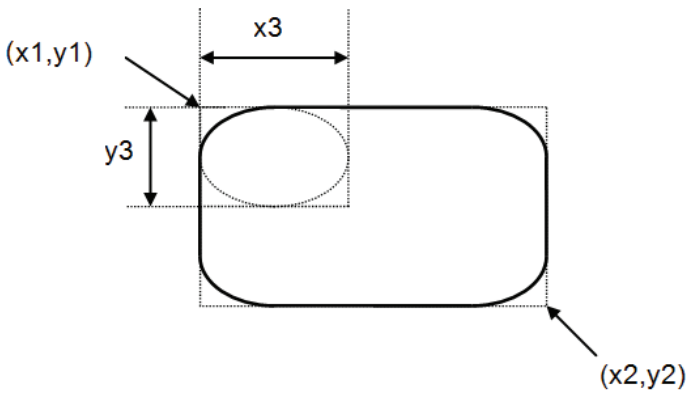


Рисунок 3.8. Метод RoundRect вычерчивает прямоугольник со скругленными углами

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 – координата X и координата 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 3.8 приведена программа, которая в виде полигона показывает динамику изменения курса доллара (рис. 3.9). Данные загружаются из файла.

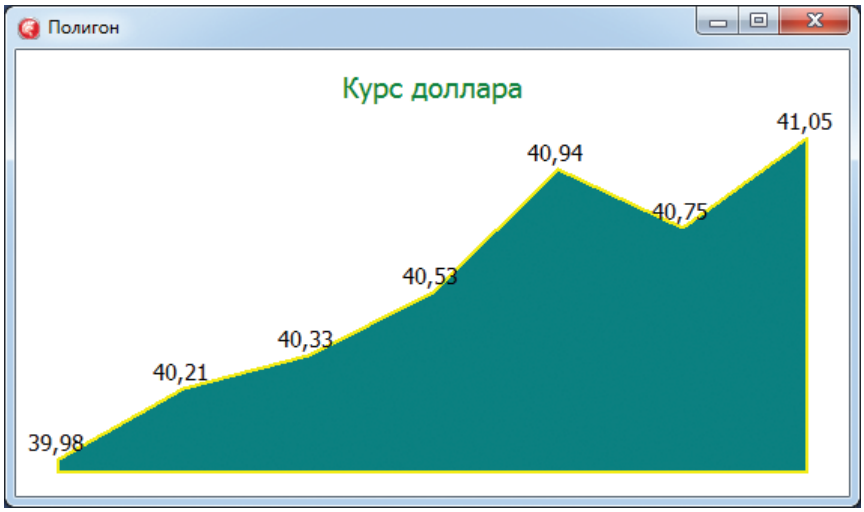


Рисунок 3.9. Полигон

Листинг 3.8. Полигон

```
var
  path: string; // путь к файлу данных
  dataFile: string;
  kurs: array[1..15] of real; // массив данных
  nrec: integer; // количество чисел, прочитанных из файла
procedure TForm1.FormActivate(Sender: TObject);
var
  f: TextFile;
begin
  // Внимание! В директиву uses надо добавить ссылку на
  // модуль System.IOUtils
  path:= System.IOUtils.TPath.GetDocumentsPath();
  dataFile := path + '\kurs.txt';
  AssignFile(f, dataFile);
  try
    reset(f); // открыть файл для чтения
    nrec:=0;
    while (NOT EOF(f)) AND (nrec < 7) do
      begin
        nrec := nrec +1;
        readln(f,kurs[nrec]);
      end;
  except on EInOutError do
    //MessageDlg('Нет файла данных ' + dataFile, mtError,[mbOK],0);
  end;
end;
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
  min,max: integer; // мин. и макс. значения ряда данных
  p: array[0..8] of TPoint; // координаты точек полигона
  // p[0] - левый нижний угол полигона,
  // p[1]..p[7] - углы, изображающие данные
  // p[8] - правый нижний угол полигона
```

```
    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 := 14;
      Font.Color := clGreen;
      x := Round( (PaintBox1.Width - TextWidth('Курс доллара')) /2);
      Brush.Style := bsClear;
      TextOut(x,5,'Курс доллара');
            // найти минимальное и максимальное значение 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;
      // *** 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 := 20;
       // 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[nrec-i+1] - 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[nrec+1].X :=  PaintBox1.Width -20;
      p[nrec+1].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 := 11;
      Font.Color := clBlack;
      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[nrec-i+1],ffFixed,5,2);
                TextOut(p[i].X-20,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. 3.10). 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.

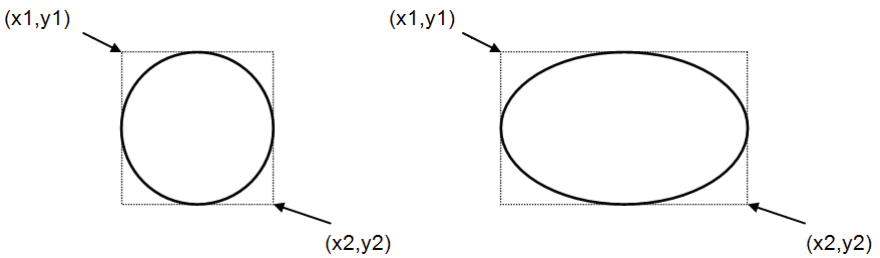


Рисунок 3.10. Значения параметров метода Ellipse определяют вид геометрической фигуры

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 Arc 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. 3.11).

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

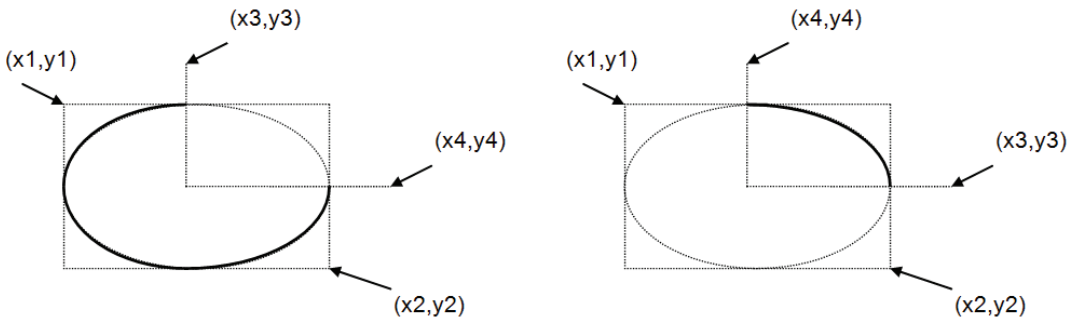


Рисунок 3.11. Значения параметров метода Arc определяют дугу как часть эллипса

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. 3.12).

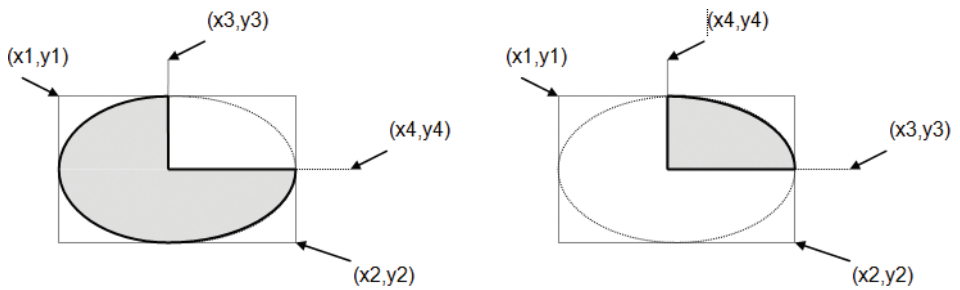


Рисунок 3.12. Значения параметров метода Pie определяют сектор как часть эллипса (окружности)

Следующая программа, ее окно приведено на рис. 3.13, демонстрирует использование метода Pie для построения круговой диаграмма.

Рисунок 3.13. Круговая диаграмма

Текст процедур приведен в листинге 3.9. Загрузку данных из файла energy.txt (Листинг 3.10) выполняет процедура обработки события Activate формы. Процедура preparation выполняет предварительную обработку данных: сортирует данные по возрастанию и вычисляет долю каждой категории в общей сумме. Строит диаграмму процедура обработки события Paint компонента PaintBox.

Листинг 3.9. Круговая диаграмма

```
const
  R = 80;  // радиус диаграммы
  HB = 6;  // количество категорий данных
var
  path: string; // путь к файлу данных
  dataFile: string;
  // 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, clMo-
neyGreen);
  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 i-ый и i+1-ый 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
  path:= System.IOUtils.TPath.GetDocumentsPath();
  dataFile := path + '\energy.txt';
  AssignFile(f, dataFile);
  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 осью 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-ти 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 3.10. Файл данных (energy.txt)

```
Источники энергии
Другие
0.5
Гидро электростанции
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 и сохранить в bmp формате.

Битовый образ — это объект типа `TBitmap`. Некоторые свойства класса TBitmap приведены в табл. 3.7.

Таблица 3.7. 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 компонента Image

Перед тем как приступить к разработке программы, загружающей битовые образы из ресурса, рекомендуется в каталоге проекта создать папку **Images** и поместить в нее bmp фалы картинок. Затем в проект надо добавить картинки. Для этого в меню **Project** выбрать команду **Add To Project**, раскрыть папку **Images** и выбрать bmp файлы. В результате этих действий в проект будет добавлена ссылка на папку **Images** и будет создан файл ресурсов, в который будут помещены картинки. Теперь, если в окне **Project Manager** выбрать файл картинки (рис. 3.14), то в окне **Object Inspector** будет отображаться подробная информация о ресурсе (рис. 3.15), в том числе имя ресурса, которое надо будет указать в методе загрузки битового образа из ресурса.

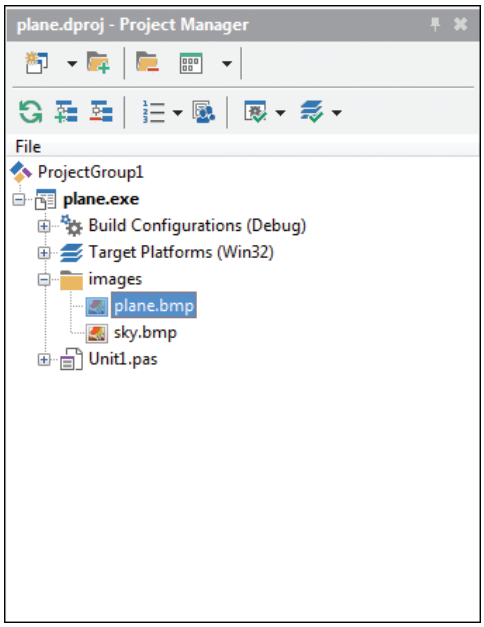


Рисунок 3.14. Результат добавления в проект ссылок на файлы картинок

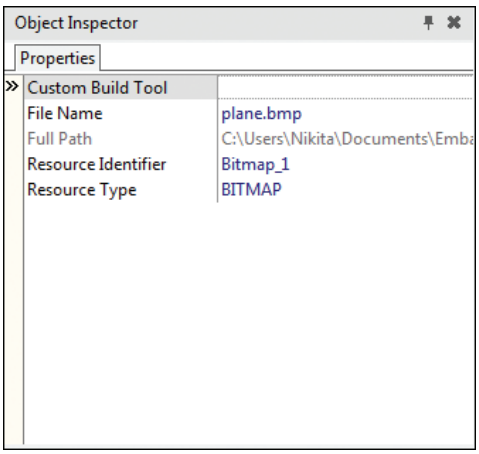


Рисунок 3.15. В окне Object Inspector отображается информация о ресурсе

Загрузку битового образа из ресурса обеспечивает метод LoadFromResourceName, которому в качестве параметра передается находящейся в глобальной переменной HInstance идентификатор приложения и имя ресурса. Например, следующий фрагмент создает битовый образ и загружает картинку из ресурса.

```
Plane := TBitmap.Create;
Plane.LoadFromResourceName(HInstance,'Bitmap_1');
```

После того как битовый образ загружен, его можно вывести на графическую поверхность, например, компонента PaintBox или формы. Вывод битового образа на графическую поверхность обеспечивает метод Draw. В качестве параметров метода указывают координаты левой верхней точки области, в которой надо отобразить битовый образ, и сам битовый образ. Например, инструкция

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

выводит на поверхность формы битовый образ Plane.

Если свойству Transparent битового образа присвоить значение True, то фрагменты картинки битового образа, окрашенные цветом, совпадающим с цветом левой нижней точки битового образа, не будут выведены на графическую поверхность. Такой прием используется для создания эффекта прозрачности. «Прозрачный» цвет можно задать и напрямую, присвоив значение свойству Transparent.Color.

Следующая программа, ее окно приведено на рис. 3.16, демонстрирует использование битовых образов для формирования сложных изображений.

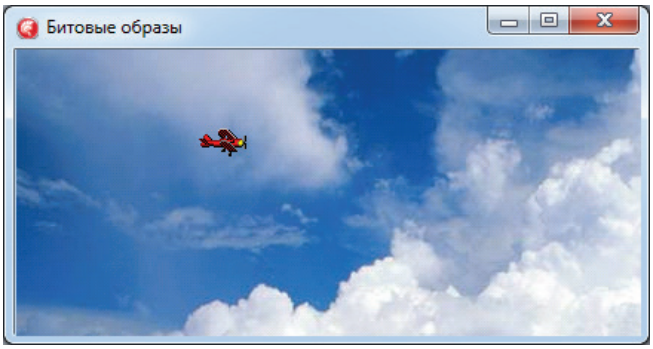


Рисунок 3.16. Изображение неба и самолета — битовые образы, загруженные из файлов

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 implementation. Процедуры обработки событий и объявления битовых образов приведены в листинге 3.11.

Листинг 3.11. Загрузка и отображение битовых образов

```
var
  // битовые образы
    Sky: TBitmap;    // небо
    Plane: TBitmap;  // самолет
// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  // создать два объекта TBitmap
  // и загрузить в них картинки
  Plane := TBitmap.Create;
  Plane.LoadFromResourceName(HInstance,'Bitmap_1');
  Plane.Transparent := True; // прозрачный фон
  Sky := TBitmap.Create;
  Sky.LoadFromResourceName(HInstance,'Bitmap_2');
  // установить размер окна в соответствии
  // с размером фоновой картинки
  Form1.ClientWidth:= Sky.Width;
  Form1.ClientHeight := Sky.Height;
```

```
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;
```

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 предыдущем положением объекта, можно добиться того, что у наблюдателя будет складываться впечатление, что объект равномерно движется.

Следующая программа, в ее окне «плывет» корабль (рис. 3.17), демонстрирует описанный метод реализации анимации.

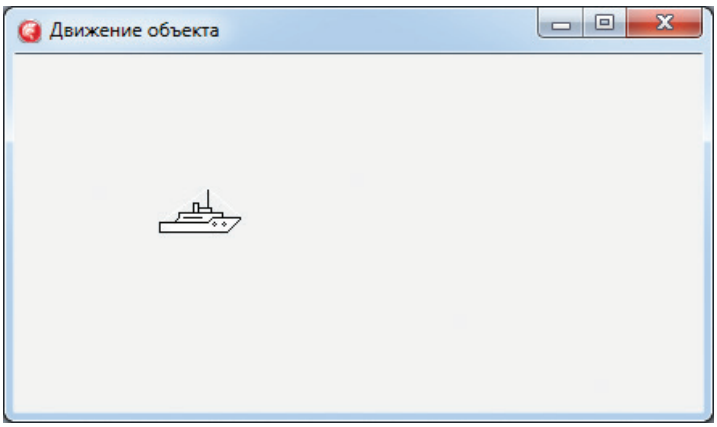


Рисунок 3.17. В окне программы «плывет» кораблик

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. 3.18). 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.

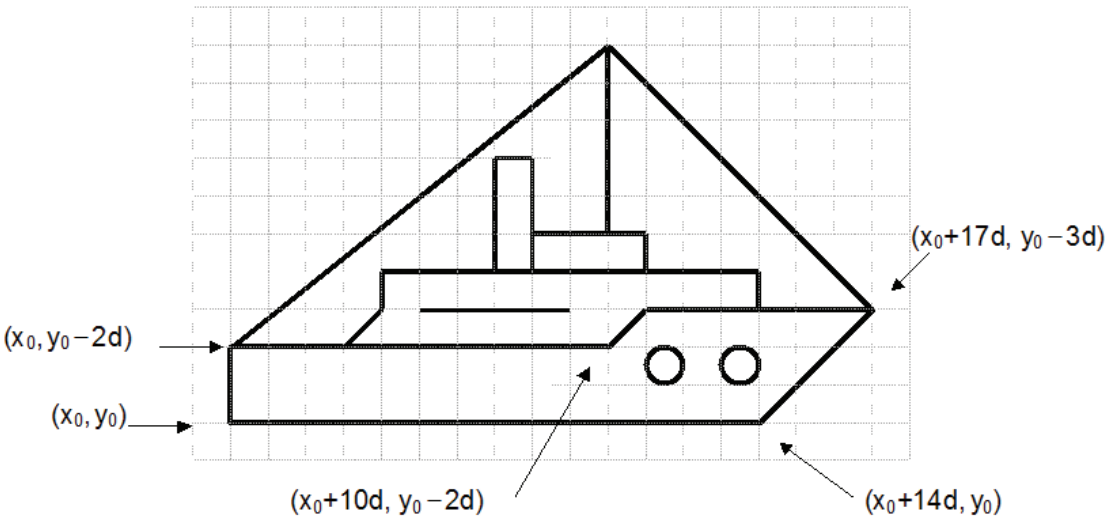


Рисунок 3.18. Базовая точка (x_0,y_0) определяет положение объекта

На рис. 3.19 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).

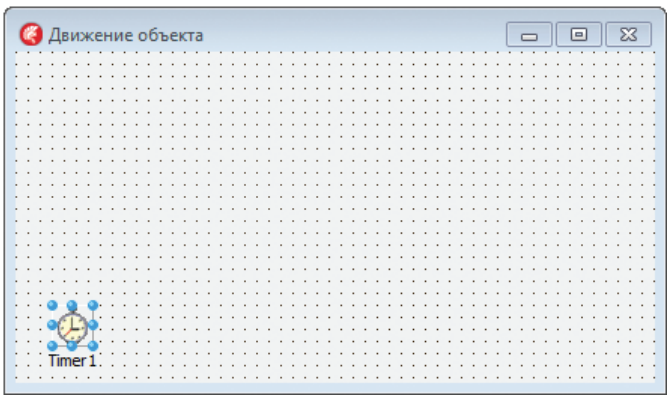


Рисунок 3.19. Форма программы

В листинге 3.12 приведена процедура Ship, которая рисует объект, процедура обработки события Timer, заставляющая объект двигаться, и процедура обработки события Create формы.

Листинг 3.12. 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;
  // рисуем ...
  // 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. 3.20). В начале игры в окне программы отображаются два объекта: летящий самолет и пусковая установка. Игрок, нажав пробел, может запустить ракету, которая собьет самолет, если момент запуска выбран правильно. Также с помощью клавиш перемещения курсора, игрок может сместить пусковую установку влево или вправо. Текст программы приведен в листинге 3.13.

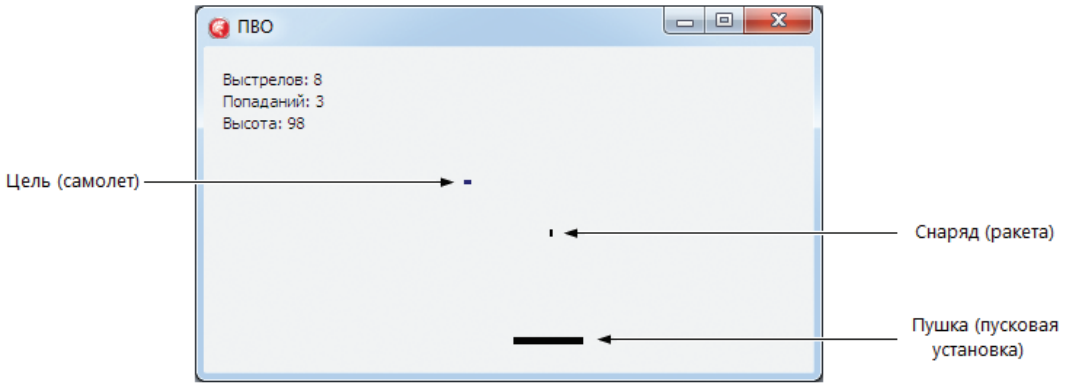


Рисунок 3.20. Игра ПВО

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 нажат пробел, но ракета еще 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 3.13. 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 от 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 достаточно очевиден: чтобы сформировать более-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 за счет использования битовых образов.

Как и в предыдущих программах, эффект движения (полет самолета) достигается за счет 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 была «испорчена» на предыдущем шаге, перекрыта объектом. В рассматриваемой программе используется второй способ.

Форма программы **Полет в облаках** содержит только один компонент – таймер. Объявления битовых образов и процедуры обработки событий приведены в листинге 3.14.

Листинг 3.14. Полет в облаках

```
var
    back: TBitmap;  // фон
    plane: TBitmap; // объект
    x,y: integer;   // координаты объекта
// конструктор
procedure TForm1.FormCreate(Sender: TObject);
begin
  try
    // создать два объекта TBitmap
    // и загрузить в них картинки
    plane := TBitmap.Create;
    plane.LoadFromResourceName(HInstance,'Bitmap_1');
    plane.Transparent := True; // прозрачный фон
    back:= TBitmap.Create;
    back.LoadFromResourceName(HInstance,'Bitmap_2');
    // установить размер окна в соответствии
    // с размером фоновой картинки
    Form1.ClientWidth:= back.Width;
    Form1.ClientHeight := back.Height;
    // исходное положение объекта
    x :=  -30;
    y :=  70;
    Timer1.Interval := 25;
   Timer1.Enabled := True;
  finally
  end;
end;
// сигнал от 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) из ресурса, 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 от 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 от 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. 3.21). 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.

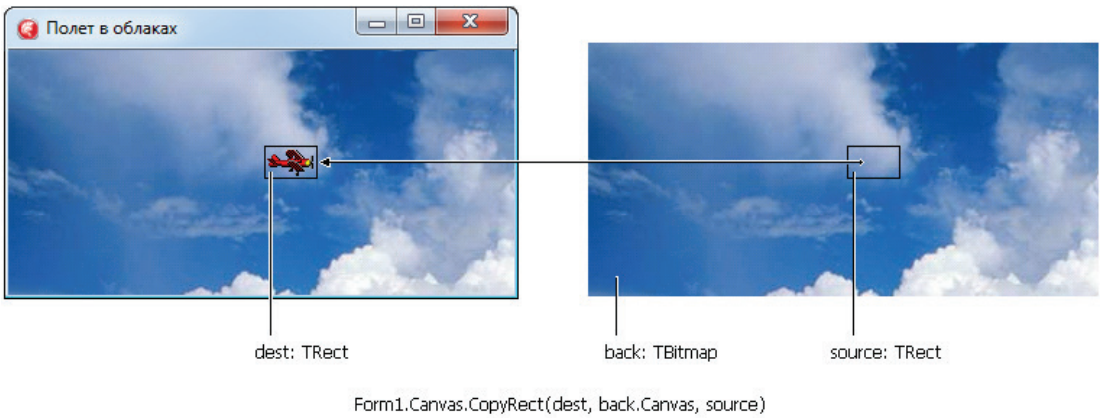


Рисунок 3.21. Восстановление фона перед перерисовкой объекта на новом месте обеспечивает метод CopyRect

Запустив программу Полет в облаках, можно заметить, что изображение самолета мерцает. Это объясняется тем, что глаз успевает заметить, как самолет исчезает и

появляется снова. Чтобы устранить мерцание, надо чтобы самолет не исчезал, а смещался. Добиться этого можно, если формировать изображение не на поверхности формы, а на невидимой для пользователя графической поверхности, и затем выводить готовое изображение на поверхность формы. Рис. 3.22 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 3.15 программа демонстрирует реализацию описанного метода формирования изображения.

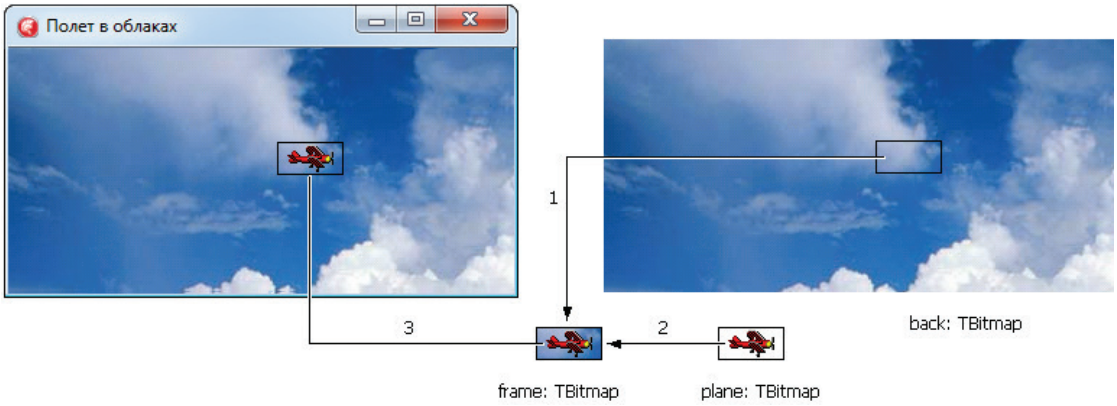


Рисунок 3.22. Формирование и отображение кадра

Листинг 3.15. Формирование изображения на невидимой поверхности

```
var
    back: TBitmap;  // фон
    plane: TBitmap; // объект (самолет)
    frame: TBitmap; // кадр = область фона + объект
    x,y: integer;   // координаты объекта
// конструктор
procedure TForm1.FormCreate(Sender: TObject);
begin
  try
    plane := TBitmap.Create;
    plane.LoadFromResourceName(HInstance,'Bitmap_1');
    plane.Transparent := True; // прозрачный фон
    back:= TBitmap.Create;
    back.LoadFromResourceName(HInstance,'Bitmap_2');
    Form1.ClientWidth:= back.Width;
    Form1.ClientHeight := back.Height;
    // исходное положение объекта
    x :=  -30;
    y :=  70;
    Timer1.Interval := 25;
    Timer1.Enabled := True;
  finally
  end;
end;
// сигнал от 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 4. 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 с данными, 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. Microsoft Access — это типичная локальная база данных.

В удаленных базах данных файлы данных размещают на отдельном, доступном по сети, компьютере (сервере). Хранение данных, манипулирование данными, решение задач администрирования (управления доступом) обеспечивает сервер баз данных, например, Borland InterBase, Microsoft SQL Server, MySQL, Oracle.

Программы работы с базами данных строят по технологии «клиент-сервер». Программа-клиент или просто клиент, работающая на компьютере пользователя, принимает команды (запросы) от пользователя, передает их серверу, получает и отображает данные. Серверная часть или сервер, работающая на удаленном компьютере (а в случае локальной базы данных, на том же что и клиент), принимает запросы от клиента, выполняет их и пересылает данные клиенту. Таким образом, разработка программы работы с 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 *реляционные* базы данных (англ. relation — отношение, связь). 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 таблицам 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 mdb-файле.

Механизмы доступа к данным

Существует достаточно много механизмов (технологий) доступа к данным (ADO, dbExpress, dbGo и др.).

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** и **InterBase**.

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.

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. Поэтому, перед тем как приступить к разработке программы работы с базой данных, необходимо создать эту базу данных с помощью, соответствующей СУБД.

База данных Microsoft Access

Процесс разработки программы работы с базой данных Microsoft Access рассмотрим на примере. Создадим программу работы 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). В рассматриваемом примере база данных Контакты представляет собой одну единственную таблицу contacts (табл. 4.1). Файл базы данных следует поместить в папку Документы, в отдельный каталог (например, contacts). Кроме того, в папке contacts надо создать папку Images (в ней будем хранить иллюстрации).

Таблица 4.1. Таблица contacts

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	Имя файла иллюстрации (фото)

Доступ к данным

Доступ к данным при использовании технологии ADO обеспечивают компоненты ADOConnection, ADODataSet, ADOTable и ADOQuery, которые находятся на вкладке **dbGo** (рис. 4.1).

Рисунок 4.1. Компоненты доступа 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 хранит данные, полученные из одной таблицы. Компонент ADOQuery содержит 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. 4.2.

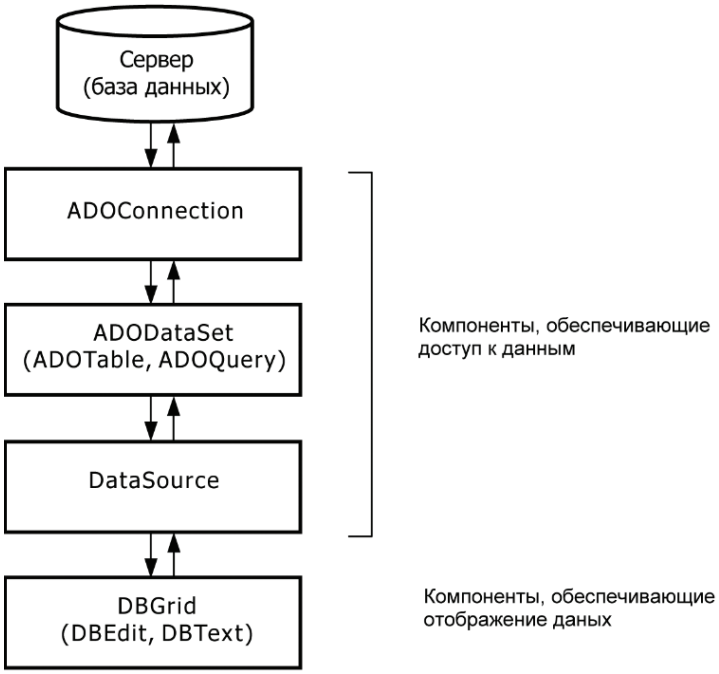


Рисунок 4.2. Взаимодействие компонентов, обеспечивающих доступ к данным и их отображение

Форма программы работы с базой данных Контакты приведена на рис. 4.3. 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.

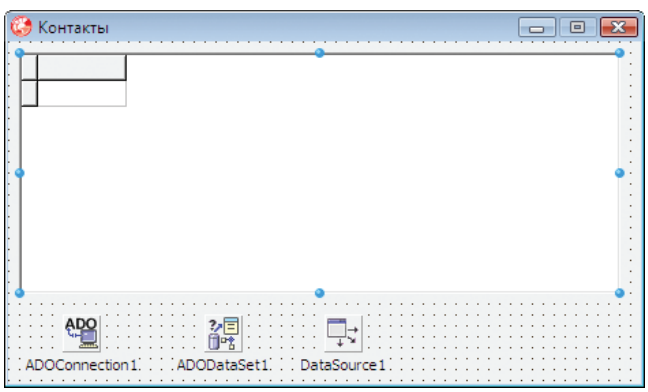


Рисунок 4.3. Форма программы работы с базой данных Контакты

Компонент ADOConnection, его свойства приведены в табл. 4.2, обеспечивает соединение с базой данных.

Таблица 4.2. 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, и в появившемся окне **Connection string** нажать кнопку **Build** (рис. 4.4).

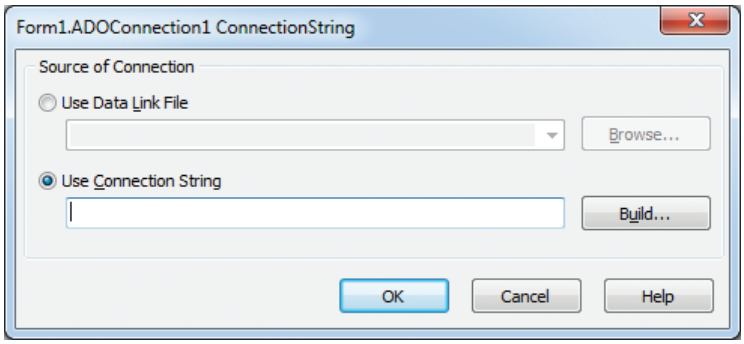


Рисунок 4.4. Настройка соединения с базой данных (шаг 1)

Затем на вкладке **Поставщик данных** окна **Свойства канала передачи данных** следует выбрать поставщика данных. Для базы данных Microsoft Access в качестве поставщика указать **Microsoft Jet OLE DB Provider** (рис. 4.5).

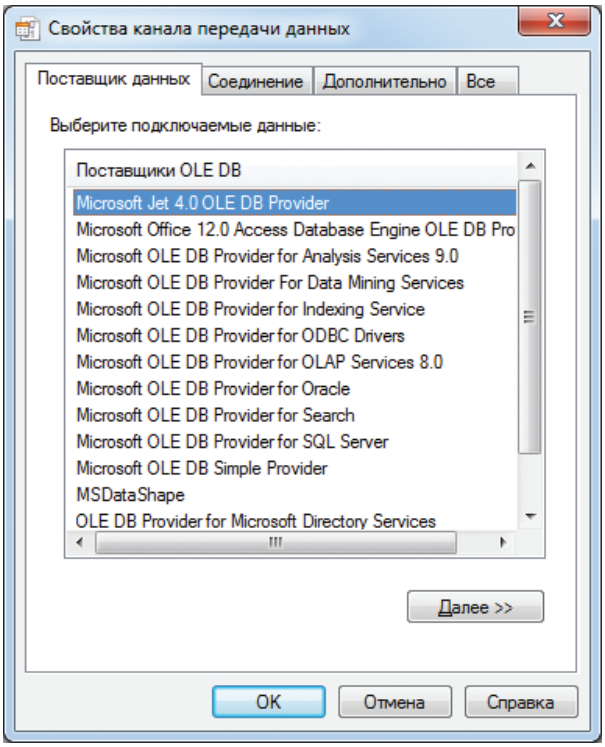


Рисунок 4.5. Настройка соединения с базой данных (шаг 2)

После выбора поставщика данных следует нажать кнопку **Далее** и на вкладке **Соединение** (рис. 4.6) задать базу данных — щелкнуть на кнопке с тремя точками и в раскрывшемся окне выбрать файл базы данных. Если для доступа к базе данных требуется пароль и идентификатор пользователя, то их надо ввести (по умолчанию доступ к базе данных, созданной в Microsoft Access, есть у пользователя Admin, в этом случае вводить пароль не нужно). Затем необходимо удостовериться, что соединение с базой данных настроено правильно, для этого сделайте щелчок на кнопке **Проверить соединение**, убедившись закройте окно **Свойства канала передачи данных** щелчком на кнопке **OK**.

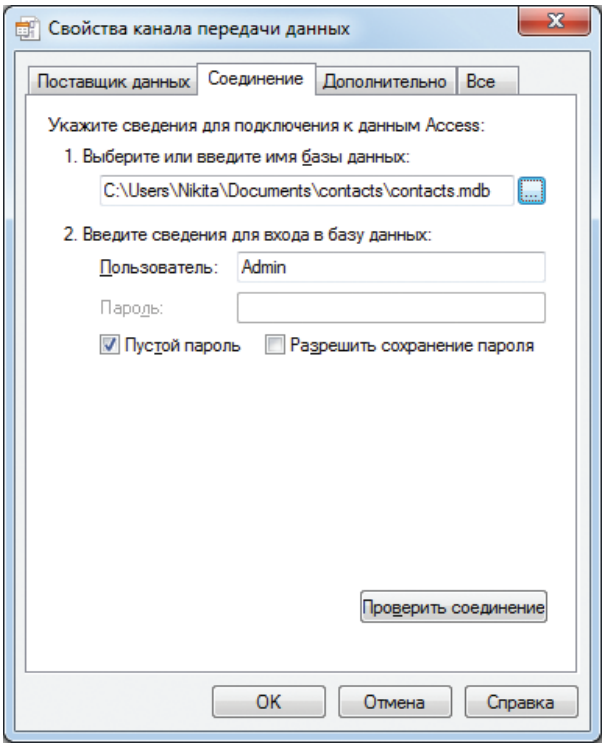


Рисунок 4.6. Настройка соединения с базой данных (шаг 2)

Значения свойств компонента ADOConnection1 после настройки соединения с базой данных Контакты приведены в табл. 4.3.

Таблица 4.3. Значения свойств компонента ADOConnection1

Свойство	Значение
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Users\Nikita\Documents\contacts\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. 4.4.

Таблица 4.4. Свойства компонента ADODataSet

Свойство	Описание
Connection	Ссылка на компонент (ADOConnection), обеспечивающий соединение с источником данных (базой данных)
CommandText	Команда, которая направляется серверу
Parameters	Параметры команды

Свойство	Описание
Filter	Фильтр. Позволяет отобрать записи, удовлетворяющие критерию отбора
Filtered	Признак использования фильтра
Activate	Открывает или делает недоступным набор данных

В базе данных contacts.mdb информация хранится в таблице contacts. Для того чтобы данные из этой таблицы попали в компонент ADODataSet, в свойство CommandText нужно записать SQL-команду, обеспечивающую выбор необходимой информации.

Выбор информации из таблицы базы данных обеспечивает SQL-команда SELECT. В качестве параметров команды SELECT нужно указать имя таблицы базы данных и список имен полей, содержимое которых надо получить (если нужна информация из всех полей, то вместо списка полей можно указать звездочку). Если информацию, выбранную из таблицы, требуется упорядочить, то в команде SELECT после слов ORDER BY следует указать поле по содержимому которого надобно упорядочить отобранные записи.

SQL-команда, обеспечивающая чтение данных из таблицы contacts, может выглядеть, например, так:

```
SELECT * FROM contacts ORDER BY name
```

Значения свойств компонента ADODataSet приведены в табл. 4.5.

Таблица 4.5. Значения свойств компонента ADODataSet

Свойство	Значение
Connection	ADOConnection1
CommandText	SELECT * FROM contacts ORDER BY name
Activate	False

Завершив настройку компонента ADODataSet приступить к настройке компонента DataSource — задать значение свойства DataSet, определяющего набор данных, связь с которым обеспечивает компонент (табл. 4.6).

Таблица 4.6. 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 в окне программы отображается в виде таблицы, что позволяет видеть одновременно несколько записей. Этот режим обычно используется для просмотра информации. В режиме формы в окне программы отображается одна запись. Обычно этот режим используется для ввода и редактирования информации. Часто эти два режима комбинируют: краткая информация (содержимое ключевых полей) выводится в табличной форме, а при необходимости видеть содержимое всех полей выполняется переключение в режим формы.

Отображение данных в виде таблицы обеспечивает находящийся на вкладке Data Controls компонент DBGrid (рис. 4.7).

Рисунок 4.7. Значок компонента DBGrid

Свойства компонента DBGrid (табл. 4.7) определяют вид таблицы и действия, которые могут быть выполнены над данными во время работы программы.

Таблица 4.7. Свойства компонента DBGrid

Свойство	Описание
DataSource	Ссылка на источник данных (например, ADODataSet)
Columns	Отображаемая информация (столбцы)
BorderStyle	Вид границы вокруг компонента
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 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 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.dg RowLines	Разрешает (True) выводить линии, разделяющие строки таблицы

Свойство Columns компонента DBGrid представляет собой коллекцию объектов типа TColumn. Свойства объекта TColumn (табл. 4.8) определяют информацию, которая отображается в колонке.

Таблица 4.8. 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)

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 компонента DbGrid выбрать команду **Columns Editor**. В окне редактора коллекции (рис. 4.8) сделать щелчок на кнопке **Add New**.

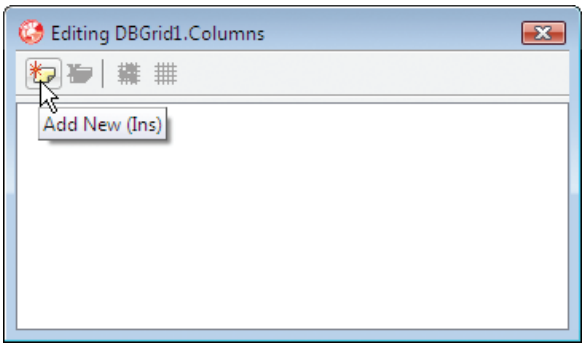


Рисунок 4.8. Чтобы добавить элемент в коллекцию Columns компонента DBGrid, надо сделать щелчок на кнопке **Add New**

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. 4.9 приведены значения свойств компонента DBGrid1, а на рис. 4.9 — вид формы после его настройки.

Таблица 4.9. Значения свойств компонента DBGrid1

Свойство	Значение
Font.Name	Tahoma
Font.Size	9
Columns[0].FieldName	name
Columns[0].Width	220
Columns[0].Title.Caption	Имя
Columns[0].Title.Font.Style	FsBold
Columns[1].FieldName	phone
Columns[1].Width	120

Свойство	Значение
Columns[1].Title.Caption	Телефон
Columns[1].Title.Font.Style	fsBold
Columns[2].FieldName	email
Columns[2].Title.Caption	E-Mail
Columns[2].Width	120
Columns[2].Title.Font.Style	fsBold

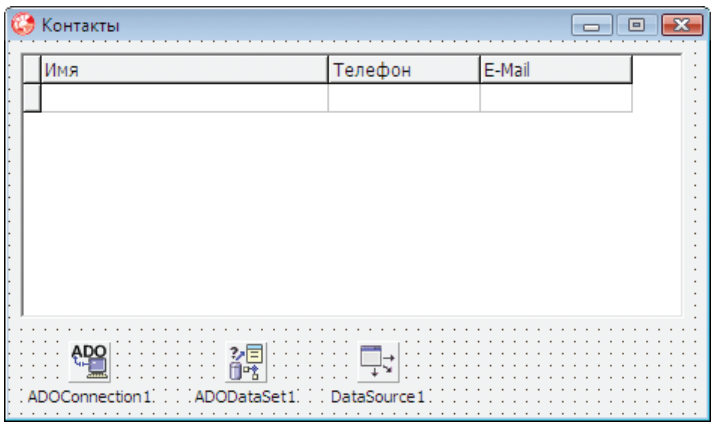


Рисунок 4.9. Форма программы работы с базой данных Контакты после настройки компонента DBGrid

После того как компоненты ADOConnection, ADODataSet, DataSource и DBGrid будут настроены, можно создать процедуры обработки событий Activate и Close формы (листинг 4.1).

Листинг 4.1. База данных Контакты

```
// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
var
   p,p2: integer;
begin
  try
    ADOConnection1.Open;
    ADODataSet1.Active := True;
  except
    on e:Exception do begin
        DBGrid1.Enabled := False;
        p := StrUtils.PosEx('Source=',ADOConnection1.ConnectionString,1) +7;
        p2 := PosEx(';',ADOConnection1.ConnectionString,p);
        MessageDlg(  'Нет Файла БД ' +
                    Copy(ADOConnection1.ConnectionString,p,p2-p),
                    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;
```

Процедура обработки события Activate формы приложения открывает базу данных, события Close — сохраняет изменения, сделанные пользователем. Здесь нужно обратить внимание на то, что все изменения, сделанные пользователем во время работы программы, автоматически фиксируются в базе данных (в файле) в момент перехода к следующей записи. Однако если пользователь, не завершив ввод или редактирование текущей записи, закроет окно программы, то изменения в редактируемой записи не будут записаны в файл. Поэтому, перед тем как завершить работу программы, следует 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.

Выбор информации из базы данных

При работе с базой данных пользователя, как правило, интересует не все ее содержимое, а только некоторая конкретная информация. В простейшем случае найти нужные сведения можно, просмотрев таблицу. Однако такой способ поиска неудобен и малоэффективен.

Чтобы выбрать необходимую информацию из базы данных следует, направить серверу SQL-команду SELECT или, если информация уже загружена из базы данных, активизировать фильтр.

SQL-запрос

Для того 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 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 = ‘Никита Культин’

обеспечивает выборку из таблицы contacts записи, у которой в поле name находится текст Никита Культин.

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 «равно» надо использовать оператор LIKE. Например, запрос

```
SELECT * FROM contacts WHERE name LIKE 'Ку%'
```

выберет из таблицы Phones 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'
```

Следующая программа, ее форма приведена на рис. 4.10, демонстрирует использование SQL-запроса для поиска информации в базе данных.

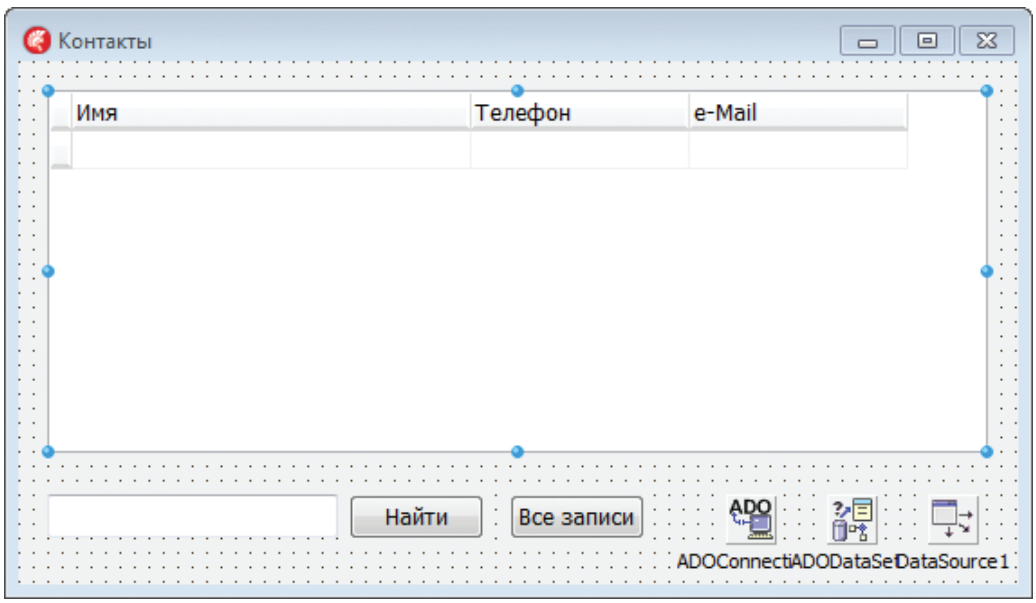


Рисунок 4.10. Форма программы работы с базой данных Контакты

Завершив настройку компонентов, можно приступить к созданию процедур обработки событий. Процедуры обработки событий главной формы такие же, как и в предыдущей программе. Процедуры обработки событий Click на кнопках **Найти** и **Все записи** приведены в листинге 4.2.

Листинг 4.2. Выбор информации из БД

```
// щелчок на кнопке Найти
procedure TForm1.Button1Click(Sender: TObject);
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
  Edit1.Clear;
  ADODataset1.Close;
  ADODataset1.CommandText :=
               'SELECT * FROM contacts ORDER BY Name';
  ADODataset1.Open;
end;
```

Фильтр

Часто нужная информация уже есть в загруженной таблице. В этом случае 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. Отличие фильтрации от выборки записей командой SELECT состоит в том, что фильтр воздействует на записи, уже загруженные из базы данных, 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 4.3 приведены процедуры обработки событий Click для кнопок **Найти** и **Все записи** программы работы с базой данных Контакты.

Листинг 4.3. Выбор информации (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;
// щелчок на кнопке Все записи
procedure TForm1.Button2Click(Sender: TObject);
begin
  ADODataSet1.Filtered := False;
end;
```

Работа с базой данных в режиме формы

Как было сказано раньше, 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 менять 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, находятся на вкладке **Data Controls** (рис. 4.11).

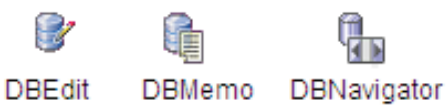


Рисунок 4.11. Компоненты, обеспечивающие редактирование полей и навигацию по БД

Компоненты DBEdit и DBMemo являются аналогами компонентов Edit и Memo, но они ориентированы на работу с базами данных.

Свойства компонентов DBEdit и DBMemo приведены в табл. 4.10.

Таблица 4.10. 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. Форма программы работы с БД Контакты приведена на рис. 4.12.

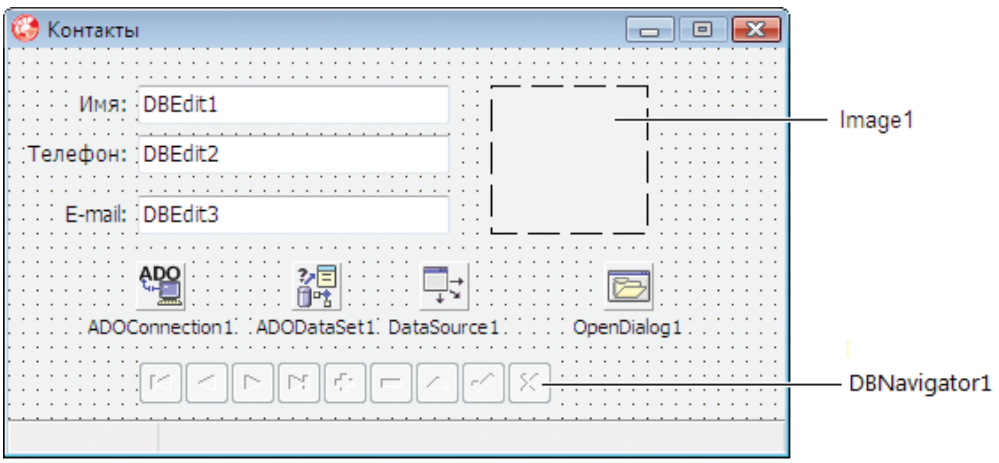


Рисунок 4.12. Форма программы работы с базой данных **Контакты** (режим формы)

Компоненты DBEdit обеспечивают отображение полей текущей записи, компонент Image — отображение иллюстрации, имя файла которой находится в поле Image. Значения свойств компонентов DBEdit приведены в табл. 4.11.

Таблица 4.11. Значения свойств компонентов

Компонент	Свойство	Значение
DBEdit 1	DataSource	DataSource 1
	DataField	name
	AutoSelect	False
	ReadOnly	True
DBEdit 2	DataSource	DataSource 1
	DataField	phone
	AutoSelect	False
	ReadOnly	True
DBEdit 3	DataSource	DataSource 1
	DataField	email
	AutoSelect	False
	ReadOnly	True

В форме рассматриваемой программы отображается только одна запись базы данных (эта запись называется текущей). Для перехода к другой следующей или предыдущей записи, а также выполнения операций добавления, удаления и редактирования записей используется компонент DBNavigator.

Компонент DBNavigator представляет собой совокупность командных кнопок (табл. 4.12), в результате щелчка на которых выполняются соответствующие операции. Свойства компонента DBNavigator представлены в табл. 4.13.

Таблица 4.12. 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 4.13. Свойства компонента DBNavigator

Свойство	Определяет
DataSource	Источник данных. В качестве источника данных может выступать компонент ADODataSet, ADOTable или ADOQuery
VisibleButtons	Кнопки, которые отображаются в поле компонента. Скрыв некоторые кнопки, можно запретить выполнение соответствующих действий

Свойство VisibleButtons позволяет скрыть некоторые кнопки компонента DBNavigator и тем самым запретить выполнение соответствующих операций с базой данных. Например, присвоив значение False свойству VisibleButtons.nbDelete, можно скрыть кнопку nbDelete и тем самым запретить операцию удаление записей.

Значения свойств компонента DBNavigator1 приведены в табл. 4.14.

Таблица 4.14. Значения свойств компонента DBNavigator1

Свойство	Значение
DataSource	DataSource1
VisibleButtons.bnRefresh	False

Модуль формы программы работы с базой данных приведен в листинге 4.4. Процедура обработки события Activate извлекает из строки соединения путь к файлу базы данных и записывает его в переменную aPath, для того, чтобы процедура вывода иллюстрации знала, где находятся нужные файлы. Процедура обработки события AfterScroll для компонента ADODataSet, 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 Image текущей записи пустое, картинку nobody.jpg. В поля Name, Phone и Comment информация вводится обычным образом. Чтобы ввести имя файла иллюстрации в поле Image, пользователь должен сделать щелчок на компоненте 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 4.4. 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);
var
   st: string;
   p,p2: integer;
begin
   // получить из строки соединения имя каталога, в котором находится файл БД
    p := StrUtils.PosEx('Source=',ADOConnection1.ConnectionString,1) +7;
    p2 := PosEx(';',ADOConnection1.ConnectionString,p);
    // файл БД
    st :=  Copy(ADOConnection1.ConnectionString,p,p2-p);
    // путь к файлу БД (каталогу Images)
    aPath := ExtractFilePath(st);
  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(e.Message,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 4.5 процедура показывает, как это можно сделать. В данном примере из INI-файла (листинг 4.6) загружается не строка соединения, а только имя папки, в которой находится файл базы данных, после чего строка соединения формируется путем замены ее фрагмента.

Листинг 4.5. 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 находится 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 находится файл БД.
    // Ключ aPath находятся 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 4.6. INI-файл

```
[data]
aPath=D:\Database\
```

Глава 5. Multi-Device приложение

Исторически сложилось так, что каждая среда программирования ориентирована на разработку под определенную платформу на своем языке программирования. Поэтому, если возникает задача перенести приложение на другую платформу, то приходится переписывать приложение в другой среде и, как правило, на другом языке программирования. Развитие вычислительной техники, широкое распространение различных, конкурирующих между собой платформ и огромное количество разных устройств в рамках этих платформ привело к пониманию того, что для повышения эффективности процесса разработки программного обеспечения необходим универсальный инструмент, позволяющий создавать универсальные приложения, которые будут способны работать на всех, или почти на всех, существующих устройствах. Очевидно, что создать универсальное приложение в чистом виде нельзя. Однако можно создать приложение, которое с минимальными изменениями разворачивается на разные платформы. Именно такое приложение и называется Multi-Device приложением.

Процесс разработки Multi-Device приложения рассмотрим на примере приложения **Скидка**, с помощью которого можно сосчитать цену с учетом скидки, по известным значениям цены и скидки. Вид окна программы, работающей на Android устройстве приведен на рис. 5.1.

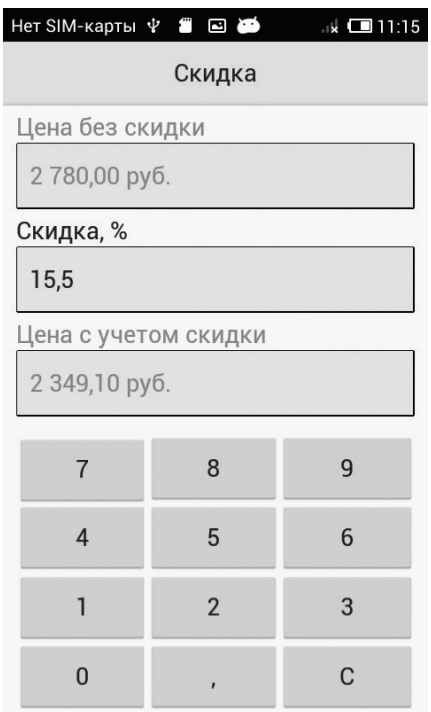


Рисунок 5.1. Окно программы **Скидка**

Начало работы над новым приложением

Чтобы начать работу над универсальным (Multi-Device) приложением, нужно в меню **File** выбрать команду **New - Multi-Device Application – Delphi** и затем, в появившемся окне (рис. 5.2), выбрать шаблон приложения. Шаблон приложения задает вид окон разрабатываемого приложения и механизм навигации между ними. Простейшими шаблонами являются шаблоны **Blanc Application Header/Footer.** Они предполагают, что разрабатываемое приложение будет однооконным. Естественно, в процессе разработки программист, если возникнет необходимость, сможет добавить нужное количество окон.

Рассматриваемое приложение не сложное, все элементы управления можно разместить в одном окне, поэтому в окне Multi-Device Application можно выбрать Header/Footer.

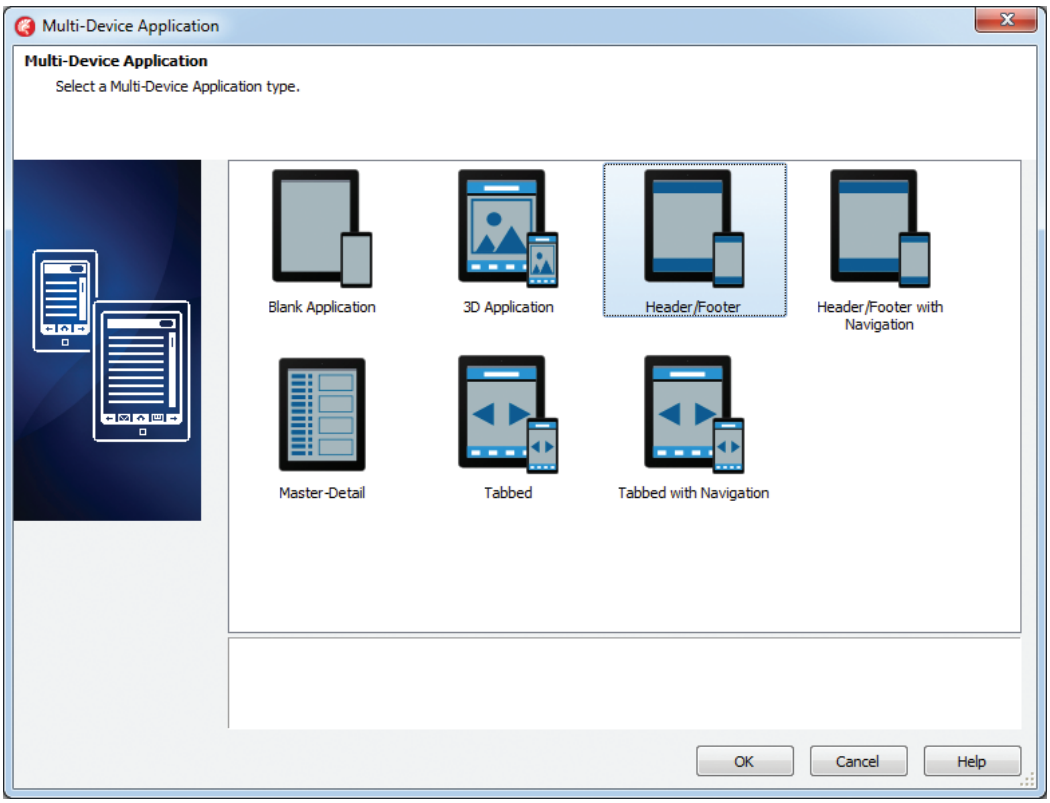


Рисунок 5.2. В окне надо выбрать шаблон (тип) приложения

После выбора типа приложения, в результате нажатия кнопки *OK*, становится доступным стандартное окно **Обзор папок**, в котором надо указать папку, предназначенную для проектов Delphi, нажать кнопку **Создать папку** и задать имя папки разрабатываемого приложения (рис. 5.3).

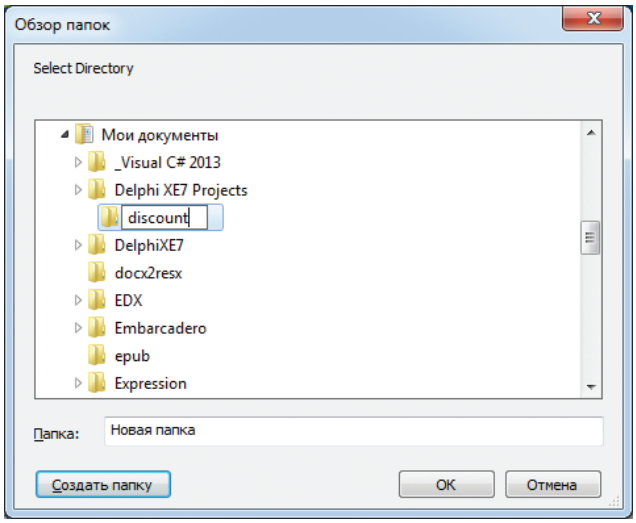


Рисунок 5.3. Создание папки для приложения

Вид окна Delphi XE7 в начале работы над новым универсальным приложением приведен на рис. 5.4.

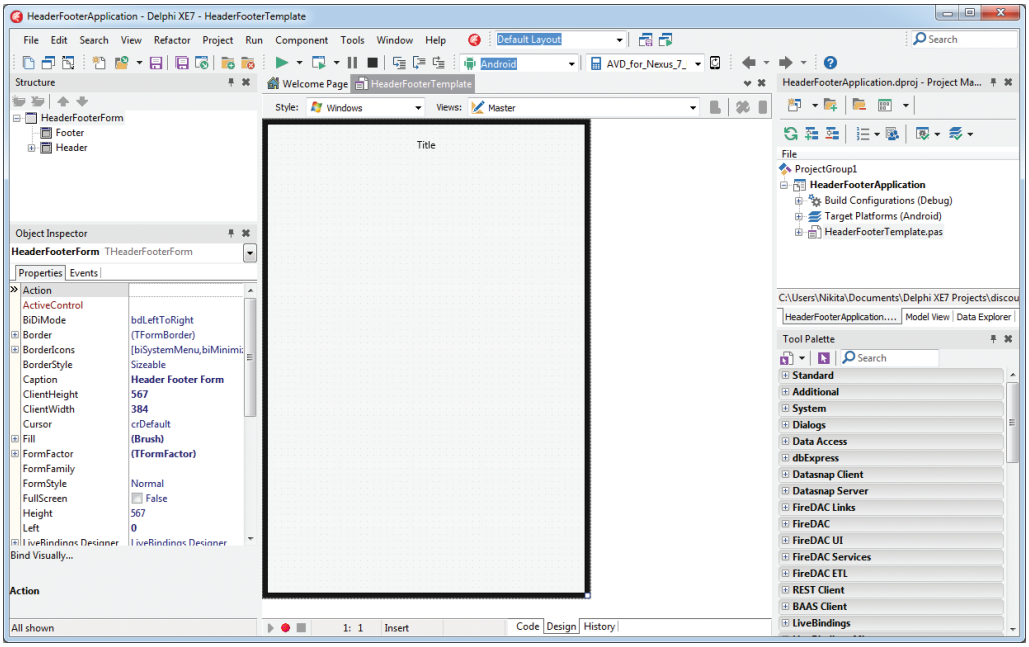


Рисунок 5.4. Окно Delphi в начале работы над новым проектом

В целом оно мало чем отличается от окна Delphi XE7 при работе над VCL Forms Application. Центральную часть окна занимает конструктор формы, за которым находится окно редактора кода. Окна **Object Inspector**, **Tool Palette**, **Project Manager** и **Structure также** на привычных местах.

Вместе с тем следует обратить внимание на панель инструментов **Platform Device**

Selection (рис. 5.5), в которой отображается текущая платформа и устройство, для которой в данный момент идет разработка.

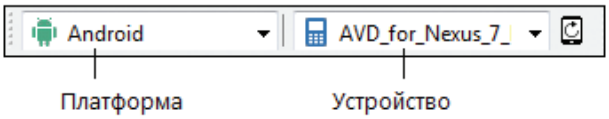


Рисунок 5.5. Панель Platform Device Selection

В списке платформ перечислены платформы (рис. 5.6), на которые может быть установлено разрабатываемое приложение, а в списке устройств – устройства, доступные для выбранной платформы (рис. 5.7).

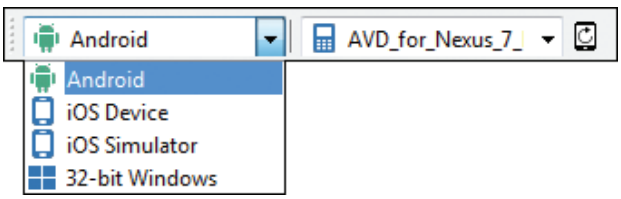


Рисунок 5.6. Список доступных платформ

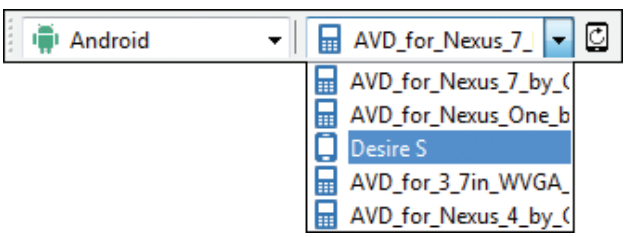


Рисунок 5.7. Список доступных устройств для платформы Android

Следует обратить внимание на то, что список доступных платформ определяется конфигурацией среды разработки, которая задается в процессе установки Delphi XE7 на компьютер разработчика, а список устройств – подключенными в данный момент к компьютеру реальными и виртуальными устройствами (перед именем виртуального Android устройства указывается префикс AVD – Android Virtual Device). Приведенный на рис. 5.7 список устройств показывает, что к компьютеру, помимо нескольких виртуальных Android устройств, подключено реальное Android устройство – смартфон HTS. Следует обратить внимание, для того чтобы устройство, подключенное к компьютеру, появилось в списке доступных устройств, на компьютер необходимо установить USB-драйвер этого устройства.

Форма и конструктор форм

Работа над универсальным (Multi-Device) приложением начинается с создания стартовой формы. Форма создается в окне конструктора форм (рис. 5.8) путем размещения на форме необходимых компонентов (элементов пользовательского интерфейса) и последующей их настройки.

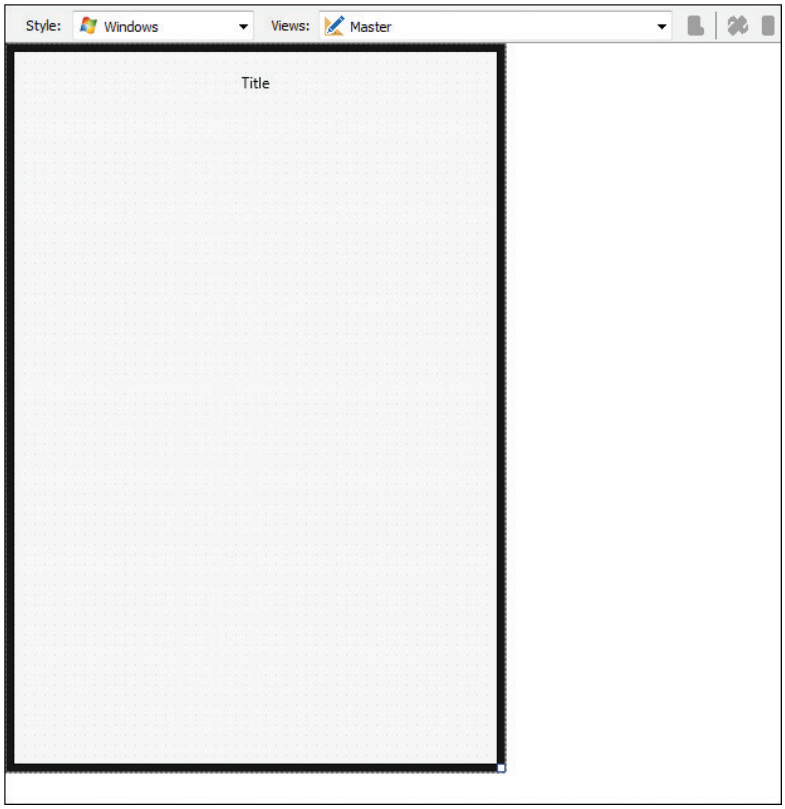


Рисунок 5.8. Конструктор форм

В верхней части окна конструктора форм находятся два списка. В поле редактирования списка Style отображается название платформы, соответствующее виду формы, отображаемому в данный момент. Выбрав в списке Style название нужной платформы (рис. 5.9), можно увидеть, как будет выглядеть окно программы на соответствующем устройстве. В поле редактирования списка Views отображается название вида для текущей платформы.

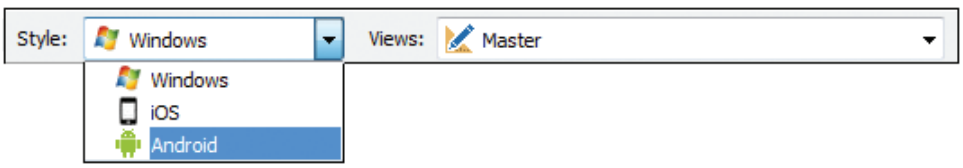


Рисунок 5.9. Чтобы увидеть, как будет выглядеть окно на другом устройстве, надо выбрать название платформы

По умолчанию для каждой платформы среда разработки создает один универсальный (Master) вид. Это предполагает, что на разных устройствах одной платформы компоненты пользовательского интерфейса будут одинакового размера и размещены они будут одинаково. Очевидно, что создать универсальную форму, которая будет выглядеть одинаково красиво как на устройстве с экраном размера 3.5 дюйма, так и на устройстве

с экраном 10 дюймов невозможно. Вместе с тем, для каждого устройства можно создать вид (View), учитывающий особенности (прежде всего размер экрана) этого устройства. В списке Views (рис. 5.10) перечислены устройства, для которых программист может создать специальный вид. Таким образом, конструктор позволяет создать форму, которая по составу компонентов для всех устройств платформы будет одинаковой, а по размещению и размеру компонентов у каждого устройства будет своей.

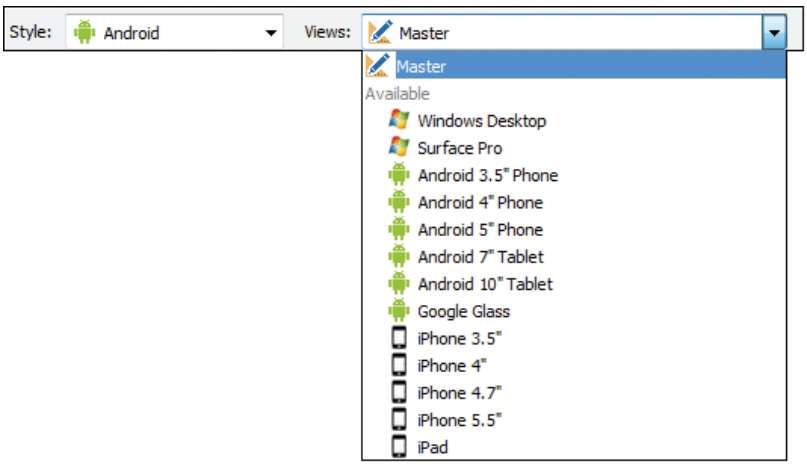


Рисунок 5.10. Список устройств, для которых можно создать специальный вид

Настройка формы, впрочем, как и других компонентов, осуществляется путем изменения значений свойств. Здесь следует обратить внимание на то, что при разработке Multi-Device приложения в окне Object Inspector отображаются все свойства формы, в том числе и те, которые в текущей (целевой) платформе не оказывают влияния на ее вид. В рассматриваемом примере настройка формы не требуется, достаточно в списке Style выбрать целевую платформу Android, и форма будет выглядеть так, как изображено на рис. 5.11.

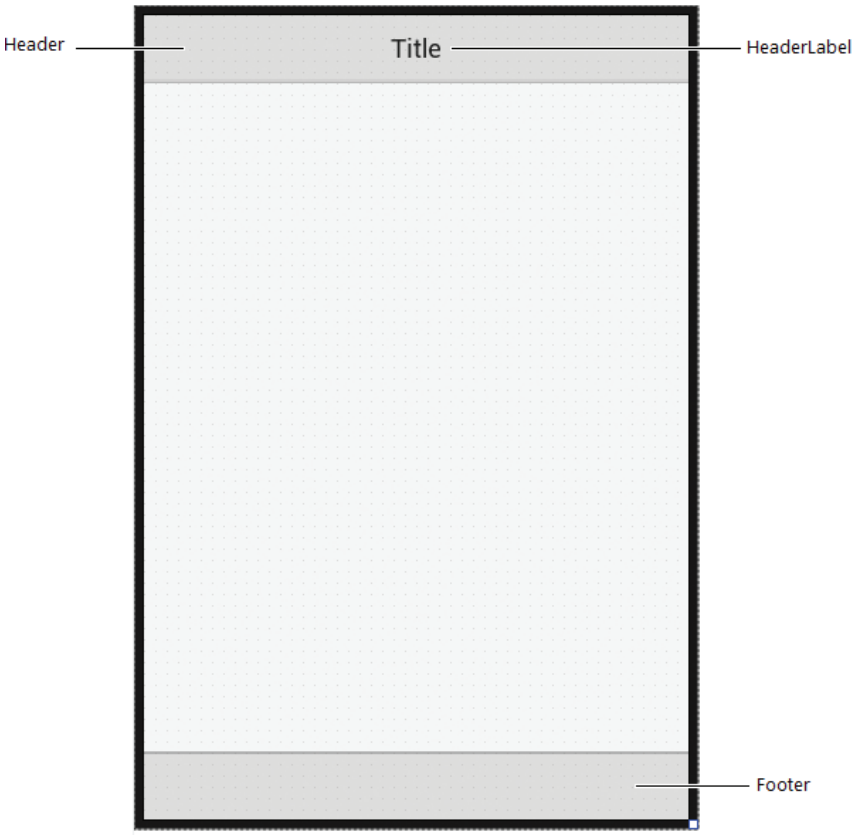


Рисунок 5.11. Форма для платформы Android

Обратите внимание, текст Title в верхней части формы это не заголовок, а текст в поле компонента HeaderLabel (TLabel), который находится на поверхности компонента Header (TToolBar). В нижней части формы находится еще один компонент TToolBar. Все эти компоненты являются элементами шаблона Header/Footer, выбранного в начале работы над проектом. Таким образом, чтобы изменить текст в заголовке формы, надо изменить значение свойства Text компонента HeaderLabel. Выбрать компонент HeaderLabel можно обычным образом, сделав щелчок в поле компонента, или в списке Structure (рис. 5.12).

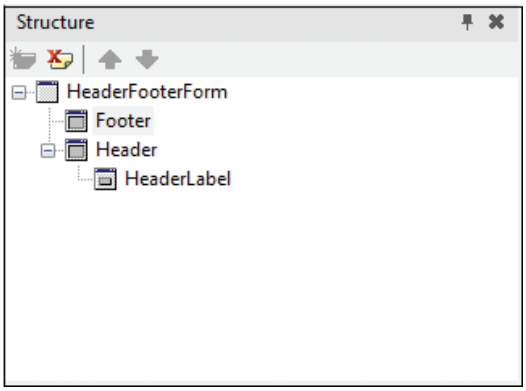


Рисунок 5.12. Окно **Structure**

FMX Компоненты

Компоненты, которые программист может использовать в своих приложениях, находятся в палитре компонентов. Ее вкладки и значки компонентов выглядят также, как и во время работы над VCL Forms приложением. Однако, если установить указатель мыши на какой-либо из компонентов, то в появившемся окне подсказки (рис. 5.13) d в префиксе имени модуля компонента будет указано FMX, а на VCL.

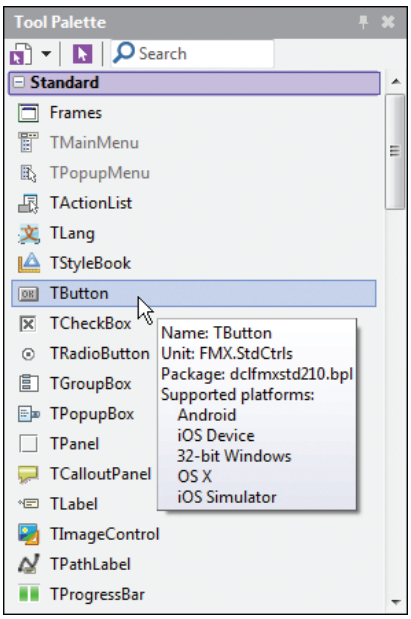


Рисунок 5.13. Multi-Device приложения строятся на основе FireMonkey (FMX) компонентов

FMX это – аббревиатура библиотеки компонентов FireMonkey, которая используется Multi-Device приложениями. Помимо названия модуля, в котором находится компонент, в окне подсказки перечисляются платформы, на которых компонент может использоваться. Следует обратить внимание, что некоторые компоненты могут работать не на всех платформах. Например, компонент WebBrowser (он находится на вкладке Internet) может использоваться только в Android и iOS приложениях.

С точки зрения использования различия между соответствующими VCL и FireMonkey компонентами несущественны. Вместе с тем у FireMonkey компонентов есть свойства, которых у VCL компонентов нет, которые весьма полезны при разработке приложений, ориентированных на разные устройства (в частности, с экранами разного размера) в рамках одной платформы. Это свойства Align, Anchors, Scale. Свойство Align (выравнивание) управляет размером и размещением компонента в поле родительского элемента. Например, если на форму поместить панель (компонент TToolBar) и присвоить ее свойству Align значение Bottom, то панель «прилипнет» к нижней границе формы и займет всю область формы по ширине. Свойство Scale позволяет масштабировать компонент, не меняя других его характеристик. Например, чтобы увеличить текст, отображаемый в поле компонента Label (строго говоря, сам компонент), достаточно изменить значения свойств Scale.X и Scale.Y, а не менять размер шрифта, используемого для вывода текста. Такой подход весьма эффективен для приложений,

которые потенциально могут работать на устройствах с различными размерами экранов: программа, например, в начале работы, может определить размер экрана и настроить масштаб компонентов формы так, чтобы она выглядела наилучшим образом.

Помимо обычных, визуальных компонентов, при создании форм удобно использовать компоненты-контейнеры, которые сами на форме не отображаются, но позволяют эффективно управлять размещением визуальных компонентов. Компоненты-контейнеры находятся на вкладке Layouts. Наиболее часто для размещения компонентов на форме используют компонент GridPanelLayout, представляющий собой таблицу, в ячейки которой можно поместить другие компоненты, и компонент Layout, позволяющий поместить один компонент на другой. Помимо компонентов GridPanelLayout и Layout также удобно использовать компонент Panel, позволяющий объединить в группу несколько компонентов и управлять как одним всеми компонентами, находящимися в панели. Компонент Panel находится на вкладке Standard.

Проектирование интерфейса

Так как во время создания формы универсального приложения, которое может работать на различных устройствах, неизвестны характеристики экрана устройства, на котором возможно будет запущено приложение, то использовать подход к размещению компонентов, заключающийся в их привязке к фиксированным точкам формы нельзя. Например, если во время создания формы программист будет ориентироваться на устройство с экраном 5 дюймов, то при запуске программы на устройстве с экраном 4 дюйма компоненты, расположенные у правой и нижней границ формы, не будут отображаться, а при запуске приложения на устройстве с экраном 7 дюймов справа и снизу области отображения компонентов будет пустое поле. Также надо учитывать, что во время работы приложения пользователь может изменить ориентацию устройства, что равносильно изменению характеристик экрана.

Для того чтобы приложение было действительно универсальным, при разработке интерфейса, следует использовать подход, который можно назвать плавающей верстkой. Суть этого подхода заключается в том, что компоненты формы привязываются не к координатам формы, путем задания значений свойств Position.X и Position.Y, а к ее границам или к другим компонентам.

Привязка компонента к форме или другим компонентам выполняется путем установки свойства Align, задающим метод выравнивания (размещения) компонента в контейнере. Контейнер это – компонент (форма или, например, компонент Panel) в поле которого размещен другой компонент. Контейнером верхнего уровня является форма. Некоторые из возможных значений свойства Align приведены в табл. 5.1.

Таблица 5.1. Значения свойства Align

Значение	Положение компонента в контейнере
Top	Компонент прижимается к верхней границе контейнера (например, формы), ширина компонента увеличивается до ширины контейнера, высота не меняется
Bottom	Компонент прижимается к нижней границе контейнера (например, формы), ширина компонента увеличивается до ширины контейнера, высота не меняется

Значение	Положение компонента в контейнере
Left	Компонент прижимается к левой границе контейнера (например, формы, компонента FlowLayout), высота компонента увеличивается до высоты контейнера, ширина не меняется
Right	Компонент прижимается к правой границе контейнера (например, формы), высота компонента увеличивается до высоты контейнера, ширина не меняется
Client	Компонент занимает всю свободную область контейнера
Center	Компонент располагается в центе свободной области контейнера по горизонтали и вертикали, размер компонента не меняется
VertCenter	Компонент располагается в центе контейнера по вертикали, ширина компонента устанавливается равной ширине свободной области контейнера
HorCenter	Компонент располагается в центе по горизонтали, ширина компонента не меняется, высота устанавливается высоте свободной области контейнера

Необходимо обратить внимание на то, что при использовании описываемой технологии создания формы важен порядок добавления компонентов на форму, а также порядок их настройки (присваивания нужного значения свойству Align).

Для позиционирования компонента внутри контейнера (смещения относительно других компонентов) следует использовать свойство Margins, которое задает отступы границ компонента от границ контейнера или других компонентов формы.

При создании формы в качестве контейнера удобно использовать компонент Panel, компоненты FlowLayout, GridLayout, GridPanelLayout.

Компонент FlowLayout позволяет связать несколько компонентов, выстроить их последовательно, один за другим. При этом изменение ширины предыдущего компонента приводит к сдвигу вправо следующих за ним компонентов. Благодаря этому свойству компонент удобно использовать для связывания, например, поясняющего текста (компонент Label) и поля ввода данных (компонент Edit), при изменении ширины компонента Label (изменении текста подсказки) компонент Edit автоматически будет менять свое положение, оставаясь на заданном расстоянии от правой границы компонента Label.

Компонент GridLayout представляет собой сетку, в ячейки которой можно поместить другие компоненты. Он полезен, если на форму надо поместить в виде таблицы (матрицы) несколько одинаковых компонентов, например, кнопок (клавиатура) или иллюстраций (страница просмотра эскизов).

Привязку компонента к границам формы можно выполнить также путем установки уточняющих значений свойства Anchors. При помощи этого свойства компонент можно привязать к границам контейнера. Например, для того чтобы кнопка находилась в левом нижнем углу формы вне зависимости от ориентации устройства, то свойствам Anchors. akLeft и Anchors.akBottom надо присвоить значение True, а свойствам Anchors.Top и Anchors.akRight – значение False.

Концепция универсальности предполагает, что для приложения создается одна форма, которая выглядит красиво или по крайней мере правильно отображается на всех

устройствах, независимо от платформы и размера экрана. Тем не менее на практике добиться этого не всегда удается. Поэтому, программист может для каждого устройства создать свою форму и путем изменения настроек компонентов, например, отступов (Margins) и масштаба отображения (Scale), скорректировать вид компонентов с учетом размера экрана.

Таким образом, методика создания формы универсального приложения может быть представлена последовательностью следующих шагов:

1. Создать Master форму приложения.
2. В окне дизайнера формы в списке View выбрать вид, соответствующий размеру экрана устройства, на котором должно работать приложение.
3. Если необходимо, то выполнить «тонкую» настройку формы путем изменения, например, масштаба отображения компонентов (свойство Scale), величин отступов (свойство Margins).
4. Повторить шаги 2 и 3 для каждого размера экрана, который должно поддерживать устройство.

Создание формы

Особенности создания формы Multi-Device приложения рассмотрим на примере. Ниже приведена последовательность шагов, которую надо выполнить, чтобы форма программы Скидка выглядела так, как показано на рис. 5.14.

1. Добавить на форму компонент Panel (находится на вкладке Standard), свойству Height присвоить значение 257, присвоить свойству Align значение Top, свойствам Margins.Left и Margins.Top присвоить значения 5.
2. Добавить в компонент Panel компонент Label (чтобы добавить компонент Label именно в панель, а не на форму, надо перед тем как перетащить Label из палитры на панель, сделать щелчок на имени Panel1 в окне Structure), присвоить свойству Align значение Top.
3. Добавить в панель компонент Layout (он находится на вкладке Layouts), свойству Align значение Top.
4. Добавить в компонент Layout1 компонент Rectangle (он находится на вкладке Shapes), свойству Align значение Client.
5. Добавить в компонент Layout1 компонент Label, свойству Name присвоить значение Label4, свойству Align присвоить значение Client, свойству Margins.Left присвоить значение 10.
6. Добавить в компонент Panel компонент Label, присвоить свойству Align значение Top.
7. Добавить в панель компонент Layout, свойству Align значение Top.
8. Добавить в компонент Layout компонент Rectangle, свойству Align значение Client.
9. Добавить в компонент Layout компонент Label, свойству Name присвоить значение

Label5, свойству Align присвоить значение Client, свойству Margins.Left присвоить значение 10.

10. Добавить в компонент Panel компонент Label, присвоить свойству Align значение Top.
11. Добавить в панель компонент Layout, свойству Align значение Top.
12. Добавить в компонент Layout компонент Rectangle, свойству Align значение Client.
13. Добавить в компонент Layout компонент Label, свойству Name присвоить значение Label6, свойству Align присвоить значение Client, свойству Margins.Left присвоить значение 10.
14. Добавить на форму компонент GridPanelLayout (он находится на вкладке Layouts), свойству Height присвоить значение 210, свойствам Margins.Left и Margins.Top присвоить значения 5.
15. Добавить в коллекцию ColumnCollection компонента GridPanelLayout один элемент (для этого в окне **Structure** из контекстного меню узла ColumnCollection компонента GridPanelLayout1 надо выбрать команду Add item). Установить ширину первого столбца 33%, второго – 33%, третьего – 34%. Чтобы это сделать, надо в окне Structured выбрать нужный столбец (элемент коллекции), присвоить свойству SizeStyle значение Percent, а свойству Value – требуемое значение.
16. Добавить в коллекцию RowCollection компонента GridPanelLayout два элемента (для этого в окне **Structure** из контекстного меню узла RowCollection компонента GridPanelLayout1, надо два раза выбрать команду Add item). Установить высоту строк 25%. Для этого, надо в окне Structured выбрать нужную строку (элемент коллекции RowCollection), присвоить свойству SizeStyle значение Percent, а свойству Value – значение 25.
17. В каждую ячейку компонента GridPanelLayout поместить командную кнопку (компонент Button), присвоить свойствам Align всех кнопок значение Client, а свойствам Margins.Top, Margins.Bottom, Margins.Left и Margins.Right значение 1.

Рисунок 5.14. Форма приложения после добавления компонентов

После того как компоненты будут помещены на форму, необходимо выполнить их настройку. Настройка компонентов заключается в присваивании нужных значений их свойствам (табл. 5.2).

Таблица 5.2. Значение свойств компонентов

Компонент	Свойство	Значение
Label1	Text	Цена без скидки
	StyleLookup	labelstyle
Label2	Text	Скидка, %
	StyleLookup	listboxheaderlabel
Label3	Text	Цена со скидкой
	StyleLookup	listboxheaderlabel
Label4	Text	
	StyleLookup	labelstyle
	HitTest	True
Label5	Text	
	StyleLookup	listboxheaderlabel
	HitTest	True

Компонент	Свойство	Значение
Label6	Text	
	StyleLookup	listboxheaderlabel
	HitTest	True

Настройка компонентов Button заключается в изменении свойств Text и Tag. Свойству Tag цифровых кнопок, надо присвоить значение, соответствующее цифре, отображаемой на кнопке (например, свойству Tag кнопки 3 — присвоить значение 3), свойствам Tag кнопок «Точка» и «Clear» — присвоить соответственно значения 10 и 11.

После настройки форма должна выглядеть так, как показано на рис. 5.15.

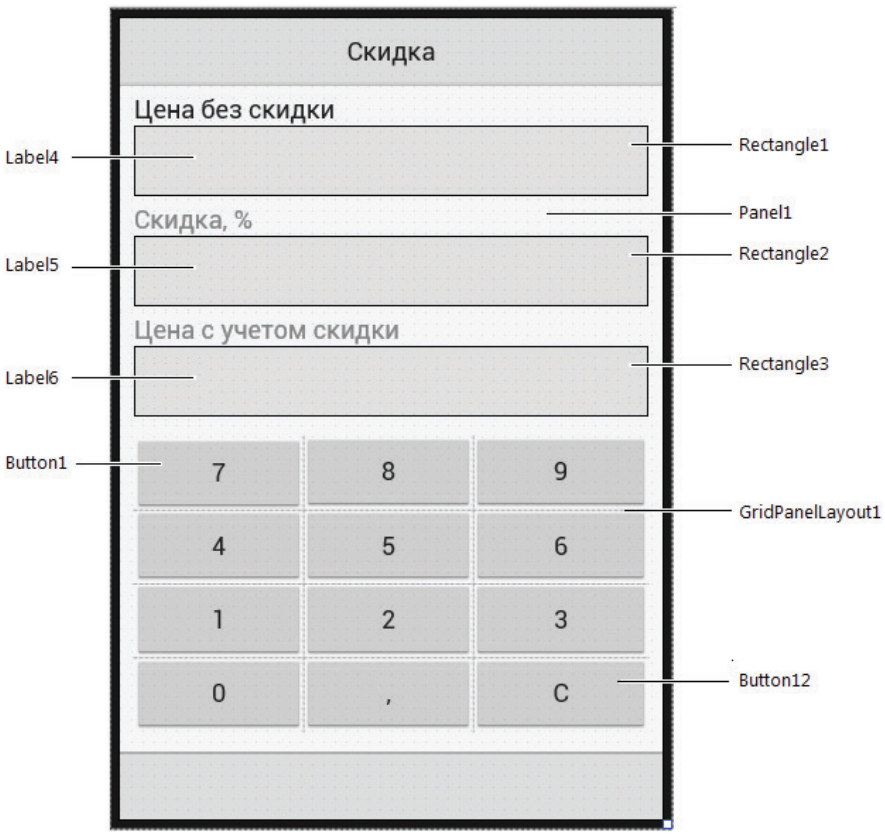


Рисунок 5.15. Окончательный вид формы приложения

Поля Label4, Label5 и Label6 предназначены для отображения исходных данных и результата. Одно из трех полей всегда активно, именно в нем отображается число, набираемое пользователем на клавиатуре, которая отображается в окне программы. Активное поле выделяется более ярким шрифтом, также более ярким шрифтом выделяется подсказка активного поля. Степень яркости шрифта определяется текущим значением свойства StyleLookup. Если текст отображается контрастным это значит, что

значение свойства равно labelstyle, в случае обычного воспроизведения текста, значение этого свойства равно listboxheaderlabel. Содержимое полей автоматически меняется при изменении содержимого активного поля. Смена активного поля происходит при касании того или другого неактивного поля. Компоненты Rectangle играют роль подложки для компонентов ввода информации.

События

Вид формы приложения подсказывает, как должна работать программа. Очевидно, что пользователь должен выбрать поле (в начале работы программы активно поле Цена без скидки), в которое он хочет ввести информацию, и при помощи клавиатуры набрать число. Затем он должен выбрать другое поле, например, Скидка и также набрать на клавиатуре число. Программа должна анализировать, какое из полей активно и выполнить расчет.

Для того чтобы программа работала так, как описано, она должна обрабатывать события, возникающие при касании полей (областей) ввода информации, и нажатии командных кнопок. При касании экрана возникает событие MouseDown. Именно эти события и обрабатываются. Следует обратить внимание, для того чтобы компонент Label воспринимал события, его свойству HitTest надо присвоить значение True (по умолчанию значение свойства HitTest компонентов Label равно False).

Процедуры обработки событий создаются обычным образом: сначала надо выбрать компонент, для которого необходимо создать процедуру обработки события, затем на вкладке **Events** окна **Object Inspector** выбрать событие и сделать двойной щелчок левой кнопкой мыши в поле редактирования, которое находится справа от имени события.

Модуль формы приложения Скидка приведен в листинге 5.1. Обратите внимание, процедура Button1MouseDown обрабатывает событие MouseDown всех кнопок. Информацию о том, какая кнопка нажата извлекается из свойства Tag параметра Sender. Для компонента Button1 процедура обработки события создается обычным образом. Процедуру обработки события MouseDown для остальных кнопок создавать не надо. Просто необходимо указать, что это событие должна обрабатывать процедура Button1MouseDown - раскрыть список процедур обработки событий и выбрать нужное имя (рис. 5.16).

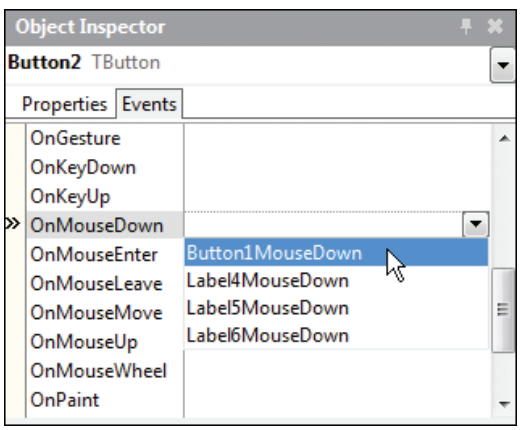


Рисунок 5.16. Назначение процедуры обработки события

Листинг 5.1. Модуль формы приложения Скидка

```
unit HeaderFooterTemplate;
interface
uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Graphics, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.StdCtrls,
  FMX.Objects, FMX.Layouts;
type
  THeaderFooterForm = class(TForm)
    Header: TToolBar;
    Footer: TToolBar;
    HeaderLabel: TLabel;
    Panel1: TPanel;
    Label1: TLabel;
    Layout1: TLayout;
    Label2: TLabel;
    Layout2: TLayout;
    Label3: TLabel;
    Layout3: TLayout;
    Rectangle1: TRectangle;
    Rectangle2: TRectangle;
    Rectangle3: TRectangle;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    GridPanelLayout1: TGridPanelLayout;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
    Button10: TButton;
    Button11: TButton;
    Button12: TButton;
    procedure Label4MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Single);
    procedure Label5MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Single);
    procedure Label6MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Single);
    procedure Button1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Single);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  HeaderFooterForm: THeaderFooterForm;
implementation
```

```
{$R *.fmx}
{
  ВНИМАНИЕ! Чтобы компонент воспринимал события,
  в том числе, Click и MouseDown,
  его свойству HitTest надо присвоить значение True
}
var
  price: double;     // цена
  percent: double ; // скидка в процентах
  total : double;    // цена с учетом скидки
  pole : integer; // 0 - активное поле Цена без скидки;
                  // 1 - активное поле Скидка;
                  // 2 - активное поле Цена со скидкой
// касание поля Цена без скидки
procedure THeaderFooterForm.Label4MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Single);
begin
      pole := 0;
      Label1.StyleLookup := 'labelstyle';
      Label4.StyleLookup := 'labelstyle';
      Label2.StyleLookup := 'listboxheaderlabel';
      Label5.StyleLookup := 'listboxheaderlabel';
      Label3.StyleLookup := 'listboxheaderlabel';
      Label6.StyleLookup := 'listboxheaderlabel';
      if price <> 0 then
          Label4.Text := FloatToStrF(price, ffFixed,9,2);
      if percent <> 0 then
          Label5.Text := FloatToStrF(percent, ffFixed,9,2);
      if total <> 0 then
          Label6.Text := FloatToStrF(total, ffCurrency,9,2);
end;
// касание поля Скидка
procedure THeaderFooterForm.Label5MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Single);
begin
      pole := 1;
      Label2.StyleLookup := 'labelstyle';
      Label5.StyleLookup := 'labelstyle';
      Label1.StyleLookup := 'listboxheaderlabel';
      Label4.StyleLookup := 'listboxheaderlabel';
      Label3.StyleLookup := 'listboxheaderlabel';
      Label6.StyleLookup := 'listboxheaderlabel';
      if price <> 0 then
          Label4.Text := FloatToStrF(price, ffCurrency,9,2);
      if percent <> 0 then
          Label5.Text := FloatToStrF(percent, ffFixed,9,2);
      if total <> 0 then
          Label6.Text := FloatToStrF(total, ffCurrency,9,2);
end;
// касание поля Цена со скидкой
procedure THeaderFooterForm.Label6MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Single);
begin
      pole := 2;
```

```
      Label3.StyleLookup := 'labelstyle';
      Label6.StyleLookup := 'labelstyle';
      Label1.StyleLookup := 'listboxheaderlabel';
      Label4.StyleLookup := 'listboxheaderlabel';
      Label2.StyleLookup := 'listboxheaderlabel';
      Label5.StyleLookup := 'listboxheaderlabel';
      if price <> 0 then
          Label4.Text := FloatToStrF(price, ffCurrency,9,2);
      if percent <> 0 then
          Label5.Text := FloatToStrF(percent, ffFixed,9,2);
      if total <> 0 then
          Label6.Text := FloatToStrF(total, ffFixed,9,2);
end;
// нажатие кнопки
// процедура обрабатывает нажатие всех кнопок
procedure THeaderFooterForm.Button1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Single);
var
  tag: integer; // идентификатор нажатой кнопки
  st : string;  // строка в активном поле
begin
     case pole of
        0: st := Label4.Text;
        1: st := Label5.Text;
        2: st := Label6.Text;
     end;
     tag := (Sender as TButton).Tag;
     case tag of
      0..9: // цифра
            begin
              st := st + Chr(Ord('0')+tag);
            end;
      10:   // десятичный разделитель
            begin
                if Pos(System.SysUtils.FormatSettings.DecimalSeparator ,st) = 0 then
                    st := st +  System.SysUtils.FormatSettings.DecimalSeparator;
            end;
       11:  // кнопка clear
            st := '';
     end;
     case pole of
        0:  // изменена цена
            // если задана величина скидки,
            // то пересчитать цену с учетом скидки
            begin
              Label4.Text := st;
              if st.Length <> 0 then
                  price := StrToFloat(st)
              else
                  price := 0;
              total := price*(1-percent/100);
              if total <> 0 then
                    label6.Text := FloatToStrF(total,ffCurrency,9,2)
              else
                    label6.Text := '';
            end;
```

```
        1:  // изменена величина скидки
            // если задана цена, то пересчитать
            // цену с учетом скидки
            begin
              Label5.Text := st;
              if st.Length <> 0 then
                    percent := StrToFloat(st)
              else
                    percent := 0;
              total := price*(1-percent/100);
              if total <> 0 then
                      label6.Text := FloatToStrF(total,ffCurrency,9,2)
              else
                      label6.Text := '';
            end;
        2:  // изменена цена с учетом скидки
            // если задан процент скидки,
            // то вычислить исходную цену
            begin
                Label6.Text := st;
                if st.Length <> 0 then
                    total := StrToFloat(st)
                else
                    total := 0;
                // расчет цены с учетом скидки
                price := total/(1-percent/100);
                if price <> 0 then
                      label4.Text := FloatToStrF(price,ffCurrency,9,2)
                else
                      label4.Text := '';
            end;
    end;
end;
end.
```

Структура проекта

Проект — это совокупность файлов, которые использует компилятор для создания выполняемой программы (exe-файл Windows приложения) или пакет (Android, iOS приложения) для дальнейшего развертывания его на реальное или виртуальное устройство.

Файлы, образующие проект, находятся в папке проекта.

Структура проекта, в том числе информация о текущей конфигурации (Debug или Release), целевой платформе и устройстве, на которое предполагается развернуть приложение, отображается в окне **Project Manager** (рис. 5.17).

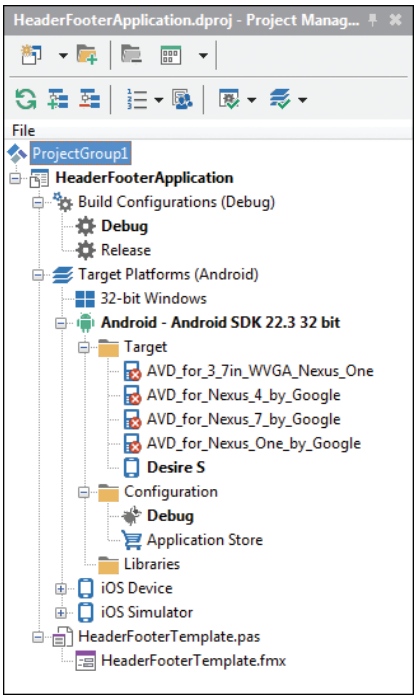


Рисунок 5.17. Текущая структура проекта разработки приложения Скидка для платформы Android

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 среда разработки создает pas и fmx файлы. Pas файл содержит код, в том числе процедур обработки событий, fmx - описание формы, подробную информацию о компонентах, их положении, размере, виде. Общая информация о проекте находится в dproj файле.

Компиляция

Перед тем как выполнить компиляцию, а тем более запуск приложения, рекомендуется сохранить проект. Чтобы сохранить проект, нужно в меню **File** выбрать команду **Save All** или сделать щелчок на соответствующей командной кнопке, находящейся в панели инструментов.

Процесс компиляции активизируется в результате выбора в меню **Project** команды **Compile**, а также в результате запуска программы из среды разработки (команда **Run** или **Run Without Debugging** меню **Run**), если с момента последней компиляции в программу были внесены изменения.

Процесс и результат компиляции отражаются в окне **Compile**. Обратите внимание, во время компиляции в окне **Compile** отображается имя файла, который создает компилятор (строго говоря, компоновщик). Так, например, приведенное на рис. 5.18 окно, отражающее процесс компиляции приложения для платформы Android, показывает, что результатом компиляции является пакет – архив специального формата.

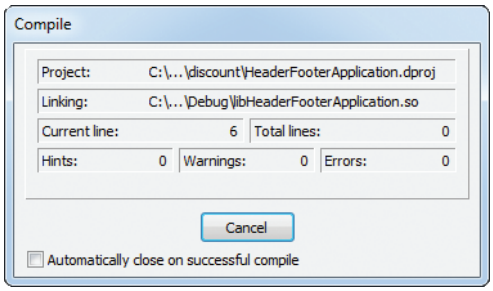


Рисунок 5.18. Результатом компиляции приложения для платформы Android является пакет

Запуск приложения

Приложение можно запустить на реальном или виртуальном устройстве (эмуляторе). В любом случае, приложение надо установить на устройство. Процесс установки приложения на устройство называется развертыванием (Deploy).

Опыт показывает, что использовать виртуальное устройство для отладки Android приложения вследствие крайне низкой скорости работы и длительного времени загрузки приложения, неудобно и практически невозможно. Поэтому будем ориентироваться на реальное устройство.

Подготовка устройства

Загрузка программы (пакета) на устройство выполняется по USB кабелю. Чтобы из среды разработки можно было развернуть приложение на устройство, на компьютере должен быть установлен USB драйвер этого устройства (рис. 5.19). Драйвер устройства, как правило, можно загрузить с сайта производителя.

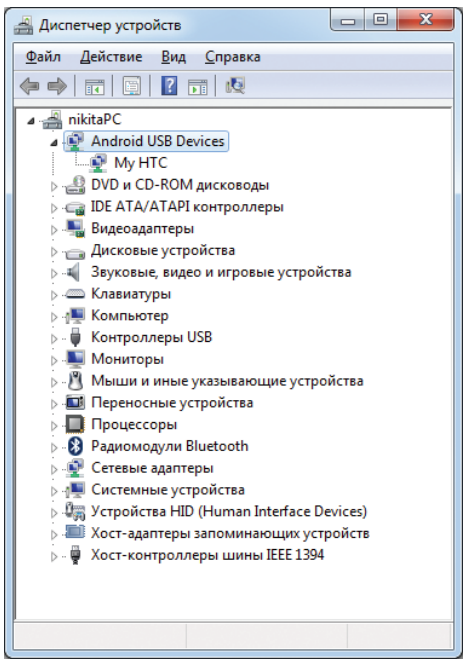


Рисунок 5.19. На компьютер разработчика необходимо установить драйвер Android устройства

Кроме установки драйвера на компьютер разработчика, необходимо выполнить настройку операционной системы устройства, чтобы она позволяла выполнять загрузку и отладку приложений по USB. Для этого на Android устройстве надо открыть вкладку **Настройки**, найти на ней раздел **Для разработчиков**, разрешить использовать устройство в режиме разработчика (установить во включенное состояние переключатель **Режим разработчика** и подтвердить разрешение разработки, рис. 5.20) и отладку по USB (установить во включенное состояние переключатель **Отладка по USB** и подтвердить разрешение отладки по USB, рис. 5.21).

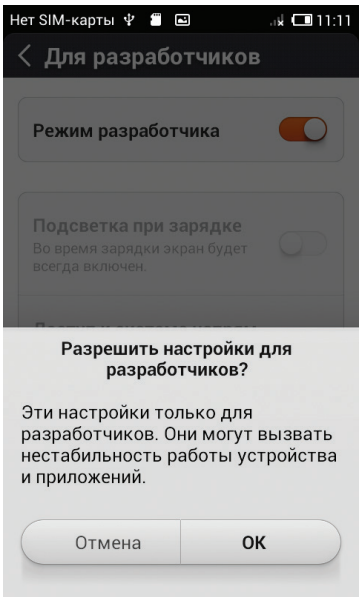


Рисунок 5.20. Разрешение использовать устройство в режиме разработки

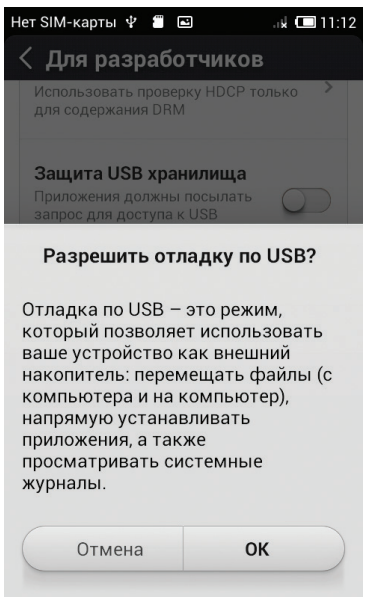


Рисунок 5.21. Разрешение отладки по USB

Развертывание и запуск

Процесс развертывания приложения на устройство активизируется автоматически в результате запуска программы из среды разработки командой **Run** или **Run Without Debugging**. Можно также сделать щелчок на кнопке **Run Without Debugging, Run,** (рис. 5.22) или нажать клавишу F9.

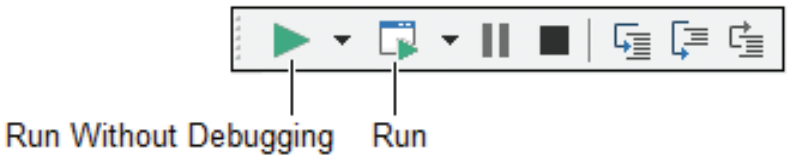


Рисунок 5.22. Чтобы запустить программу, сделайте щелчок на кнопке **Run**

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. Следует обратить внимание, что процесс запуска программы командой **Run Without Debugging** происходит быстрее.

После того как приложение будет запущено из среды разработки, его можно будет запустить на устройстве обычным образом, щелчком на плитке, изображающей программу на рабочем столе (рис. 5.23). Обратите внимание, по умолчанию под значком указывается имя проекта.

Рисунок 5.23. Приложение Скидка, установленное на устройство

Настройка приложения

После того как приложение будет отлажено, можно выполнить его окончательную настройку: задать название программы, значок (LauncherIcon), который будет изображать приложение на рабочем столе, заставку (SplashImage), отображаемую на экране во время загрузки (запуска) приложения.

Имя приложения надо ввести в поле Label (рис. 5.24) на вкладке **Version Info** окна **Project Options**, которое становится доступным в результате выбора в меню **Project** команды **Options**.

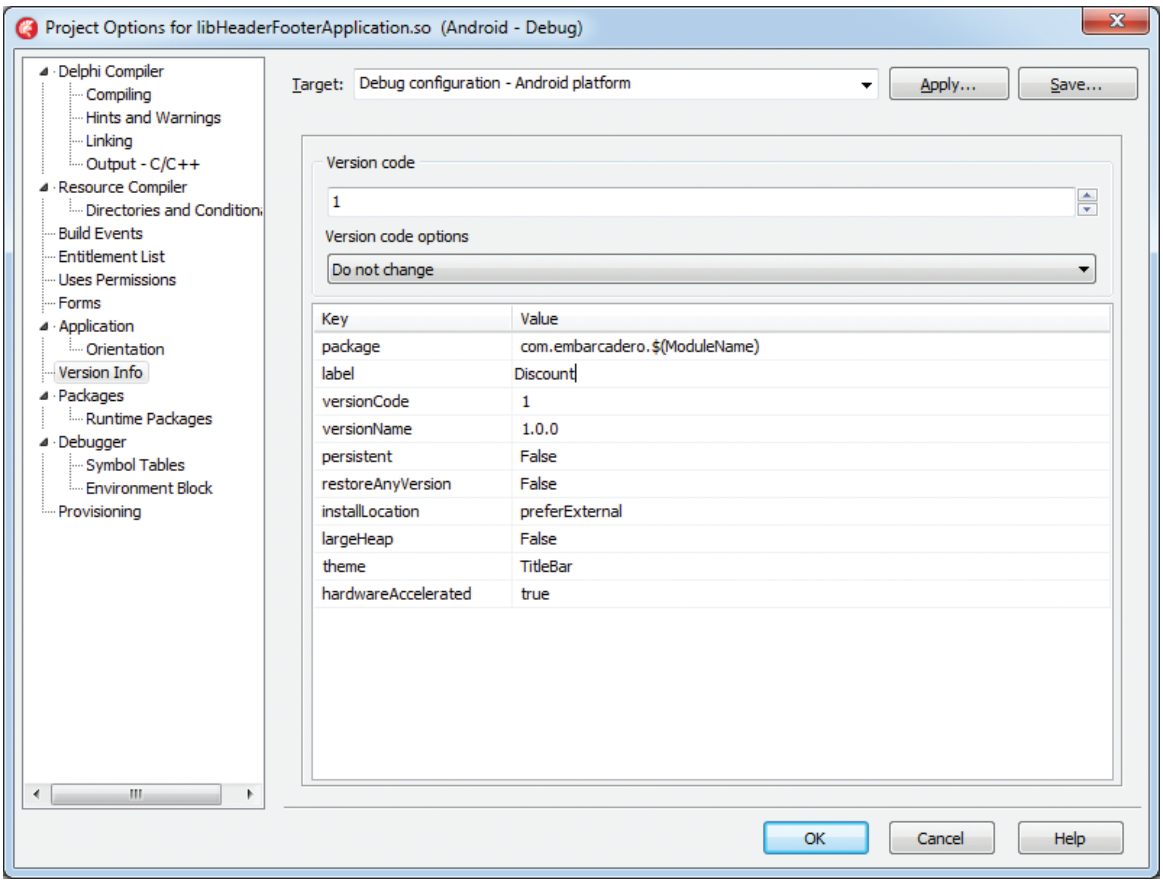


Рисунок 5.24. Имя программы надо ввести в поле Label

По умолчанию FireMonkey Android приложение изображается значком, который показан на рис. 5.23. Чтобы изменить этот стандартный значок на уникальный, надо на вкладке **Application** окна **Project Options** указать имя png файла, содержащего нужное изображение (рис. 5.25).

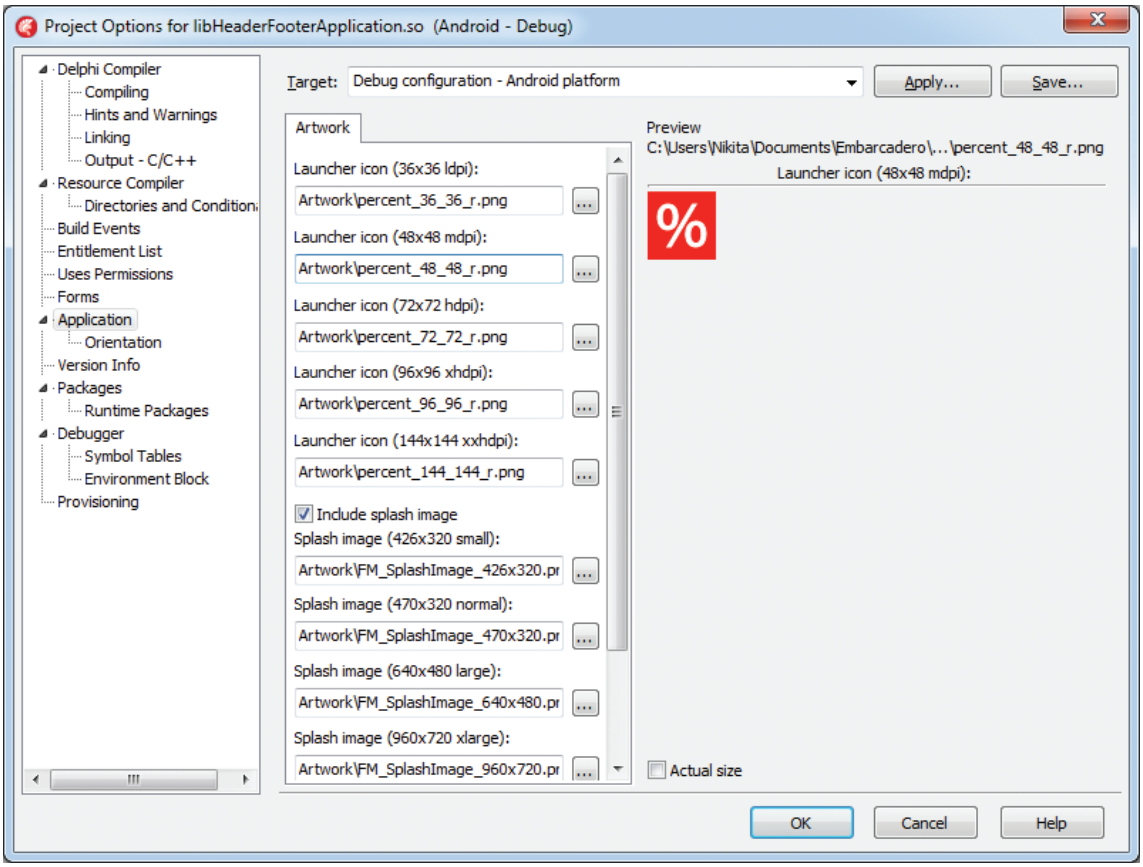


Рисунок 5.25. Настройка приложения – значок приложения на рабочем столе

Следует обратить внимание, что для приложения надо создать пять значков (png файлов), размера: 36x36, 48x48, 72x72, 96x96 и 144x144 пикселей (рис. 5.26). Файлы, содержащие значки, рекомендуется разместить в отдельном каталоге в папке проекта. Чтобы изменить значок приложения, надо сделать щелчок на кнопке с тремя точками и указать png файл, содержащий значок нужного размера.

Рисунок 5.26. Для приложения надо создать пять значков

Картинка, отображаемая во время запуска программы (Splash Image), задается аналогично.

Следует обратить внимание, изменение значка на рабочем столе происходит только после перезагрузки устройства. Результат настройки приложения приведен на рис. 5.27.

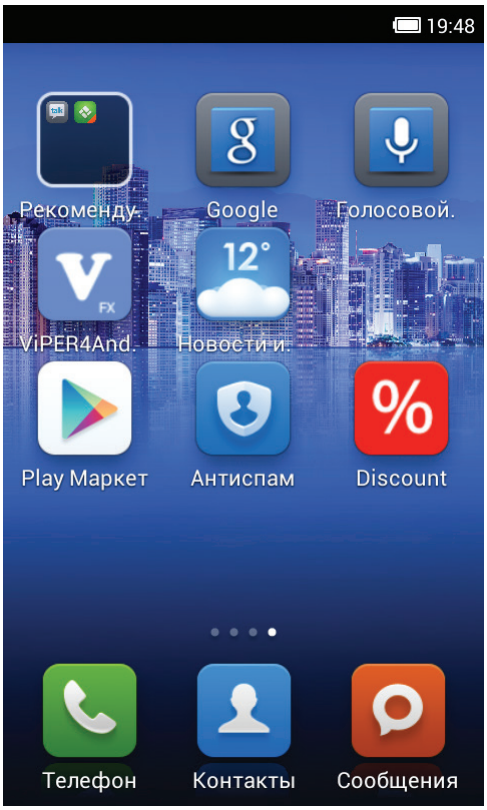


Рисунок 5.27. Значок приложения Скидка на рабочем столе

Стилевое оформление

Стиль – это совокупность характеристик компонентов, определяющих вид отдельных компонентов и окна приложения в целом.

Среда разработки предоставляет в распоряжение программиста компонент StyleBook (рис. 5.28) и набор стилей, позволяющий легко изменить стиль оформления Android или Win32 приложения.

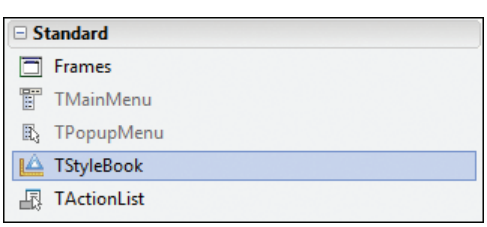


Рисунок 5.28. Компонент StyleBook

Чтобы изменить стиль приложения, надо добавить в форму компонент StyleBook, сделать щелчок на кнопке с тремя точками, находящейся в поле значения свойства Resource, в открывшемся окне нажать кнопку **Load** и в стандартном окне **Открыть**, выбрать один из файлов стилевого оформления. Предоставляемые средой разработки, файлы стилевого

оформления, находятся в папке C:\Пользователи\Общие\Общие документы\Embarcadero\Studio\ 15.0\Styles по умолчанию и имеют расширение style. Выбрав стилевой файл, например, GoldenGraphite следует нажать кнопку **Apply** and **Close**. После того как стиль будет выбран, необходимо в качестве значения свойства StyleBook формы приложения указать имя компонента StyleBook. В результате выполнения описанных действий, вид формы приложения примет облик в соответствии с выбранным стилем. На рис. 5.29 приведена форма программы Скидка, в которой использован стиль GoldenGraphite.

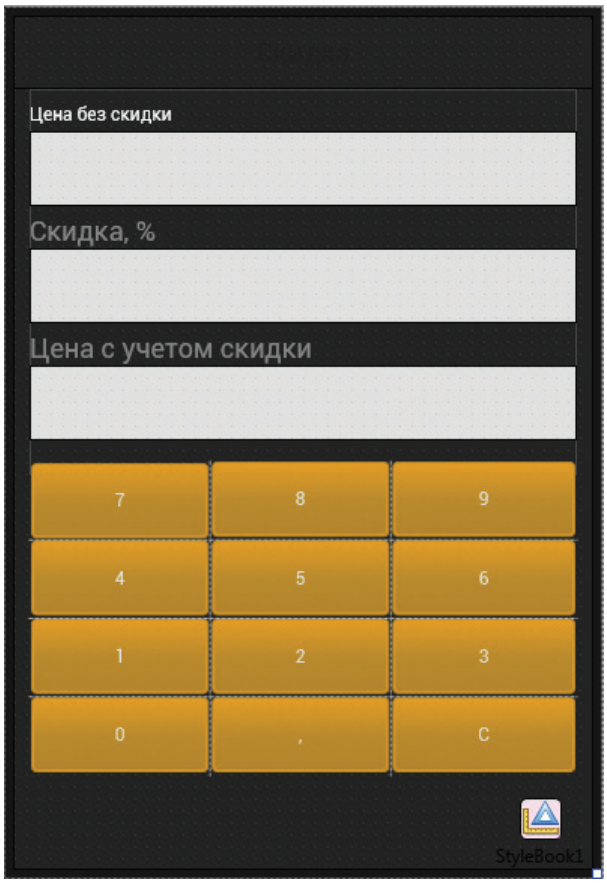


Рисунок 5.29. Пример применения стиля

Глава 6. Графика FireMonkey

В этой главе рассказывается, как отобразить графику в окне Multi Device приложения, что надо сделать, чтобы в окне программы появился график, диаграмма или картинка, в том числе фотография. Также вы узнаете, как получить доступ к галерее фотографий и камере мобильного устройства.

Формирование графики

Графику в окне программы можно сформировать из графических объектов или нарисовать на графической поверхности.

Графические объекты (фигуры) находятся на вкладке Shapes палитры компонентов. Чтобы получить картинку из фигур, надо обычным образом поместить на форму нужные объекты и выполнить их настройку – присвоить свойствам объектов требуемые значения.

Во время работы программы в качестве поверхности для формирования графики удобно использовать компонент Image. Рисование на его графической поверхности, вычерчивание графических примитивов, выполняют соответствующие методы свойства Canvas.

Графическая поверхность

Доступ к графической поверхности компонента Image, которая представляет собой объект TCanvas, осуществляется через свойство Bitmap.

Графическая поверхность представляет собой совокупность пикселей. Положение пикселя на графической поверхности определяется горизонтальной (X) и вертикальной (Y) координатами. Координата X возрастает слева направо, Y – сверху вниз. Левый верхний пиксель имеет координаты (0, 0).

Чтобы на поверхности объекта Image появилась линия, прямоугольник, эллипс или другой графический элемент, необходимо вызвать соответствующий метод. Например, инструкция

```
Image1.Bitmap.Canvas.DrawLine(p1,p2,1.0);
```

рисует линию. Параметры p1 и p2 (объекты TPoint) задают точки начала и конца линии, константа 1.0 – степень прозрачности линии.

Инструкция

```
Image1.Bitmap.Canvas.DrawRect(rect,5,5,[TCorner.TopLeft, TCorner.BottomLeft],0.8);
```

рисует на поверхности компонента Image1 прямоугольник с двумя скругленными углами. Положение и размер прямоугольника задает параметр rect (объект TRectF). Углы, которые должны быть скруглены (в приведенном примере левый верхний и правый нижний углы), задает третий параметр. Второй и третий параметры задают радиусы скругления углов по осям X и Y. Последний параметр (константа 0.8) - степень прозрачности линии границы прямоугольника.

В приведенных инструкциях для указания точек начала и конца линии, а также положения и размера прямоугольника использованы объекты TPoint и TRectF. Объект TPoint хранит координаты точки графической поверхности, объект TRectF – характеристики области.

Чтобы задать координаты точки, надо присвоить значения свойствам X и Y объекта

Чтобы задать положение и размер области, надо указать координаты левого верхнего и правого нижнего углов области. Свойства Left, Top и Right, Bottom задают, соответственно, координаты X и Y углов прямоугольника.

Графические примитивы

Графические примитивы — это элементарные геометрические фигуры. Линия, прямоугольник, эллипс, сектор – это графические примитивы.

В табл. 6.1 приведено описание методов объекта TCanvas, рисующих графические примитивы. В описании методов параметры p1 и p2, представляют собой объекты TPoint, параметр rect – объект TRectF.

Таблица 6.1. Методы класса TCanvas

Метод	Действие
DrawLine(p1,p2,op)	Рисует линию из точки p1 в точку p2. Параметр op (opacity - непрозрачность) задает степень непрозрачности линии (1.0 – линия непрозрачная, 0.5 – непрозрачность 50%, 0.0 – линия не отображается). Цвет линии определяет свойство Stroke.Color поверхности, на которой рисует метод
DrawRect(rect,rx,ry,corners,op)	Рисует границу прямоугольника. Параметр rect задает положение и размер прямоугольника. Параметры rx и ry задают радиусы скругления углов по осям x и y. Параметр corners (множество объектов TCorner) задает углы, которые должны быть скруглены. Параметр op задает степень непрозрачности линии. Цвет границы прямоугольника определяет свойство Stroke.Color поверхности, на которой рисует метод
FillRect(rect,rx,ry,corners,op)	Рисует прямоугольник. Параметр rect задает положение и размер прямоугольника. Параметры rx и ry задают радиусы скругления углов по осям x и y. Параметр corners (множество объектов TCorner) задает углы, которые должны быть скруглены. Цвет закраски прямоугольника определяет свойство Fill.Color поверхности, на которой рисует метод
DrawEllipse(rect,op)	В зависимости от соотношения длин сторон области, задаваемой параметром rect, рисует эллипс или окружность. Параметр op задает степень непрозрачности линии эллипса (окружности). Цвет линии определяет свойство Stroke.Color поверхности, на которой рисует метод
FillEllipse(rect,op)	В зависимости от соотношения длин сторон области, задаваемой параметром rect, рисует закрашенный эллипс или круг. Параметр t задает степень прозрачности закраски эллипса. Цвет закраски эллипса определяет свойство Fill.Color
DrawArc(p1, p2, st, len, op)	Рисует дугу. Параметр p1 задает центр дуги, параметр p2 - радиусы эллипса (по осям X и Y), из которого «вырезается» дуга. Параметр st задает угол начала дуги, параметр len – длину дуги в градусах. Параметр t задает степень прозрачности линии дуги

Метод	Действие
FillArc(p1, p2, st, len, op)	Рисует сектор. Параметр p1 задает центр сектора, параметр p2 - радиусы эллипса (по осям X и Y), из которого «вырезается» сектор. Параметр st задает угол начала сектора, параметр len – длину сектора в градусах. Параметр t задает степень прозрачности закраски сектора
FillText(rect, st, wrap, op, flags, HorAlign,VertAlign)	Выводит в область rect текст st. Параметры HorAlign и VertAlign задают метод выравнивания текста внутри области по горизонтали и вертикали. Параметр op задает степень непрозрачности текста. Цвет текста задает свойство Canvas.Fill.Color, параметры шрифта – свойство Canvas.Font.

Цвет лини, например, границы рисуемого методом DrawRect прямоугольника, и цвет заливки области, например, рисуемой методом FillRect, определяют, соответственно, значения свойств Fill.Color и Stroke.Color графической поверхности, на которой рисует метод. В качестве их значения следует использовать константу TAlphaColors, некоторые из которых приведены в табл. 6.2.

Таблица 6.2. Константы TAlphaColor

Цвет	Константа
TAlphaColor.Aqua	Бирюзовый
TAlphaColor.Black	Черный
TAlphaColor.Blue	Синий
TAlphaColor.Fuchsia	Ярко-розовый
TAlphaColor.Green	Зеленый
TAlphaColor.Lime	Салатный
TAlphaColor.Maroon	Каштановый
TAlphaColor.Navy	Темно-синий
TAlphaColor.Olive	Оливковый
TAlphaColor.Purple	Розовый
TAlphaColor.Red	Красный
TAlphaColor.Silver	Серебристый
TAlphaColor.Teal	Зелено-голубой
TAlphaColor.White	Белый

Толщину и вид линии (границы геометрической фигуры) определяют свойства Stroke. Thickness и Stroke.Dash, а стиль закраски внутренних областей геометрических фигур, рисуемых соответствующими методами, определяет свойство Fill.Kind (табл. 6.3).

Таблица 6.3. Свойства графической поверхности, определяющие вид объектов

Свойство	Описание
Stroke.Color	Цвет линии (границы геометрической фигуры)
Stroke.Thickness	Толщина линии (границы геометрической фигуры)

Свойство	Описание
Stroke.Dash	Стиль линии (границы геометрической фигуры). Вид линии: TStrokeDash.Solid — сплошная; TStrokeDash.Dash — пунктирная, длинные штрихи; TStrokeDash.Dot — пунктирная, короткие штрихи; TStrokeDash.DashDot — пунктирная, чередование длинного и короткого штрихов; TStrokeDash.DashDotDot — пунктирная, чередование одного длинного и двух коротких штрихов
Fill.Color	Цвет закраски внутренней области геометрической фигуры, цвет текста, выводимого на графическую поверхность методом FillText
Fill.Kind	Стиль закраски внутренней области геометрической фигуры. Закраска может быть сплошной (TBrushKind.Solid), градиентной (TBrushKind.Gradient) или прозрачной (TBrushKind.None). Также для закраски (заполнения) внутренней области может использоваться битовый образ (TBrushKind.Bitmap) или ресурс (TBrushKind.Resource)
Fill.Font	Шрифт, используемый для вывода текста на графическую поверхность.

Событие Paint

Графику на графической поверхности должна формировать процедура обработки события Paint. Это событие возникает всякий раз, когда необходимо нарисовать (перерисовать объект), в том числе в начале работы программы, когда объект первый раз появляется на экране после запуска программы, а также при смене ориентации устройства, например, с «портрета» на «ландшафт».

Цикл формирования графики, состоящий из последовательности инструкций, формирующих изображение на графической поверхности, называется сценой и должен находится между инструкциями вызова методов BeginScene и EndScene того графического объекта, на который выводится графика. Например, процедура обработки события Paint компонента Image, которая рисует рамку по границе компонента, должна выглядеть так:

```
procedure TForm1.Image1Paint(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
      r : TRectF;
begin
      r.Left := 5;
      r.Top := 5;
      r.Right := Image1.Width - 5;
      r.Bottom := Image1.Height - 5;

      Image1.Bitmap.Canvas.BeginScene;

      Image1.Bitmap.Canvas.DrawRect(r,0,0,[],1.0);

      Image1.Bitmap.Canvas.EndScene;
end;
```

Формирование графики

Следующая программа, ее текст приведен в листинге 6.1, демонстрирует процесс формирования сложного графического изображения, она, используя методы объекта Canvas, строит график изменения курса доллара. Программа спроектирована так, что график занимает всю доступную область экрана вне зависимости от ориентации устройства (рис. 6.1, рис. 6.2). При изменении ориентации экрана график автоматически перестраивается.

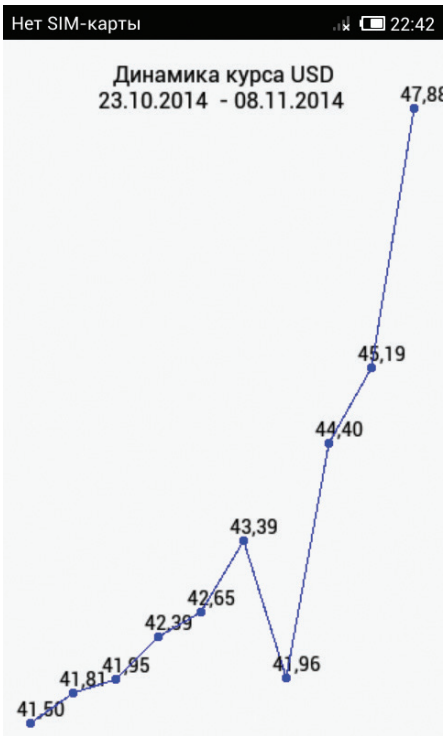


Рисунок 6.1. График при “портретной” ориентации устройства

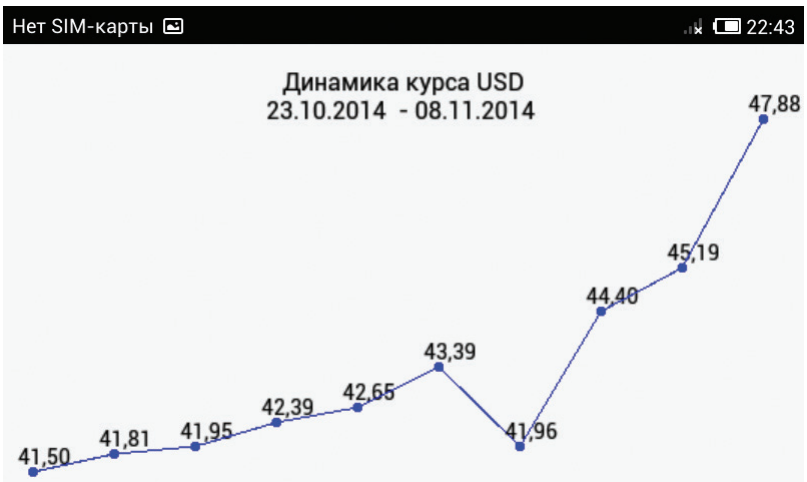


Рисунок 6.2. График при “ландшафтной” ориентации устройства

Форма программы содержит один единственный компонент Image и, поэтому, не приведена (настройку компонента Image выполняет конструктор – процедура обработки события Create формы).

Данные загружаются из файла usd.txt (листинг 6.2). Файл данных для Android приложения должен находится в папке **assets\internal**, а для приложения Win32 - в папке **Мои документы**. Загрузку данных из файла и их обработку (поиск минимального и максимального значений) выполняет функция Load, которую запускает процедура обработки события Activate формы. График строит процедура обработки события Paint компонента Image. Следует обратить внимание на то, что более «свежая» информация находится в начале файла. Поэтому график строится не от первой точки (элемента массива), а от последней. Кроме того, график строится в отклонениях от минимального значения.

Листинг 6.1. График

```
/  (c) Никита Культин, 2015.
//
// Программа демонстрирует использование методов
// отображения графических примитивов объекта Canvas
// для формирования графики в поле компонента Image.

unit Unit1;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs, FMX.Objects,
  FMX.StdCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Image1Paint(Sender: TObject; Canvas: TCanvas;
      const ARect: TRectF);
  private
    { Private declarations }
    Function Load(fname: string): integer;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.fmx}
uses System.IOUtils;
```

```
const
  HB = 15;
var
  kurs: array[1..HB] of real;       // значение
  date: array[1..HB] of string;     // дата
  nRec: integer;                    // кол-во прочитанных записей
  min, max : integer;               // номер мин. и макс. значений в ряде данных
  title: string;                    // заголовок диаграммы
procedure TForm1.FormActivate(Sender: TObject);
begin
  nRec := Load('usd.txt');
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  image1.Align :=TAlignLayout.Client;
  Image1.Bitmap := TBitmap.Create(round(Image1.Width), round(Image1.Height));
  Image1.Bitmap.Clear(TAlphaColors.Whitesmoke);
end;
procedure TForm1.Image1Paint(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
  x,dx: integer;
  m: real;        // коэффициент масштабирования
  p1,p2 : TPoint; // координаты концов сегмента
  rect: TRectF;   // область отображения значения
  rect2 : TRectF; // область отображения маркера значение
  i: integer;
begin
      // событие Paint возникает, в том числе, при изменении ориентации устройства
      image1.Align :=TAlignLayout.Client;
      Image1.Bitmap := TBitmap.Create(round(Image1.Width), round(Image1.Height));
      Image1.Bitmap.Clear(TAlphaColors.Whitesmoke);
      {  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. }
        dx:= Trunc((Image1.Width - 40) / (nrec-1));
        // Вычислим масштаб.
        // Для построения графика будем использовать
        // не всю область компонента, а ее нижнюю часть.
        // Верхняя область высотой 60 пикселей используется
        // для отображения заголовка
        if (max <> min) then
          m := (Image1.Height - 60) / (kurs[max] - kurs[min])
        else
          m:=1;
        x := 20;
        rect := TRectF.Create(0,0,20,130); // область 20x130 для подписи значения
        Image1.Bitmap.Canvas.BeginScene;
        Image1.Bitmap.Canvas.Stroke.Color := TAlphaColors.Blue; // цвет линии
        for i := nrec downto 1 do
          begin
            p2.X := x;
            P2.Y := Round(Image1.Height -  (kurs[i] - kurs[min]) * m -10);
            // поставить точку
            Image1.Bitmap.Canvas.Fill.Color := TAlphaColors.Blue; // цвет точки
```

```
              rect2.Top := p2.Y - 3;
              rect2.Left := p2.X - 3;
              rect2.Right := p2.X +3;
              rect2.Bottom := p2.Y + 3;
              Image1.Bitmap.Canvas.FillEllipse(rect2,1.0);
              if (i >= 1) and (i < nrec)then
              begin
                 // линия из p1 в  p2
                 Image1.Bitmap.Canvas.DrawLine(p1,p2,1.0);
              end;
             // подписи данных
             if ( ( i = 1) or (kurs[i] <> kurs[i-1])) then
             begin
               rect.Left := p2.X-10;
               rect.Right := p2.X + 30;
               rect.Bottom := p2.Y;
               rect.Top := p2.Y-20;
                 //Image1.Bitmap.Canvas.FillText(rect,IntToStr(i),False,1.0,[],TTextAlign.
Leading);
               Image1.Bitmap.Canvas.Fill.Color := TAlphaColors.Black;
                     Image1.Bitmap.Canvas.FillText(rect,FloatToStrF(kurs[i],ffFixed,5,2),
False,1.0,[],TTextAlign.Leading);
             end;
             // в след. цикле текущая точка конца линии должна быть
             // точкой начала линии
             p1.X := p2.X;
             p1.Y  := p2.Y;
             x := x + dx;
    end;
      // заголовок диаграммы
      Image1.Bitmap.Canvas.Fill.Color := TAlphaColors.Black;
      Image1.Bitmap.Canvas.Font.Size := 16;
      rect.Top :=0;
      rect.Left := 0;
      rect.Right := Image1.Width;
      rect.Bottom := 70;
      //Image1.Bitmap.Canvas.DrawRect(rect,0,0, [],1.0);
       Image1.Bitmap.Canvas.FillText(rect, title, True, 255, [], TTextAlign.Center,TTex-
tAlign.Center);
      Image1.Bitmap.Canvas.EndScene;
end;
// загрузить данные из файла
function TForm1.Load(fname: string): integer;
var
  f: TextFile;
  st, st1: string;
  aFile: string;    // полное имя файла
  n: integer;       // кол-во элементов данных
  i: integer;
begin
     n := 0;
    //  System.SysUtils.GetHomePath:
    //       в Win32 приложении  -  C:\Users\UserName\AppData\Roaming
    //  TPath.GetDocumentsPath:
    //       в приложении Win32  -  C:\Users\UserName\Documents
    //       в Android приложении assets\internal\
   aFile := System.SysUtils.GetHomePath + System.SysUtils.PathDelim + fname;
```

```
    aFile := Tpath.Combine(TPath.GetDocumentsPath, fname);
    Label1.Text := aFile;
    AssignFile(f, aFile);
    if ( NOT System.SysUtils.FileExists(aFile)) then
      begin
        Load := -1;
        exit;
      end
    else begin
       Reset(f);
       // читать до конца, но не более HB записей
       while (not EOF(f) ) and (n < HB) do
       begin
              n := n +1;
              readln(f, st);       // дата
              date[n] := st;
              readln(f, st1);      // значение
              kurs[n] := StrToFloat(st1);
       end;
       CloseFile(f);
       Load := n;
      // 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;
       title := 'Динамика курса USD' + #13 + date[n] + ' - ' + Date[1];
    end;
end;
end.
```

Листинг 6.2. Файл данных программы График (usd.txt)

```
08.11.2014
47,8774
07.11.2014
45,1854
06.11.2014
44,3993
01.11.2014
41,9627
31.10.2014
43,3943
30.10.2014
42,6525
29.10.2014
42,3934
28.10.2014
41,9497
24.10.2014
41,8101
23.10.2014
41,4958
```

Отображение иллюстраций

Здесь и далее под иллюстрацией будем понимать любое изображение, находящееся в файле, в том числе и фотографию.

Для отображения иллюстраций предназначен компонент Image. Если иллюстрация статичная, например, логотип разработчика приложения или фоновый рисунок формы, то для того чтобы иллюстрация появилась в поле компонента Image, надо сделать щелчок на кнопке с тремя точками, которая находится в поле значения свойства MultiResBitmap, в появившемся окне Editing (рис. 6.3) сделать щелчок на кнопке открыть и обычным образом выбрать файл иллюстрации. В результате иллюстрация появится в поле компонента Image (рис. 6.4).

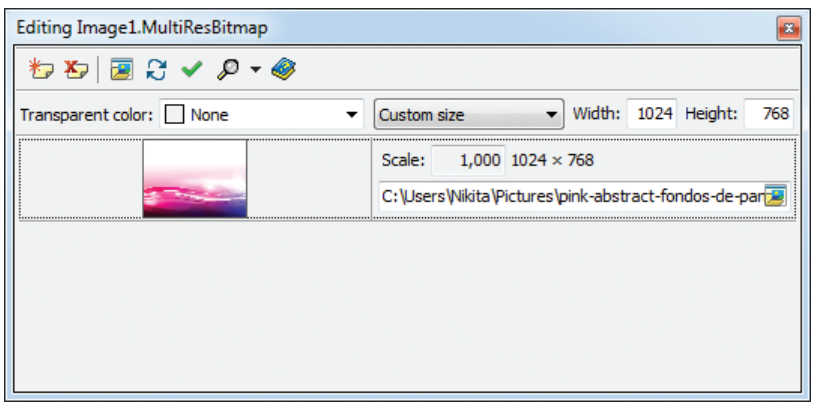


Рисунок 6.3. Выбор иллюстрации

Рисунок 6.4. Пример использования иллюстрации в качестве фона

Вид иллюстрации в поле компонента Image определяется размерами иллюстрации, размерами компонента и значением свойства WrapMode компонента Image, которое определяет метод масштабирования иллюстрации (табл. 6.4).

Таблица 6.4.

Значение свойства WrapMode	Метод масштабирования
Stretch	Изображение сжимается (растягивается) по вертикали и горизонтали так, чтобы иллюстрация целиком отображалось в поле компонента. При этом, как правило, пропорции искажаются, картинка выглядит сжатой по горизонтали или растянутой по вертикали.
Center	Если размер иллюстрации больше размера компонента, то отображается центральная часть иллюстрации без масштабирования.
Fit	Иллюстрация масштабируется так, чтобы она полностью отображалась без искажения.
Original	Если размер иллюстрации больше размера компонента, то отображается левая верхняя область иллюстрации, равная размеру компонента.
Tile	Если размер иллюстрации больше размера компонента, то отображается левая верхняя область иллюстрации, равная размеру компонента. Если размер картинки меньше, то область компонента заполняется плитками (tile - плитка) с изображениями картинки.

Если иллюстрация используется в качестве фона, то присвоив свойству Opacity значение меньше единицы, можно уменьшить контрастность иллюстрации.

Загрузить иллюстрацию в поле компонента Image можно во время работы программы. Сначала рассмотрим, как это сделать, если иллюстрация является частью приложения, затем – как загрузить иллюстрацию из галереи устройства или получить непосредственно с камеры.

Иллюстрации, которые использует приложение, обычно находятся в его рабочей папке. Приведенный в листинге 6.3 конструктор показывает, как отобразить в поле компонента Image иллюстрацию, находящуюся в рабочей папке приложения. Обратите внимание, файлы иллюстраций необходимо включить в пакет приложения.

Листинг 6.3. Загрузка иллюстрации из файла в компонент Image

```
procedure TForm1.FormCreate(Sender: TObject);
var
    bitmap: TBitmap;
    bmpFile: string;
    h: integer;
begin
    h := HourOf(System.SysUtils.GetTime);
    if (h < 7) and (h> 21) then
        bmpFile := TPath.Combine(TPath.GetDocumentsPath, 'back1.jpg')
    else
        bmpFile := TPath.Combine(TPath.GetDocumentsPath, 'back2.jpg');
    if ( System.SysUtils.FileExists( bmpFile) = true) then
    begin
   bitmap := TBitmap.CreateFromFile(bmpFile);
        Image1.Bitmap.Assign(bitmap);
    end;
end;
```

Вывод иллюстрации в Image

Если в область компонента Image, уже содержащего какую-либо картинку, например, фоновое изображение, надо вывести иллюстрацию, то это можно сделать при помощи метода DrawImage. Инструкция вызова метода DrawImage в общем случае выглядит так:

Canvas.DrawImage(bitmap, srcRect, dstRec, op)

где:

- bitmap – объект TBitmap, содержащий изображение;
- srcRect – объект TRectF, содержащий координаты и размер фрагмента области bitmap, которую надо вывести на поверхность Canvas;
- dstRec – объект TRectF, содержащий координаты и размер области Canvas, куда надо вывести изображение;
- op – коэффициент непрозрачности при отображении иллюстрации.

Следует обратить внимание, для того чтобы иллюстрация на поверхности Canvas отображалась без искажения, значение свойства WrapMode компонента Image, на который выводится изображение, не должно быть Stretch.

В качестве примера в листинге 6.4 приведена процедура обработки сигнала от таймера, которая выводит в поле компонента Image изображение солнца. Процедура запускается автоматически в начале работы программы, выводит требуемое изображение и останавливает таймер.

Листинг 6.4. Использование метода DrawImage

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
  bmpfile: string;  // файл картинки
  bitmap : TBitmap; // картинка
  srcRect : TRectF; // область картинки, которую надо вывести
  dstRect : TRectF; // область поверхности, куда надо вывести картинку
  p: TPoint;        // координата точки, куда надо вывести картинку
begin
  p := TPoint.Create(20,20);
  bmpFile := Tpath.Combine(TPath.GetHomePath, 'sun_2.png');
  if ( System.SysUtils.FileExists( bmpFile) = true) then
      begin
        bitmap := Tbitmap.CreateFromFile(bmpFile);
        srcRect := TRectF.Create(0,0, bitmap.Width, bitmap.Height);
        dstRect := TRectF.Create(p.X, p.Y, p.X + bitmap.Width, p.Y +  bitmap.Height);
        // вывод
        Image1.Bitmap.Canvas.BeginScene();
        Image1.Bitmap.Canvas.DrawBitmap(bitmap, srcRect, dstRect,1);
        Image1.Bitmap.Canvas.EndScene;
    end;
  timer1.Enabled := false;
end;
```

Галерея и камера

Следующая программа показывает, как получить изображение из галереи фотографий устройства или от фотокамеры. Выбранное пользователем изображение (или сделанная фотография) сначала преобразуется в эскиз – изображение 128x128 пикселей и сохраняется в рабочей папке приложения для дальнейшего использования, например, записи в базу данных. После чего эскиз отображается в поле компонента Image (рис. 6.5).

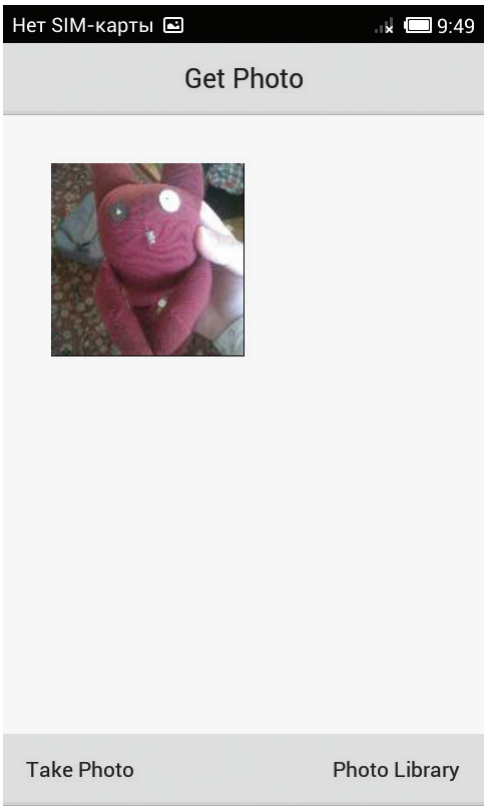


Рисунок 6.5. Эскиз фотографии, выбранной из галереи устройства

Форма приложения приведена на рис. 6.6. Для отображения эскиза используется компонент Image. Компонент ActionList1 используется для активизации процесса выбора фото из галереи или процесса съемки, а также для запуска процедур обработки действий пользователя (в окне выбора фотографии пользователь может нажатием на эскизе выбрать фотографию или отказаться от выбора, нажав кнопку **Отменить**).

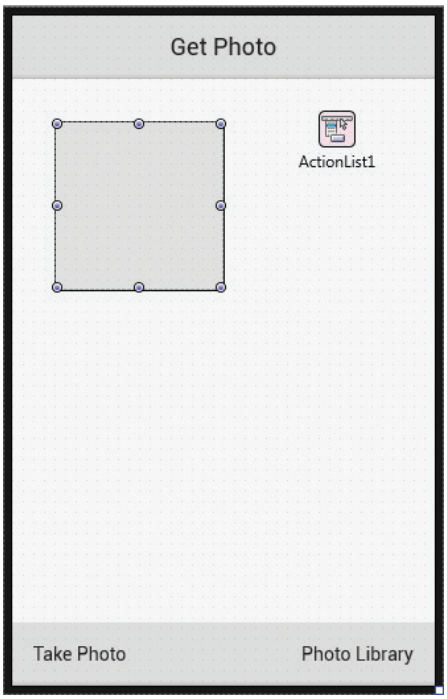


Рисунок 6.6. Форма приложения

После того как компоненты Button и ActionList будут помещены на форму, надо определить действия (action - действие), которые должны выполняться в результате нажатия кнопок. Чтобы задать действие для кнопки Button1, надо открыть список событий кнопки и в раскрывающемся списке **Action** выбрать **New Standard Action > Media Library > TTakePhotoFromLibraryAction** (рис. 6.7).

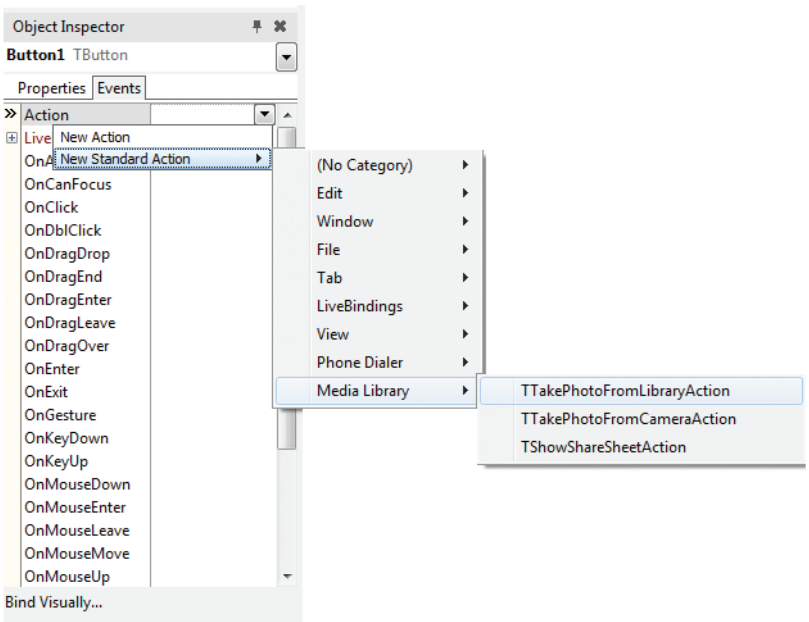


Рисунок 6.7. Выбор действия для кнопки

После того как будет определено действие для кнопки **Button1**, нужно определить процедуру обработки события которое завершает процесс выбора пользователем фотографии (или отказа от выбора). Чтобы это сделать, надо в окне **Structure** выбрать объект **TakePhotoFromLibraryAction1** (рис. 6.8) и обычным образом создать процедуру обработки события **DidFinishTakeng**.

Аналогичным образом создается действие для доступа к камере.

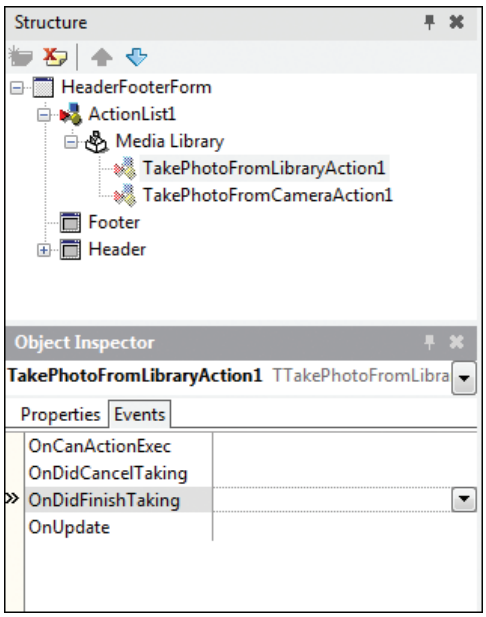


Рисунок 6.8. Настройка компонента ActionList

Конструктор формы и процедуры обработки событий приведены в листинге 6.5.

Листинг 6.5.

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Image1.Bitmap := TBitmap.Create(round(Image1.Width), round(Image1.Height));
  Image1.Bitmap.Clear(TAlphaColors.Lightgrey);
end;
procedure TMainForm.Image1Paint(Sender: TObject; Canvas: TCanvas;
  const ARect: TRectF);
var
    bitmap: TBitmap;
    bmpFile: string;
    rect: TrectF;
begin
    //   TPath.GetDocumentsPath - Мои документы
    //   TPath.GetHomePath - AddData\Roaming
    //
    bmpFile := Tpath.Combine(TPath.GetDocumentsPath, 'thumb.jpg');
    if ( System.SysUtils.FileExists( bmpFile) = true) then
      begin
          // v1. - работает!
```

```
          {
          bitmap := Tbitmap.CreateFromFile(bmpFile);
          rect := TrectF.Create(0,0, bitmap.Width, bitmap.Height);
          Image1.Bitmap.Canvas.BeginScene;
          Image1.Bitmap.Canvas.DrawBitmap(bitmap, rect, rect, 1.0);
          Image1.Bitmap.Canvas.EndScene;
           }
           //   v2  - Работает! Так быстрее!
            Thumbnail := Tbitmap.CreateFromFile(bmpFile);
            Image1.Bitmap.Assign(Thumbnail);
      end;
end;
// картинка из камеры (пользователь сделал фото)
procedure TMainForm.TakePhotoFromCameraAction1DidFinishTaking(Image: TBitmap);
begin
  { display the picture taken from the camera to the TImage control }
  image1.Bitmap.Assign(Image);
end;
// пользователь выбрал фото из галереи
procedure TMainForm.TakePhotoFromLibraryAction1DidFinishTaking(Image: TBitmap);
var
    x,y:integer;
    w,h: integer;
    TopLeft: TPoint;
    Rect:TRect;
    tmpBitmap: TBitmap;
    r: TRectf;
    fn: string;
begin
        // обрезка выполняется путем копирования
        // нужного фрагмента во временный Bitmap
        // определить положение и размер
        // копируемой области
        if (Image.Width > Image.Height) then
          begin
            // landscape картинка
            x := (Image.Width - Image.Height) div 2;
            y := 0;
            h := Image.Height;
            w := h;
          end
        else
          begin
            // portrait картинка
            y := (Image.Height - Image.Width) div 2;
            X := 0;
            w := Image.Width;
            h:= w;
          end;
        TopLeft:=Point(X, Y);
        Rect:=TRect.Create(TopLeft,w,h);
        tmpBitmap := TBitmap.Create(w,h);
        //tmpBitmap.Width := w;
        //tmpBitmap.Height := h;
        tmpBitmap.CopyFromBitmap(Image,Rect,0,0);
```

```
        // создать эскиз
        Thumbnail := tmpBitmap.CreateThumbnail(128,128);
        // сохранить эскиз в AppData\Roaming
        fn:=  Tpath.Combine(TPath.GetHomePath, 'thumb.jpg');
        Thumbnail.Canvas.Bitmap.SaveToFile(fn);
        // показать эскиз
        Image1.Bitmap.Assign(Thumbnail);
end;
```

Глава 7. 3D графика

Графическое пространство

Стартовой точкой для создания 3D приложения является шаблон 3D Application (рис. 7.1). Приложение, созданное на основе этого шаблона, изначально содержит форму, базовым классом которой является класс TForm3D. Класс TForm3D представляет собой графическое пространство, в которое можно поместить 3D объект, например, куб, сферу или цилиндр.

Следует обратить внимание, 3D объекты можно использовать и в обычных, не 3D приложениях, но в этом случае в качестве графического пространства надо использовать компонент Viewport3D.

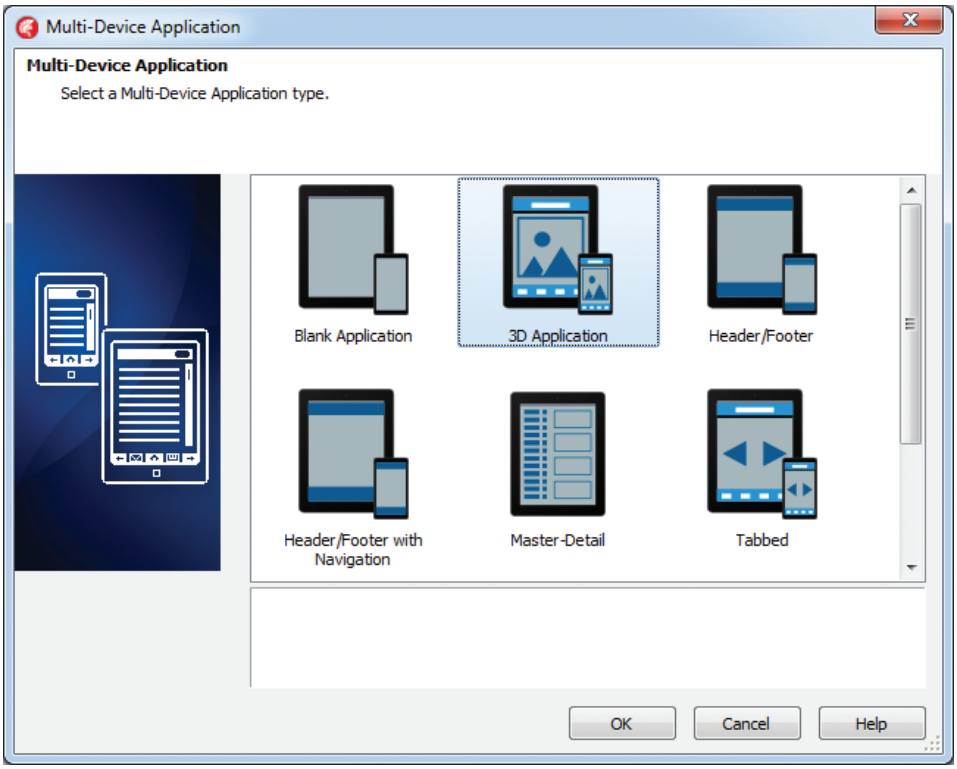


Рисунок 7.1. Начало работы над 3D графическим приложением

Координаты точки в пространстве и проекция

Положение объекта в пространстве определяется тремя координатами: X, Y и Z. Применительно к устройству, или экрану ось X направлена по горизонтали, ось Y – по вертикали, ось Z – перпендикулярно плоскости экрана.

Положение точки начала координат определяется типом проекции (Projection), применяемой для формирования изображения объекта на экране.

Возможны два варианта проекций: Camera и Screen.

Если применяется проекция Camera (тип проекции задается для каждого 3D объекта путем установки значения свойства Projection), то точка начала координат находится в центре поверхности экрана, при этом координата X возрастает слева направо, Y – снизу вверх, Z – в глубину экрана.

Если применяется проекция Screen, то точка начала координат находится в левом верхнем углу поверхности экрана, при этом координата X возрастает слева направо, Y – сверху вниз, Z – в глубину экрана.

Размеры и вид объекта

Размеры 3D объекта характеризуются шириной (Width), высотой (Height) и глубиной (Depth). Ширина задается (измеряется) по оси X, высота – по оси Y, глубина – по оси Z.

Изображение объекта на экране зависит не только от его размеров, но и от того, в какой точке экрана находится объект, а также от типа проекции, применяемой к объекту. В качестве примера на рис. 7.2 приведено изображение формы, в которой находятся девять одинаковых по размеру кубиков, (объектов TCube). Несмотря на то, что все объекты одинаковые, выглядят они по-разному, т.к. находятся в разных точках пространства, хотя и в одной плоскости (координата Z у всех объектов равно нулю). Изменение координаты Z также приводит к изменению вида объекта. Положительное значение Z отодвигает объект от экрана, объект уменьшается. Отрицательное значение Z приближает объект к наблюдателю, объект (изображение объекта) увеличивается.

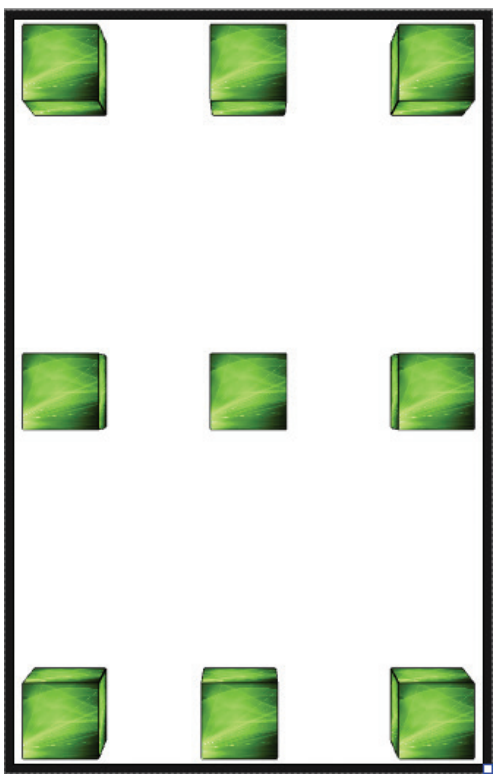


Рисунок 7.2. Вид объекта определяется не только его размером, но и положением в пространстве

Координаты и ориентация объекта в пространстве

Координаты объекта в пространстве определяются (задаются) координатами точки его центра. Положение точки центра зависит от формы объекта. Например, центр шара находится внутри шара, в точке, равноудаленной от всех точек поверхности, а центр куба – в точке пересечения линий, проведенных из вершин, находящихся в разных плоскостях.

Вид объекта на экране определяется не только его размерами и координатами, но и углами поворота «исходного» объекта вокруг осей координат. Изначально углы поворота считаются равными нулю. Изменение значения угла поворота вокруг оси X отклоняет объект вперед (положительное значение) или назад (отрицательное значение), по оси Y – вращает влево (положительное значение) или вправо (отрицательное значение), по оси Z – поворачивает против (положительное значение) или по часовой стрелке (отрицательное значение).

Фигуры

Среда разработки предоставляет в распоряжение программиста набор компонентов (геометрических фигур) из которых, как из конструктора, можно собрать любую объемную конструкцию. Компоненты геометрических фигур, находятся на вкладке 3D Shapes (рис. 7.3).

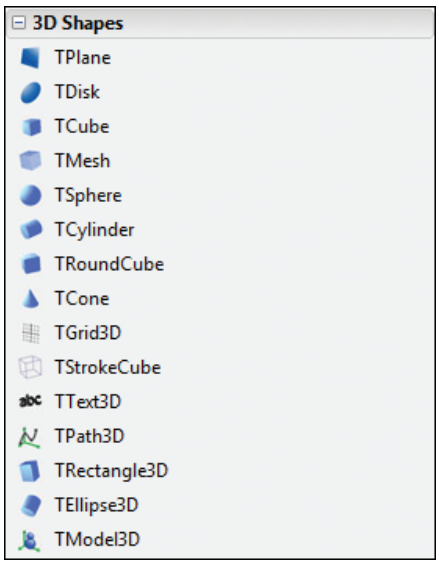


Рисунок 7.3. 3D фигуры

Чтобы фигура появилась на форме, ее значок надо перетащить с палитры компонентов в форму программы (или в поле компонента Viewport3D) и путем изменения свойств придать фигуре требуемый вид и положение в пространстве.

Основные свойства 3D объектов приведены в табл. 7.1.

Таблица 7.1. Основные свойства 3D объектов

Свойство	Описание
Width, Height, Depth	Размеры объекта, измеряемые вдоль осей X, Y и Z.
Projection	Тип проекции: Screen или Camera
MaterialSource	Определяет цвет объекта. Представляет собой ссылку на объект TTextureMaterialSource, TColorMaterialSource или TLightMaterialSource.
Position	Свойства Position.X, Position.Y и Position.Z определяют положение, координаты центра объекта в пространстве
RotationAngle	RotationAngle.X, RotationAngle.Y и RotationAngle.Z определяют углы поворота объекта соответственно вокруг осей X, Y и Z.

Значения свойств 3D объекта можно изменить обычным образом в окне Object Inspector. Значения свойств, определяющих размер (Width, Height, Depth), и свойства RotationAngle, можно изменить при помощи мыши, путем перемещения маркеров, появляющихся в результате выбора объекта в окне дизайнера формы (рис. 7.4). Чтобы переместить объект (изменить значения свойства Position) надо установить указатель мыши на левый нижний угол границы компонента, нажать левую кнопку мыши и, удерживая кнопку нажатой, перетащить объект в нужную точку.

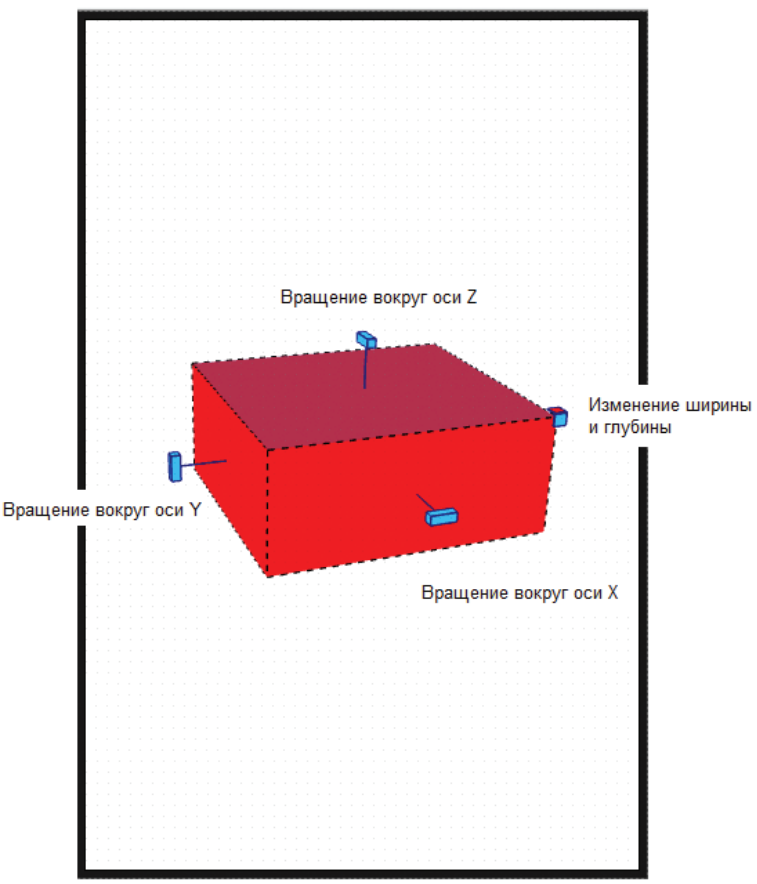


Рисунок 7.4. Изменить вид 3D объекта можно путем перемещения маркеров

Материал

В общем случае вид объекта реального мира определяется материалом, из которого он сделан. Например, куб или шар может быть сделан из сосны или дуба, алюминия или стали, мрамора или малахита. Объект может быть окрашен весь каким-либо одним цветом, на его поверхность может быть нанесен узор, имитирующий какой-либо материал, или рисунок.

Для получения сходства с объектами реального мира 3D объекты могут быть текстурированы или окрашены. Способ окраски и текстура представляют собой объекты TColorMaterialSource TTextureMaterialSource. Их свойства приведены в табл. 7.2 и табл. 7.3.

Таблица 7.2. Свойства объекта TColorMaterialSource

Свойство	Описание
Name	Имя объекта. Используется в качестве значения свойства MaterialSource объекта, представляющего собой 3D фигуру.
Color	Задает цвет окраски объекта. Значение задается путем выбора цвета в списке.

Таблица 7.3. Свойства объекта TTextureMaterialSource

Свойство	Описание
Name	Имя объекта. Используется в качестве значения свойства MaterialSource объекта, представляющего собой 3D фигуру
Texture	Текстура – рисунок, который отображается на поверхности объекта. Задается путем выбора фрагмента рисунка, находящегося в файле.

Цвет и текстура

По умолчанию все объекты окрашены красным цветом. Чтобы изменить цвет объекта, сначала нужно выполнить подготовительную работу: добавить в форму объект TColorMaterialSource и задать его цвет (присвоить требуемое свойству Color). Затем имя объекта TColorMaterialSource надо указать в качестве значения свойства MaterialSource того объекта, цвет которого надо изменить.

Текстурирование объекта выполняется аналогично. Чтобы задать текстуру надо в списке команд, который появляется в результате позиционирования мыши на кнопке с тремя точками, находящейся в поле значения свойства Texture, выбрать команду Edit (рис. 7.5). В открывшемся окне Bitmap Editor (рис. 7.6) надо нажать кнопку **Load** и выбрать иллюстрации. Если в качестве текстуры нужен фрагмент картинки, то надо нажать кнопку Crop и установить появившуюся рамку на нужный фрагмент.

Пример использования структуры для окраски куба приведен на рис. 7.7.

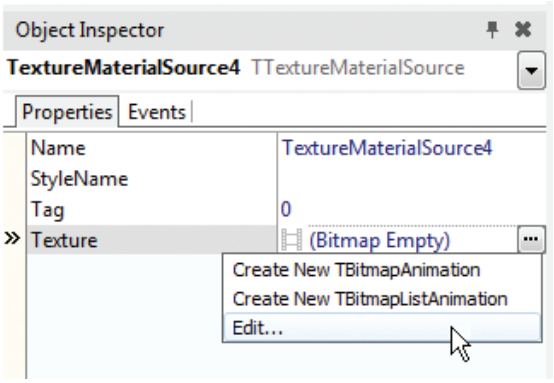


Рисунок 7.5. Выбор текстуры (шаг 1)

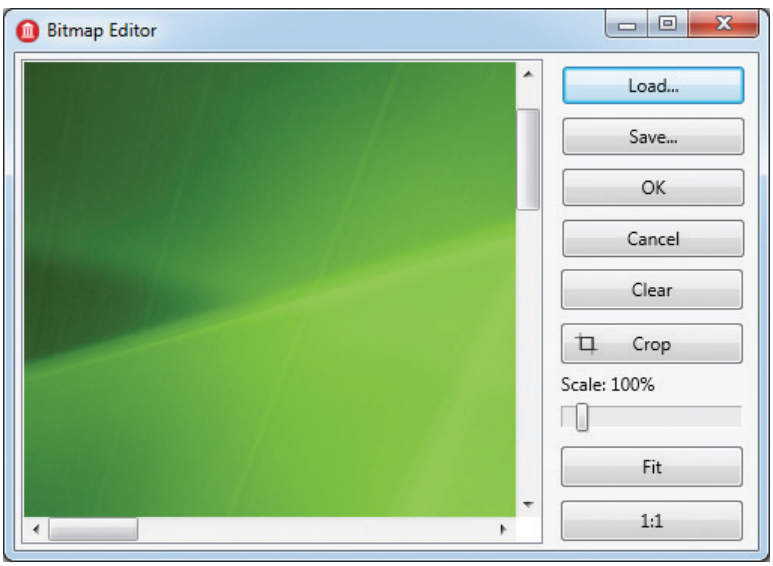


Рисунок 7.6. Выбор текстуры (шаг 2)

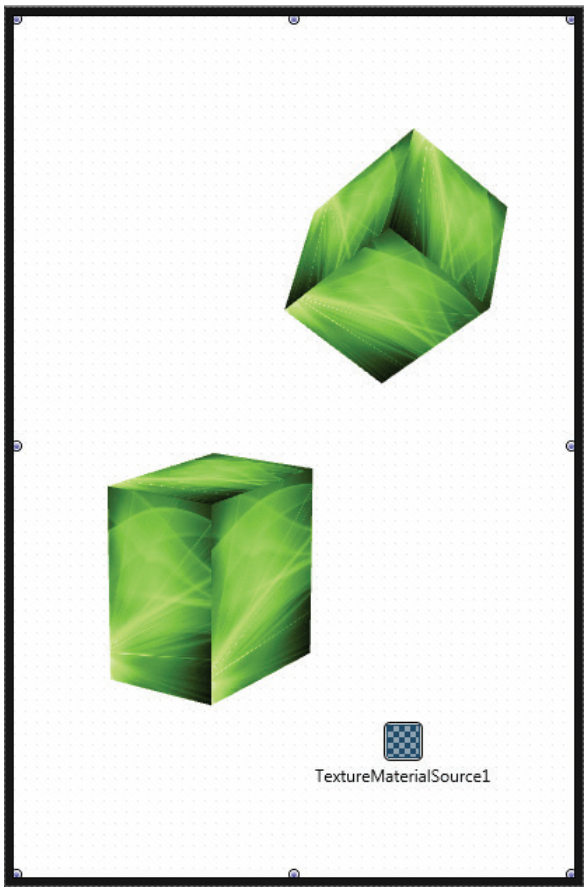


Рисунок 7.7. Пример использования текстуры для окраски куба

Пример 3D приложения

В качестве примера рассмотрим 3D приложение, в окне которого отображается столбчатая 3D диаграмма (рис. 7.8). Исходными данными для программы являются абсолютные значения по трем категориям. Приложение вычисляет долю каждой категории в общей сумме и строит диаграмму.

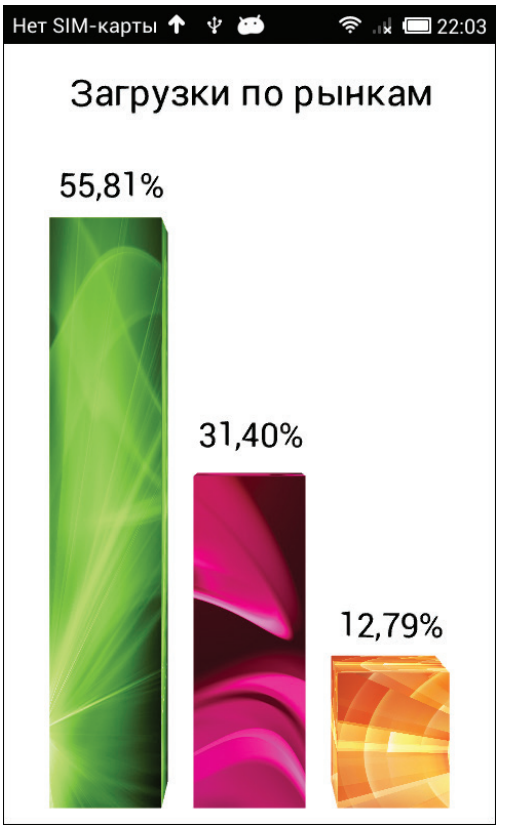


Рисунок 7.8. 3D диаграмма

Особенность программы в том, что диаграмма формируется динамически, во время работы программы путем изменения свойств 3D объектов.

Форма программы приведена на рис. 7.9. Следует обратить внимание, что невизуальные компоненты в форме 3D приложения не отображаются. Список всех компонентов приложения можно увидеть в окне Structure. Значения свойств компонентов приведены в табл. 7.4. Компоненты рекомендуется добавлять в форму и настраивать в том порядке, в котором они следуют в таблице. В таблице приведены значения только тех свойств, которые влияют на процесс построения диаграммы. Значения остальных свойств, например, определяющих положение объектов можно установить так, чтобы форма выглядела примерно так, так как показано на рисунке.

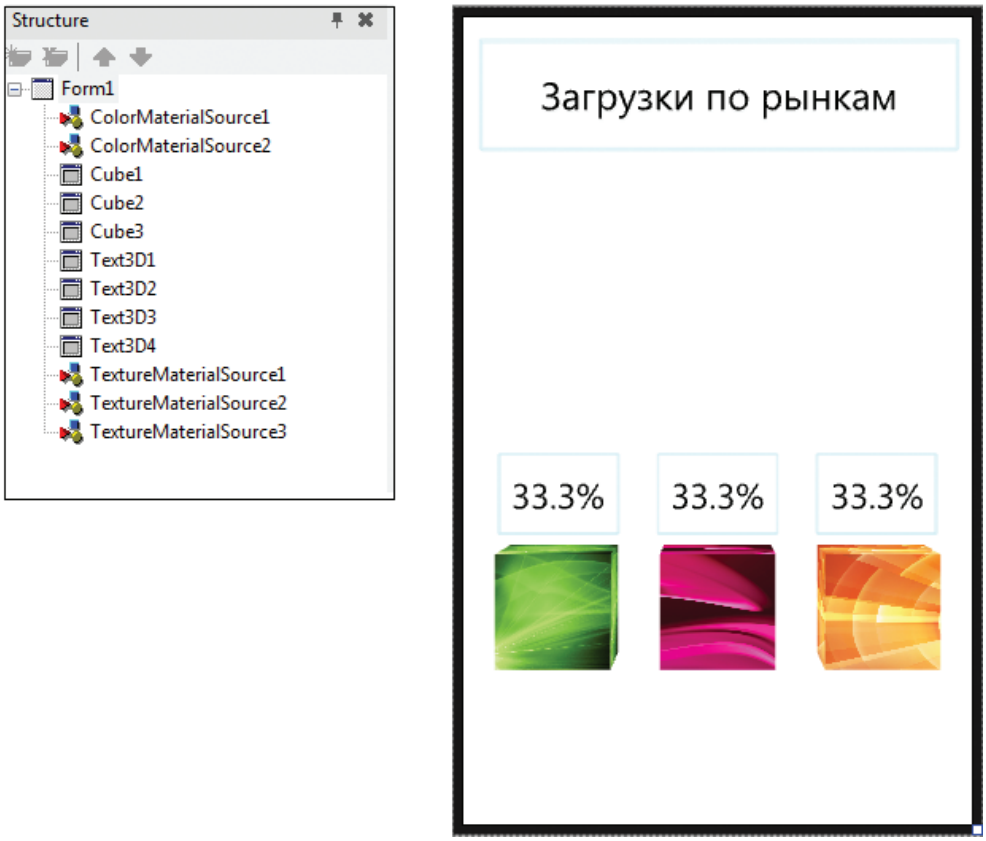


Рисунок 7.9. Список компонентов и форма

Таблица 7.4. Значения свойств компонентов

Компонент	Свойство	Значение
TextureMaterialSource1	Name	TextureMaterialSource1
	Texture	(Bitmap)
TextureMaterialSource2	Name	TextureMaterialSource2
	Texture	(Bitmap)
TextureMaterialSource3	Name	TextureMaterialSource3
	Texture	(Bitmap)
Cube1	Projection	Screen
	Width	70
	Height	70
	Depth	70
	MaterialSource	TextureMaterialSource1
	Position.X	60
	Position.Y	370
	Pozition.Z	0

Компонент	Свойство	Значение
Cube2	Projection	Screen
	Width	70
	Height	70
	Depth	70
	MaterialSource	TextureMaterialSource2
	Position.X	160
	Position.Y	370
	Pozition.Z	0
Cube3	Projection	Screen
	Width	70
	Height	70
	Depth	70
	MaterialSource	TextureMaterialSource3
	Position.X	260
	Position.Y	370
	Pozition.Z	0
ColorMaterialSource1	Name	ColorMaterialSource1
	Color	Black
ColorMaterialSource2	Name	ColorMaterialSource2
	Color	Whitesmoke
Text3D1	Projection	Screen
	Width	75
	Height	50
	MaterialSource	ColorMaterialSource1
	MaterialShaftSource	ColorMaterialSource2
	Font.Size	22
	Text	33%
Text3D2	Projection	Screen
	Width	75
	Height	50
	MaterialSource	ColorMaterialSource1
	MaterialShaftSource	ColorMaterialSource2
	Font.Size	22
	Text	33%
Text3D3	Projection	Screen
	Width	75
	Height	50
	MaterialSource	ColorMaterialSource1
	MaterialShaftSource	ColorMaterialSource2
	Font.Size	22
	Text	33%

Компонент	Свойство	Значение
Text3D4	Projection	Screen
	Width	300
	Height	50
	MaterialSource	ColorMaterialSource1
	MaterialShaftSource	ColorMaterialSource2
	Font.Size	24
	Text	Загрузки по рынкам

Текст программы приведен в листинге 7.1. Конструктор формы выполняет обработку данных: вычисляет долю каждой категории в общей сумме (именно эта информация отображается на диаграмме), а также определяет максимальное значение ряда данных. Максимальное значение используется при построении диаграммы. Столбик, соответствующий максимальному значению, должен занимать всю доступную для построения диаграммы область окна. Высота области отображения диаграммы вычисляется как разность высоты окна и суммы высоты области отображения заголовка, высоты отображения подписи данных и высоты области отступа от нижней границы окна до нижней точки столбика.

Построение диаграммы выполняет процедура Diagram3D. Обратите внимание, как вычисляется координата столбца. Вспомните, что для того чтобы построить 3D объект, надо знать координаты центра объекта. Программа сначала вычисляет высоту столбика, а затем – координаты центра объекта.

Процесс построения диаграммы активизирует процедура обработки события Resize формы. Это событие возникает в начале работы программы, а также при изменении ориентации устройства. Поэтому, независимо от ориентации устройства диаграмма на экране отображается целиком (рис. 7.10).

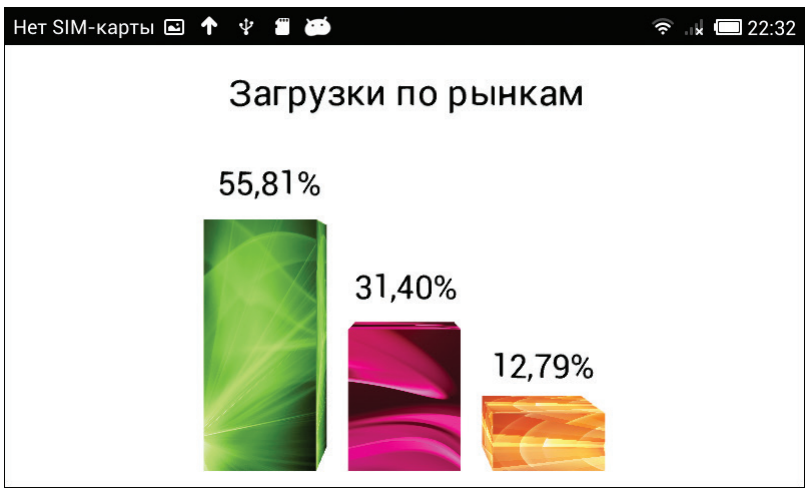


Рисунок 7.10. Вид диаграммы при изменении ориентации устройства

Листинг 7.1. 3D диаграмма

```
implementation
{$R *.fmx}
{$R *.Windows.fmx MSWINDOWS}
const
  HB=3;
var
  d: array[1..HB] of integer;    // ряд данных
  p: array[1..HB] of double;     // доля категории в общей сумме
  max: integer; // номер макс. эл-та
procedure TForm1.Form3DCreate(Sender: TObject);
var
    sum: double;
    i: integer;
begin
    d[1]:=48;
    d[2]:=27;
    d[3]:=11;
    sum:=0;
    max := 1;
    // вычислить сумму и
    // найти max. элемент ряда данных
    for i:= 1 to HB do
    begin
      sum:= sum+d[i];
      if d[i] > d[max] then
          max := i;
    end;
    // вычислить долю каждой категории
    for i:= 1 to HB do
    begin
      p[i] := d[i]/sum*100;
    end;
end;
Procedure TForm1.Diagram3D;
var
   k: double;     // коэф. масштабирования
begin
    // Столбик, соответствующий максимальному значению,
    // должен занимать всю доступную по высоте область.
    // Высота остальных должна быть пропорциональна его высоте.
    // Вычислим коэф. масштабирования.
    // В скобках - высота столбца, соотв. максимальному значению
    k:= (Form1.Height - Text3d4.Height - Text3d1.Height - 20)/d[max];
    Cube1.Height := d[1]*k;
    Cube2.Height := d[2]*k;
    Cube3.Height := d[3]*k;
    // Вычислить координаты второго столбца
    // чтобы он был в центре по X
    Cube2.Position.X := Form1.Width /2;
    Cube2.Position.Y := Form1.Height - Cube2 .Height/2 - 20;
    // Подпись данных
    Text3d2.Text := Format('%5.2f%%',[p[2]]);
    Text3d2.Position.X := Cube2.Position.X;
```

```
    Text3d2.Position.Y := Cube2.Position.Y - Cube2.Height/2 - Text3d1.Height/2;
    // Первый столбец сдвинуть влево относительно среднего
    Cube1.Position.X := Cube2.Position.X - Cube2.Width/2  - Cube1.Width/2 - 20;
    Cube1.Position.Y := Form1.Height - Cube1 .Height/2 - 20;
    Text3d1.Text := Format('%5.2f%%',[p[1]]);
    Text3d1.Position.X := Cube1.Position.X;
    Text3d1.Position.Y := Cube1.Position.Y - Cube1.Height/2 - Text3d2.Height/2;
    // Третий сдвинуть вправо относительно второго
    Cube3.Position.X := Cube2.Position.X + Cube2.Width/2  + Cube3.Width/2 + 20;
    Cube3.Position.Y := Form1.Height - Cube3 .Height/2 - 20;
    Text3d3.Text := Format('%5.2f%%',[p[3]]);
    Text3d3.Position.X := Cube3.Position.X;
    Text3d3.Position.Y := Cube3.Position.Y - Cube3.Height/2 - Text3d3.Height/2;
    Text3d4.Text := 'Загрузки по рынкам';
    Text3d4.Position.X := Form1.Width/2;
end;
procedure TForm1.Form3DResize(Sender: TObject);
begin
      // построить диаграмму
      Diagram3D;
end;

end.
```

Глава 8. Анимация

Анимация — процесс придания объектам способности изменяться, например, перемещаться, вращаться, менять цвет.

Delphi предоставляет в распоряжение программиста набор FireMonkey компонентов (они находятся на вкладке Animations) позволяющих анимировать практически любой объект: геометрическую фигуру, иллюстрацию, элемент пользовательского интерфейса. Следует обратить внимание, анимировать можно не только отдельный объект, но и группу объектов, например, находящиеся на панели поля ввода, списки и командные кнопки.

Анимация объекта выполняется путем анимирования его свойств. Например, чтобы заставить «расти» прямоугольник (элемент столбчатой) диаграммы, надо анимировать свойство Height. Чтобы объект, например, картинка, начал двигаться, надо анимировать свойства, определяющие положение объекта: Position.X и Position.Y.

Свойства объекта, которые могут быть анимированы, помечены значком кинопленки (рис. 8.1). Приведенный пример показывает, что у объектов много свойств, которые могут быть анимированы. Это свойства, определяющие размер компонента (Width, Height), положение (Position) и ориентацию компонента (RotationCenter и RotationAngle), цвет (Color), степень непрозрачности (Opacity) и другие.

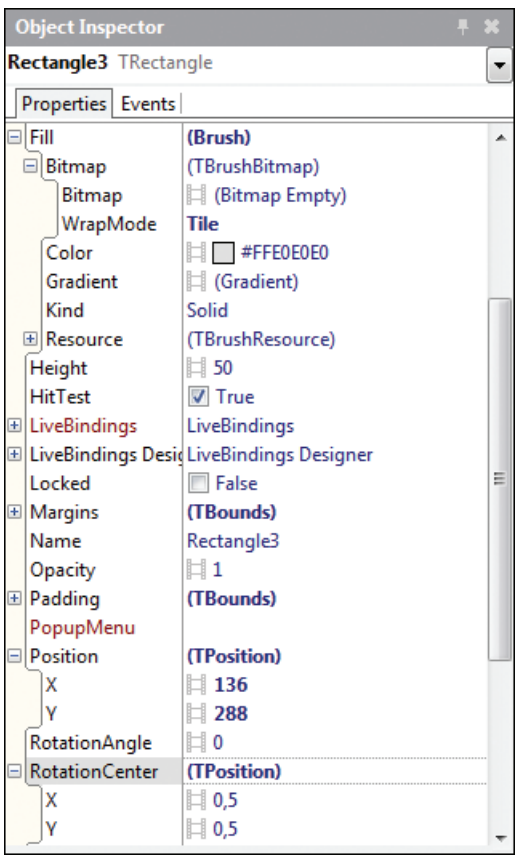


Рисунок 8.1. Свойства, которые могут быть анимированы, помечены значком кинопленки

Компоненты Animation

Компоненты, обеспечивающие анимацию свойств объектов, находятся на вкладке Animations палитры компонентов (рис. 8.2).

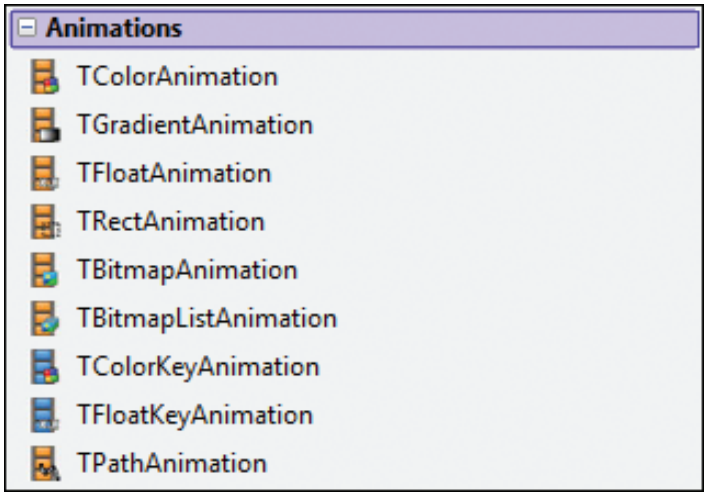


Рисунок 8.2. Компоненты Animations

Компонент ColorAnimation позволяет изменить цвет объекта, например, цвет закраски области или контура геометрической фигуры. Компонент GradientAnimation позволяет анимировать параметры градиентной закраски. Компоненты FloatAnimation и PathAnimation предназначены для анимации свойств, определяющих положение объекта. Компонент FloatAnimation используется для получения эффекта простого прямолинейного движения, PathAnimation – движения по сложной траектории. Компонент RectAnimation позволяет выполнить трансформацию областей. Эффекты анимации битовых образов, например, появление или смену картинки в поле компонента Image, можно получить при помощи компонента BitmapAnimation.

Ключевые свойства компонентов Animation приведены в табл. 8.1.

Таблица 8.1. Ключевые свойства компонентов Animation

Свойство	Описание
PropertyName	Анимируемое свойство объекта. Например, для FloatAnimation – Position.X.
StartValue	Начальное значение анимируемого свойства. Например, для FloatAnimation, X координата точки, из которой объект должен начать движение. Если установлен флажок StartFromCurrent, то в качестве начального значения анимируемого свойства используется его текущее значение свойства PropertyName.
StopValue	Конечное значение анимируемого свойства. Например, для FloatAnimation - координата точки, в которую должен переместиться объект.
Interpolation	Задает правило изменения значения анимируемого свойства как функцию времени. Например: Linear – линейная зависимость; Exponential – экспоненциальная. Замечание: вид функции зависит также от значения свойства AnimationType. Если значение Interpolation равно Exponential и значение AnimationType равно atIn, то экспонента выпуклая вниз (медленный рост), если значение AnimationType равно atOut, то экспонента выпуклая вверх (быстрый рост).

Свойство	Описание
AnimationType	Задает тип анимации (см. описание свойства Interpolation)
StartFromCurrent	Флаг использования текущего значения анимируемого свойства в качестве начального в процессе анимации: True – использовать текущее значение свойства, False – использовать значение StartValue.
Duration	Длительность анимации в секундах. Определяет время изменения анимируемого свойства со значения StartValue на значение StopValue. Например, для FloatAnimation – время перемещения объекта из точки, заданной параметром StartValue, в точку, заданную параметром StopValue.
Delay	Время задержки запуска процесса анимации от момента ее активации
Enabled	Признак автоматического запуска анимации
Inverse	Свойство управляет направлением воспроизведения анимации. Если значение свойства равно False, то анимация воспроизводится в прямом направлении (анимируемое свойство меняется от значения StartValue до StopValue). Если значение свойства равно True, то анимация воспроизводится в обратном направлении (анимируемое свойство меняется от значения StopValue до StartValue).
AutoReverse	Признак автоматического воспроизведения анимации в обратном порядке по достижении конца процесса анимации
Loop	Признак циклического воспроизведения анимации.

Если флаг Enabled компонента Animation установлен, то анимация запускается автоматически. Анимацию можно запустить и остановить в нужный момент программно. Метод Start объекта Animation запускает анимацию, Stop – останавливает.

Компонент Animation генерирует события Process и Finish. Событие Process возникает в начале процесса воспроизведения анимации, событие Finish – в конце.

FloatAnimation

Следующий пример демонстрирует использование компонента FloatAnimation для организации интерфейса приложения.

На рис. 8.3 приведена форма приложения работы с базой данных контакты (основную часть окна занимает компонент ListBox). Панель, содержащая компоненты предназначенные для ввода информации, не отображается (значение свойства Visible компонента Panel1 равно False). Если значению свойства компонента Panel1 присвоить True, то форма будет выглядеть так, как показано на рис. 8.4.

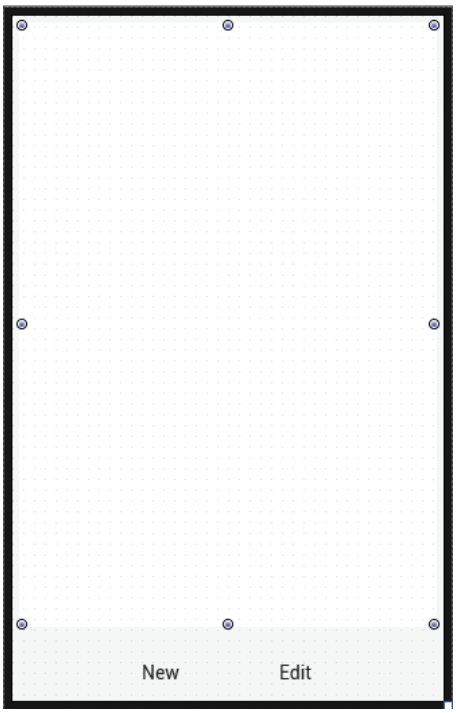


Рисунок 8.3. Форма работы с базой данных Контакты

Рисунок 8.4. Панель ввода информации

Как управлять доступностью (видимостью) панели путем изменения свойства Visible понятно. Чтобы панель с компонентами ввода по нажатию кнопок New и Edit выезжала, например, снизу экрана, а при нажатии кнопки OK уезжала обратно, надо анимировать свойство Position.Y компонента Panel1.

Чтобы анимировать свойство, надо в списке значений свойства выбрать Create New TFloatAnimation (рис. 8.5).

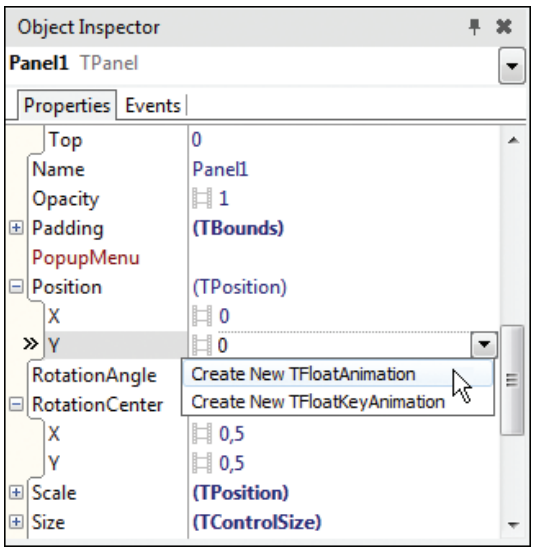


Рисунок 8.5. Создание объекта TFloatAnimation для свойства, которое надо анимировать

После этого надо задать параметры анимации – свойства компонента FloatAnimation1. В рассматриваемом примере анимация заключается в том, что панель выезжает из-за нижней границы формы, поэтому начальное значению координаты (свойству Start Value) следует присвоить значение равное значению свойства Height формы. Конечное значение (StopValue) определяет положение панели в конце процесса анимации, ему надо присвоить значение 0.

Значения свойств компонента FloatAnimation1 приведены в табл. 8.2.

Таблица 8.2. Значения свойств компонента FloatAnimation1

Свойство	Значение
PropertyName	Position.Y
StartValue	510
StopValue	0
StartFromCurrent	False
Duration	0,3
Delay	0
Inverse	False
Enabled	False
AutoReverse	False
Loop	False

Следует обратить внимание, в окне дизайнера формы панель находится в видимой части экрана. В начале работы программы она должна быть внизу, за границей формы. Переместить панель в нужную позицию можно в конце работы над формой, присвоив соответствующее значение свойству Position.Y. А можно возложить эту задачу на конструктор.

В листинге 8.1 приведен конструктор и процедуры обработки событий на командных кнопках, которые демонстрируют процесс запуска анимации. Компонент FloatAnimation используется для воспроизведения анимации в прямом и обратном направлениях. В прямом направлении значение свойства Position.Y меняется от значения StartValue до StopValue. В обратном направлении значение свойства Position.Y меняется от значения StopValue до StartValue. Направлением воспроизведения управляет свойство Inverse.

Листинг 8.1.

```
// конструктор
procedure TForm1.FormCreate(Sender: TObject);
begin
      Panel1.Align := TAlignLayout.None;
      Panel1.Position.Y := Panel1.Height;
      FloatAnimation1.StartValue := Panel1.Height
end;

// нажатие кнопки New
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Text := 'New contact';
  // воспроизведение анимации в прямом направлении
    FloatAnimation1.Inverse := False;
    FloatAnimation1.Start;
end;

// нажатие кнопки Edit
procedure TForm1.Button2Click(Sender: TObject);
begin
    Label1.Text := 'Edit contact';
    FloatAnimation1.Inverse := False;
    FloatAnimation1.Start;
end;

// нажатие кнопки OK
procedure TForm1.Button3Click(Sender: TObject);
begin
  // воспроизведение анимации в обратном направлении
     FloatAnimation1.Inverse := True;
     FloatAnimation1.Start;
end;
```

PathAnimation

Компонент FloatAnimation позволяет реализовать, в том числе, движение объекта по одной или, если анимировать обе координаты объекта, двум координатам. Траектория движения объекта определяется выбранным методом интерполяции.

Более широкими возможностями реализации траектории движения обладает компонент PathAnimation. Основное отличие PathAnimation от FloatAnimation состоит в том, что траектория движения объекта задается не координатами начальной и конечной точек, а структурой данных Path, описывающей траекторию движения объекта как последовательность участков, состоящих в общем случае из кривых Безье и прямых линий.

При описании траектории движения используется синтаксис Move and Draw (https://msdn.microsoft.com/en-us/library/dn535793.aspx). На рис. 8.6 приведены примеры кривых и их описание.

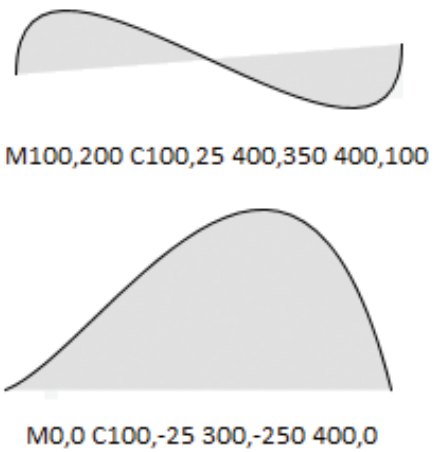


Рисунок 8.6. Кривые Безье и их описание

Краткий список команд, при помощи которых можно описать практически любую траекторию, приведен в табл. 8.3. Параметры команд, координаты точек, задаются двумя разделенными запятой числами.

Таблица 8.3. Команды описания траектории движения объекта

Команда	Описание
M *startpoint*	Начало рисования новой фигуры
L *endpoint*	Рисует линию в указанную точку
H *x*	Рисует горизонтальную линию в точку с указанной X координатой
V *y*	Рисует вертикальную линию в точку с указанной Y координатой
C *controlPoin1 controlPoint2 end-Point*	Кубическая кривая Безье (Cubic Bézier curve)
Q *controlPoin endpoint*	Квадратичная кривая Безье (Quadratic Bézier curve)
Z	Соединяет текущую точку с точкой начала (замыкает контур)

Чтобы анимировать объект, например, Image, надо поместить компонент PathAnimation на форму, затем в окне Structure перетащить изображение объекта PathAnimation на значок объекта, который надо анимировать (результат выполнения описанных действий приведен на рис. 8.7).

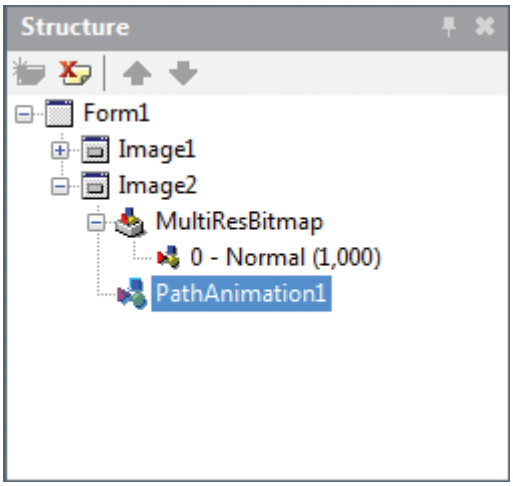


Рисунок 8.7. Компонент PathAnimation надо поместить на объект

После этого надо настроить компонент Animation. Чтобы задать траекторию движения объекта, надо сделать щелчок на находящейся в строке свойства Path кнопке активизации редактора свойства и в появившемся окне Path Designer ввести описание траектории (рис. 8.8). Координаты точек, описывающих траекторию движения, отсчитываются от точки текущего положения объекта. Именно поэтому первой командой в описании траектории должна быть команда M 0,0.

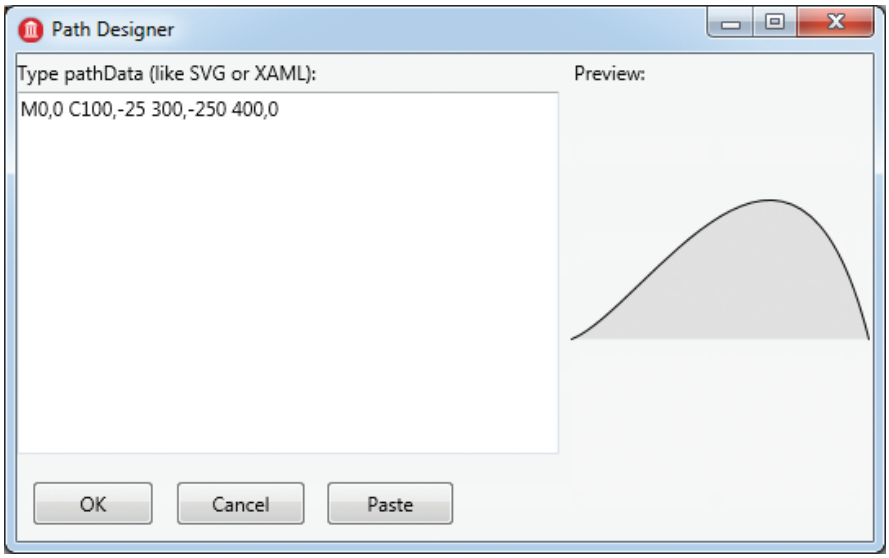


Рисунок 8.8. Ввод описания траектории движения объекта

Глава 9. Базы данных в мобильных приложениях

В Multi-Device приложениях для работы с базами данных программист может использовать компоненты dbExpress, dbGo и FireDAC. Эта глава посвящена использованию FireDAC компонентов. В ней на примере работы программы со встроенной базой данных SQLite **Расходы** демонстрируется назначение и особенности использования компонентов FireDAC, технология связывания данных.

В мобильных приложениях для хранения данных используют СУБД, которые принято называть встроенными (embedded – встроенный, включенный, интегрированный). С точки зрения архитектуры, встроенная СУБД является обычной, хотя и, как правило, с меньшими возможностями СУБД, построенной на основе SQL сервера. В качестве встроенной СУБД в Multi-Device приложениях можно использовать сервер IBLite или свободно распространяемый сервер баз данных SQLite.

FireDAC компоненты доступа к данным

Основные компоненты FireDac находятся на вкладке FireDac палитры компонентов, они обеспечивают доступ к данным и манипулирование ими. Компонент FDConnection обеспечивает соединение с базой данных, компонент FDTable – доступ к таблицам, компонент FDQuery обеспечивает взаимодействие с базой данных посредством SQL запросов.

Компоненты отображения данных

Наиболее часто информация в мобильных приложениях представляется в виде списков или в комбинированном формате. В виде списка обычно представляется содержимое ключевых полей. При комбинированном способе ключевая информация отображается в виде списка, а для отображения всей, подробной информации об объекте используется форма или таблица.

Для представления информации в виде списка можно использовать компоненты ListBox и ListView. Для представления информации в виде таблицы используется компонент StringGrid (на вкладке Grids).

Создание базы данных

База данных представляет собой файл в котором находятся таблицы, содержащие информацию и, как правило, она создается до начала работы над приложением при помощи соответствующей СУБД. Базу данных SQLite, которая будет использоваться на этапе разработки приложения, можно создать при помощи FireDac Explorer. Задачу создания рабочей базы данных можно возложить на приложение работы с базой данных: при первом запуске программы на устройстве приложение создаст базу данных.

Процесс создания базы данных SQLite рассмотрим на примере. Создадим при помощи FireDAC Explorer базу данных **expenses** (**расходы**), которая представляет собой одну единственную таблицу expenses (табл. 9.1).

Таблица 9.1. Таблица expenses базы данных `Расходы`

Поле	Тип	Описание	Примечание
Date	DATETIME	Дата	NOT NULL
Sum	MONEY	Сумма	NOT NULL
Description	TEXT	Описание	NOT NULL

Замечание. Перед тем как приступить к созданию базы данных рекомендуется: создать папку для проекта разработки приложения. Именно в нее надо будет поместить файл базы данных.

Утилита FireDAC Explorer запускается выбором соответствующей команды в меню **Tools**.

Чтобы создать новое соединение с базой данных, надо в меню **File** выбрать команду **New > Connection Definition**, ввести в поле редактирования имя соединения (рис. 9.1) и в меню **Edit** выбрать команду **Apply**. Соединение будет создано.

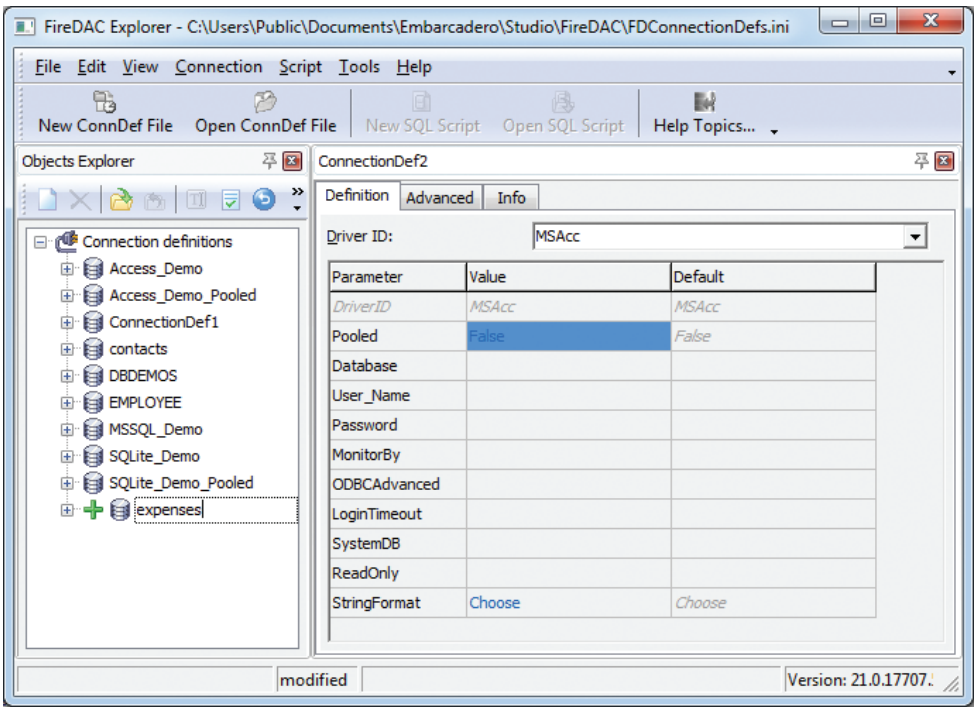


Рисунок 9.1. Создание соединения

После этого на вкладке **Definition,** в списке **Driver ID,** надо выбрать **SQLite** и в поле **Value** строки **Database** ввести полное (с указанием пути) имя файла базы данных. Чтобы ввести имя файла, надо сделать щелчок на значке папки, находящемся в поле **Value** строки **Database**, в стандартном окне **Открыть** найти папку проекта приложения работы с базой данных, и в поле **Имя файла** ввести имя файла базы данных. После этого в качестве значения параметра **LockingMode** надо указать значение **Normal**.

Окно **FireDac Explorer** в конце процесса создания соединения с базой данных SQLite Expenses приведено на рис. 9.2.

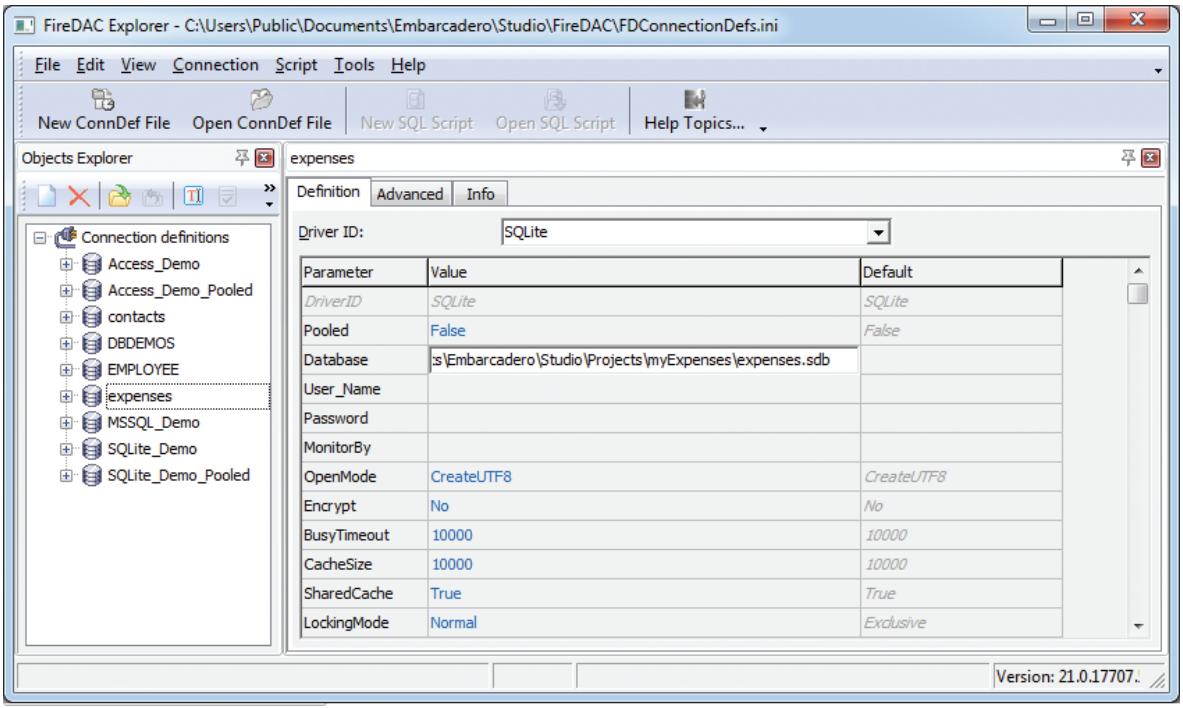


Рисунок 9.2. База данных Expenses и соединение с ней созданы

После того как база будет создана, можно направить серверу SQL команду, обеспечивающую создание в базе данных таблицы expenses:

CREATE TABLE IF NOT EXISTS Expenses (Date DATETIME NOT NULL, Sum MONEY NOT NULL, Description TEXT NOT NULL)

Чтобы направить серверу SQL команду, надо:

1. Открыть соединение – сделать щелчок на находящемся перед именем соединения значке раскрывающегося списка и в появившемся окне **Login** нажать кнопку **OK** (т.к. во время создания базы данных имя пользователя и пароль не задавались, то для подключения к базе данных они не нужны). Обратите внимание, в раскрывшемся списке отображается структура базы данных, с которой установлено соединение.
2. Нажать кнопку **New SQL Script** и в окне набрать SQL нужную команду (рис. 9.3)
3. Сделать щелчок на кнопке **Run** или нажать клавишу F9. Обратите внимание, командная кнопка **Run** доступна только в том случае, если курсор находится внутри теста команды (перед последней закрывающей скобкой).

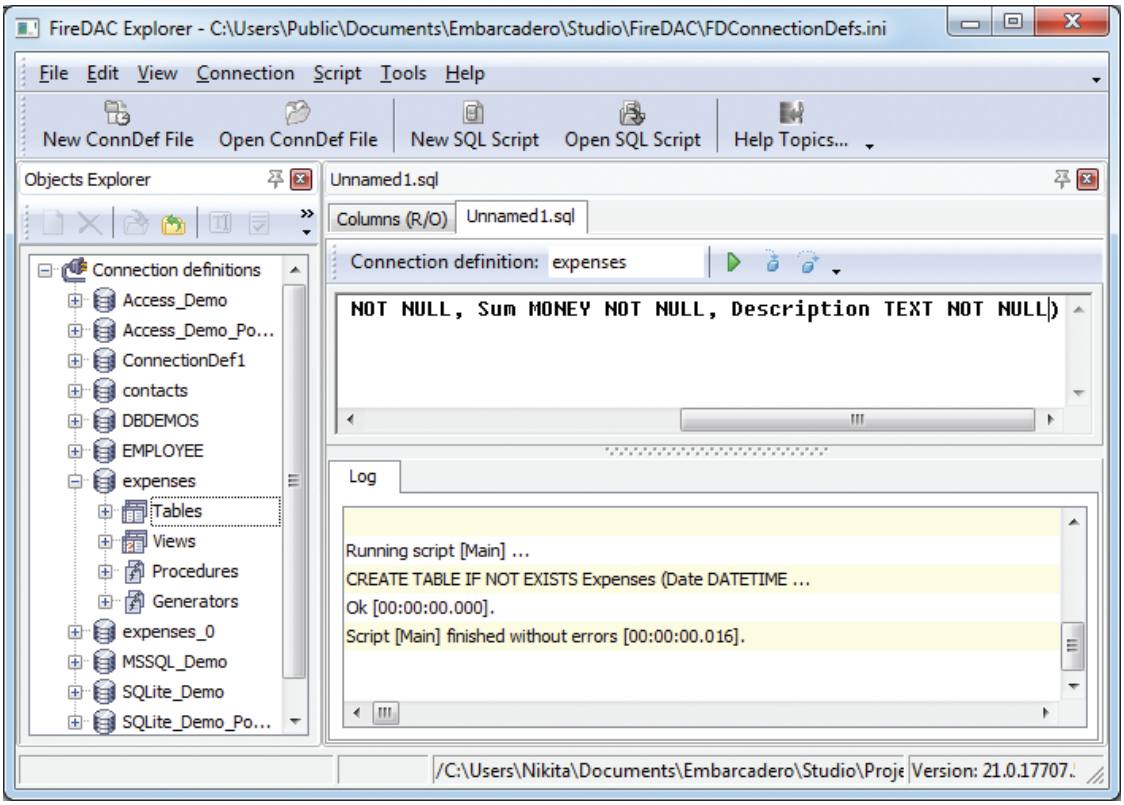


Рисунок 9.3. Создание таблицы в базе данных

Результат выполнения команды можно увидеть, раскрыв список таблиц базы данных (соединения) expenses, предварительно выбрав в меню **View** команду **Refresh**.

Аналогичным образом можно направить серверу команды добавления в таблицу expenses нескольких записей. Например, последовательность команд (SQL Script) добавления трех записей и просмотра таблицы выглядит так:

```
INSERT INTO Expenses Values('2014-11-20','170','Кофе');
INSERT INTO Expenses Values('2014-11-20','220','Круасан');
INSERT INTO Expenses Values('2014-11-20','250','Мороженое');
SELECT * FROM Expenses
```

Увидеть данные, находящиеся в таблице базы данных, можно также на вкладке **Data** (рис. 9.4).

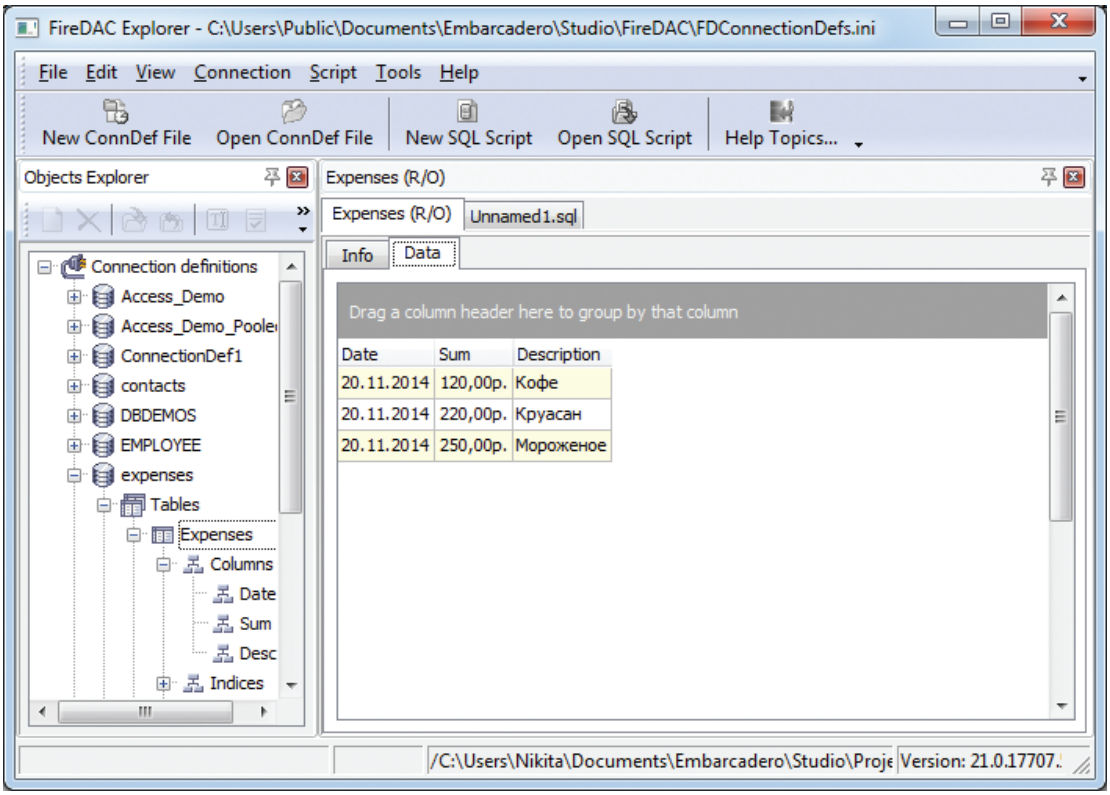


Рисунок 9.4. Результат добавления записей в базу данных

Форма приложения

В качестве основы для приложения работы с базой данных **Расходы** будем использовать **Header/Footer with Navigation** шаблон. На главной (стартовой) странице будет отображаться список расходов (рис. 9.5), вторая страница, переход к которой выполняется по нажатию кнопки **Добавить**, будет использоваться для добавления информации в базу данных (рис. 9.6).

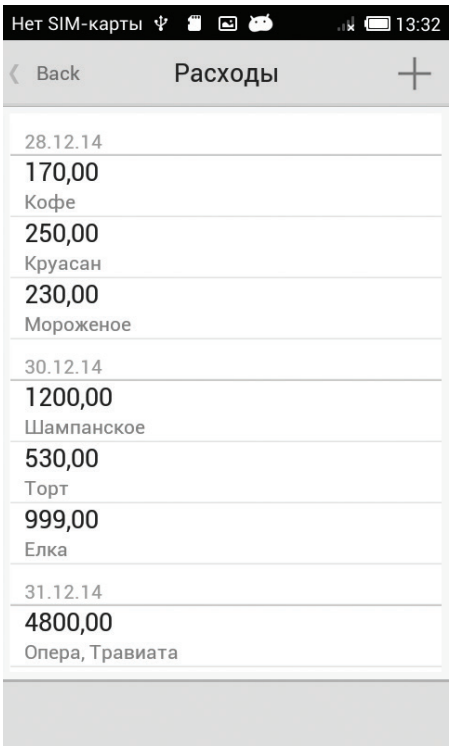


Рисунок 9.5. Главная страница приложения

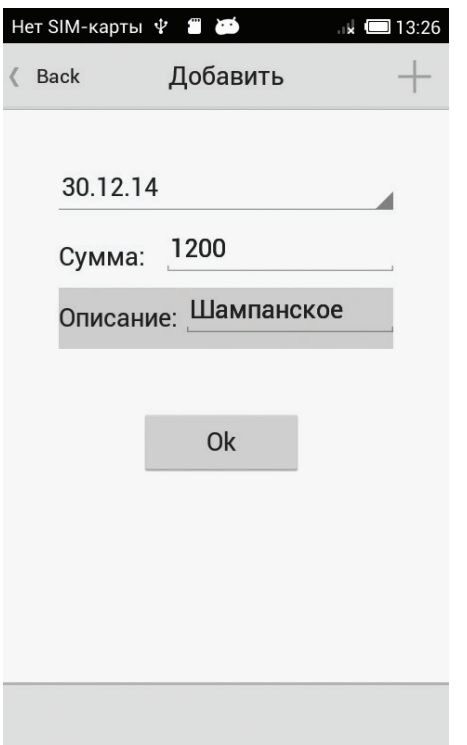


Рисунок 9.6. Страница **Добавить**

Доступ к данным

Доступ к данным осуществляется при помощи компонентов FDConnection и FDTable.

Компонент FDConnection (его свойства приведены в табл. 9.2) обеспечивает соединение с базой данных. Компонент FDTable используется для получения данных из таблицы. Он, в зависимости от настройки, может содержать данные всей таблицы или только выборку.

Таблица 9.2. Свойства компонента *FDConnection*

Свойство	Описание
DriverName	Имя драйвера, обеспечивающего взаимодействие с базой данных
LoginPrompt	Признак необходимости в момент подключения к базе данных запросить у пользователя имя и пароль. Если значение свойства равно False, то при подключении к базе данных имя и пароль не запрашиваются (окно **Login** в момент подключения к базе данных не отображается)
Connected	Признак того, что соединение установлено

Чтобы выполнить настройку компонента FDConnection, надо сделать щелчок на находящейся в нижней части окна **Object Explorer** ссылке **Connection Editor** (ссылка отображается, если в конструкторе формы выбран компонент FDConnection). В появившемся окне **FireDAC Connection Editor** (рис. 9.7) в списке DriverID надо выбрать имя сервера базы данных и в поле **Database** ввести имя файла базы данных. После этого можно сделать щелчок на кнопке **Test** и убедиться, что параметры соединения с базой данных установлены правильно. Следует обратить внимание, если в поле **Database** ввести имя несуществующего файла, то в результате щелчка на кнопке **Test** файл будет создан. После этого надо активизировать соединение – присвоить свойству LoginPrompt значение False, а свойству Connected - значение True. Значения свойств компонента FDConnection1 после настройки соединения с базой данных **Расходы** приведены в табл. 9.3.

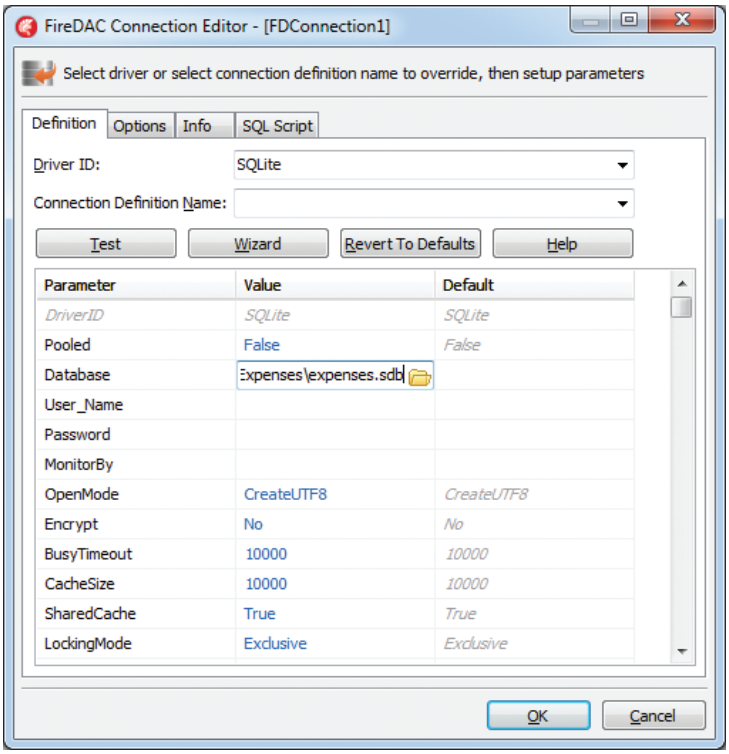


Рисунок 9.7. Настройка соединения с базой данных

Таблица 9.3. Значения свойств компонента `FDConnection1`

Свойство	Значение
DriverName	SQLite
LoginPrompt	False
Connected	True

После того как будет настроен компонент FDConnection, на форму следует поместить компонент FDTable и выполнить его настройку.

Компонент FDTable (таблица данных) хранит данные, полученные из таблицы базы данных. Свойства компонента FDTable приведены в табл. 9.4.

Таблица 9.4. Свойства компонента `FDTable`

Свойство	Описание
Connection	Ссылка на компонент FDConnection, обеспечивающий соединение с источником данных
TableName	Имя таблицы данных, данные из которой загружаются в компонент FDTable
Filter	Фильтр – условие отбора записей
Filtered	Признак использования фильтра
Activate	Открывает или делает недоступным набор данных

Настройка компонента FDTable выполняется обычным образом, путем присвоения требуемых значений свойствам компонента (табл. 9.5). Следует обратить внимание, значение свойств устанавливать в том порядке, в котором они перечислены в таблице.

Таблица 9.5. Значения свойств компонента FDTable

Свойство	Значение
Connection	FDConnection1
TableName	expenses
Active	True

Отображение данных

Для отображения информации, находящейся в таблице expenses базы данных **Расходы**, будем использовать компонент ListBox.

Значение свойств компонента ListBox. Они определяют его положение на форме и вид информации, отображаемой в списке, приведены в табл. 9.6.

Таблица 9.6. Значения свойств компонента ListBox

Свойство	Описание
Align	Client
Margins.Top	10
Margins.Bottom	10
Margins.Left	5
Margins.Right	5
GroupingKind	Grouped
DefaultItemStyles.GroupHeaderStyle	listboxgroupheader
DefaultItemStyles.ItemStyle	listboxbottomitemdedail

Связывание данных

Для того чтобы данные, находящиеся в компоненте FDTable, появились в списке ListBox, необходимо связать источник данных (компонент FDTable) с компонентом отображения данных (компонент ListBox).

Чтобы начать процесс связывания, надо в окне дизайнера формы выбрать компонент FDTable и сделать щелчок на ссылке **Bind Visually** в нижней части окна **Object Inspector**.

В появившемся окне **LiveBinding Designer** (рис. 9.8) отображается графическая модель компонентов приложения.

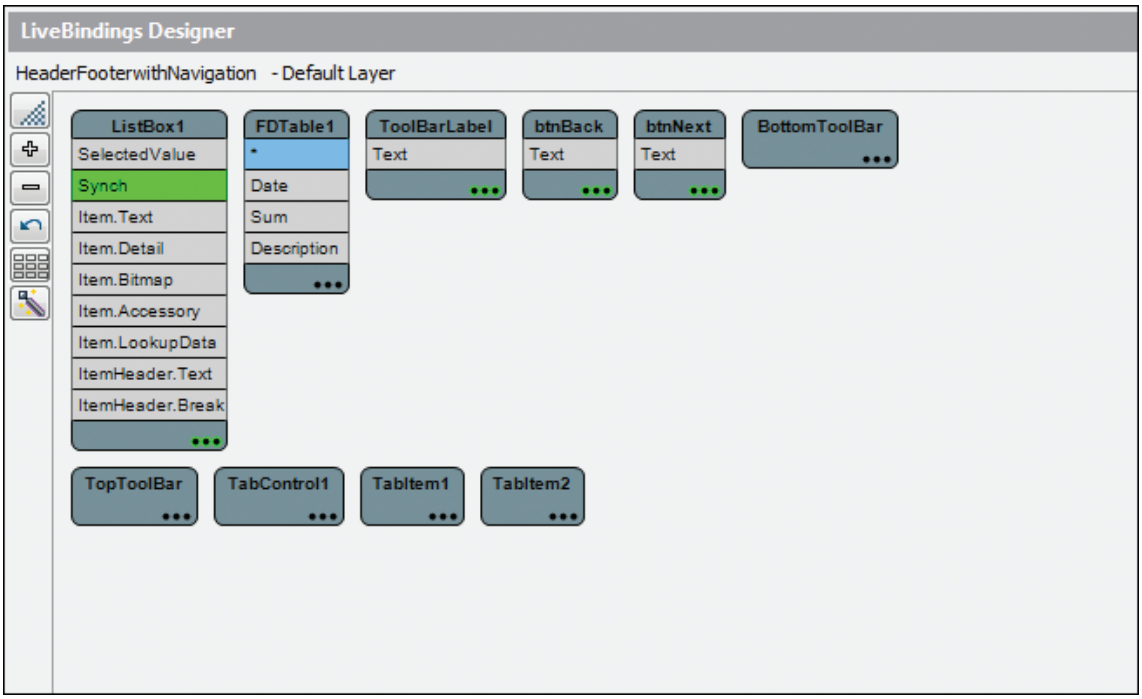


Рисунок 9.8. Окно **LiveBinding Designer**

Прямоугольники в окне **LiveBinding Designer** изображают компоненты. В верхней части прямоугольника указано имя компонента, далее следуют свойства, которые могут выступать в качестве источника или приемника данных. Обратите внимание, в прямоугольнике FDTable1 перечислены имена полей таблицы expenses (вспомните, при настройке компонента FDTable1 свойству TableName было присвоено значение expenses). В прямоугольнике ListBox1следует обратить внимание на элементы item. Text, item.Detal и itemHeader.Text. Если вспомнить, что компоент ListBox представляет собой список, состоящий из элементов (item – элемент), то очевидно, что свойство item. Text определяет текст, который отображается в элементе (строке) списка, свойство item. Detal – дополнительную информацию, а свойство itemHeader.Text – текст заголовка элемента. Таким образом, для того, чтобы в элементе списка ListBox1 отображалось содержимое записи таблицы, находящейся в компоненте FDTable1, надо Sum связать с item.Text, Description – с item.Detail, Date – с itemHeader.Text.

Чтобы связать поле Sum компонента TDTable1 с полем с item.Text компонента ListBox1 надо установить указатель мыши на изображение прямоугольника Sum, нажать левую кнопку мыши и, удерживая ее нажатой, установить указатель мыши на прямоугольник item.Text, отпустить кнопку мыши. В результате выполнения описанных действий связь будет установлена, а между Sum и item.Text появится стрелка, изображающая эту связь. Кроме того, в форму будут добавлены компоненты BindSourceDB и BindingList. Аналогичным образом выполняется связывание Description с item.Detail и Date с itemHeader.Text.

Вид окна **LiveBindings Designer** после выполнения связывания приведен на рис. 9.9, а вид формы (страницы TabItem1) – на рис. 9.10.

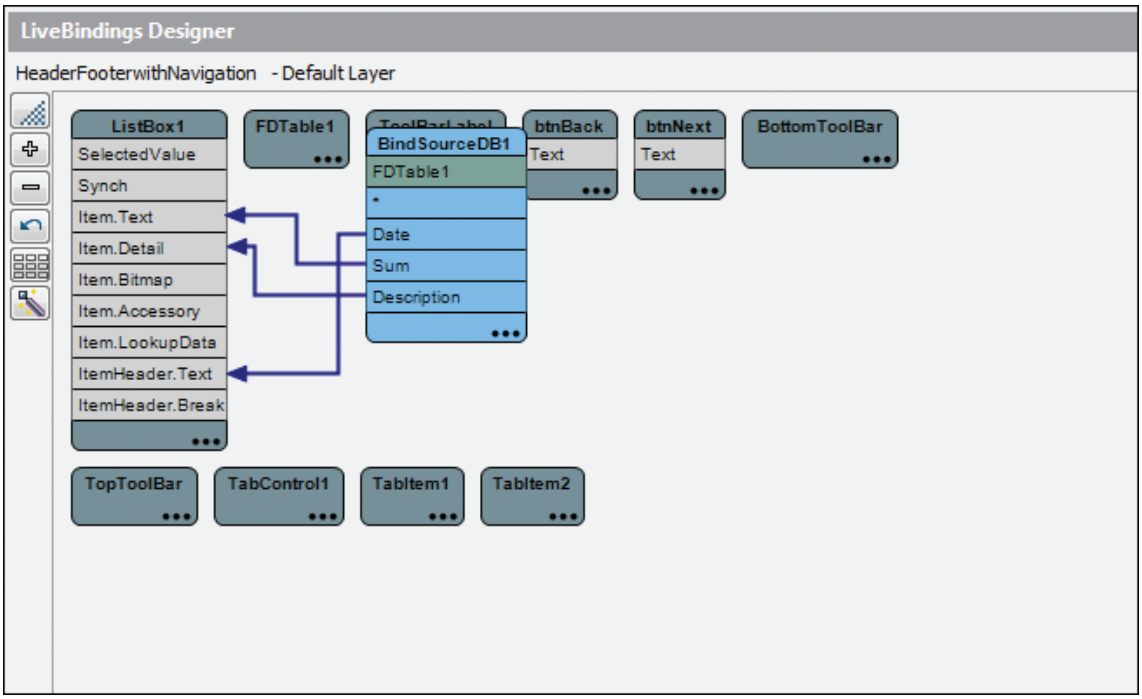


Рисунок 9.9. Результат связывания

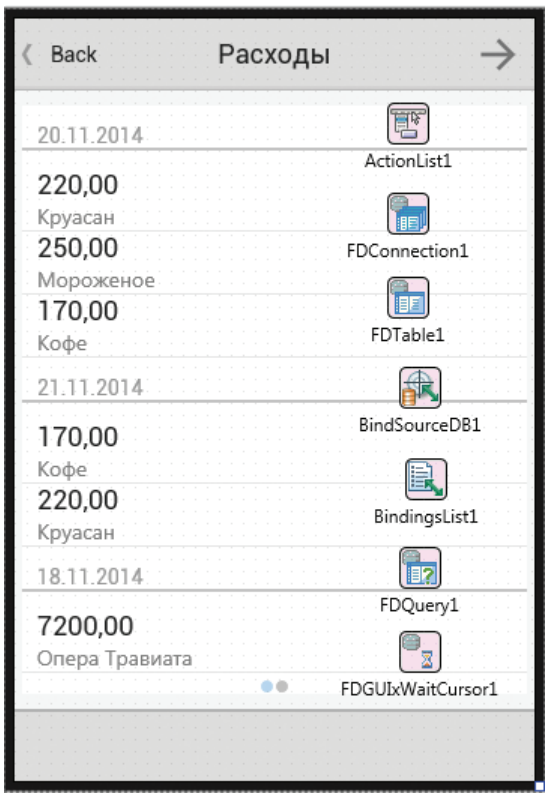


Рисунок 9.10. Вид страницы TabItem1 после связывания компонентов FDTable и ListBox

ВНИМАНИЕ!

В форму программы работы с базой данных необходимо добавить компонент FDGUIxWaitCursor (он находится на вкладке FireDAC UI) и компонент FDPhysSQLiteDriverLink (он находится на вкладке FireDAC Links).

Пробный запуск программы

При попытке запустить программу, например, на Android устройстве, на экране появиться сообщение об ошибке (рис. 9.11).

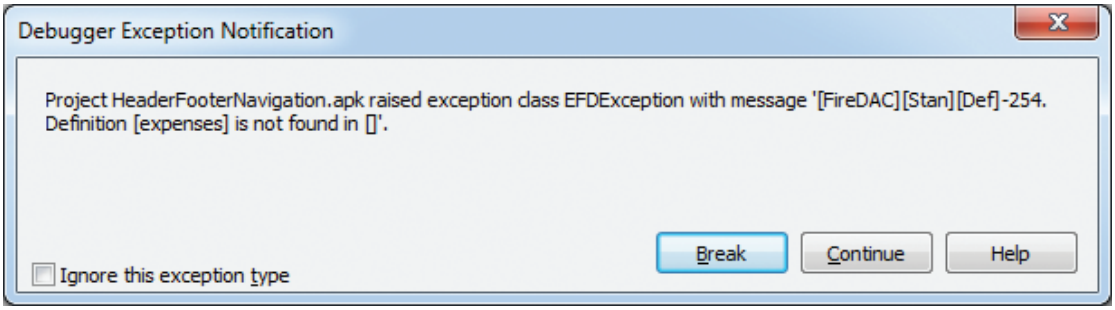
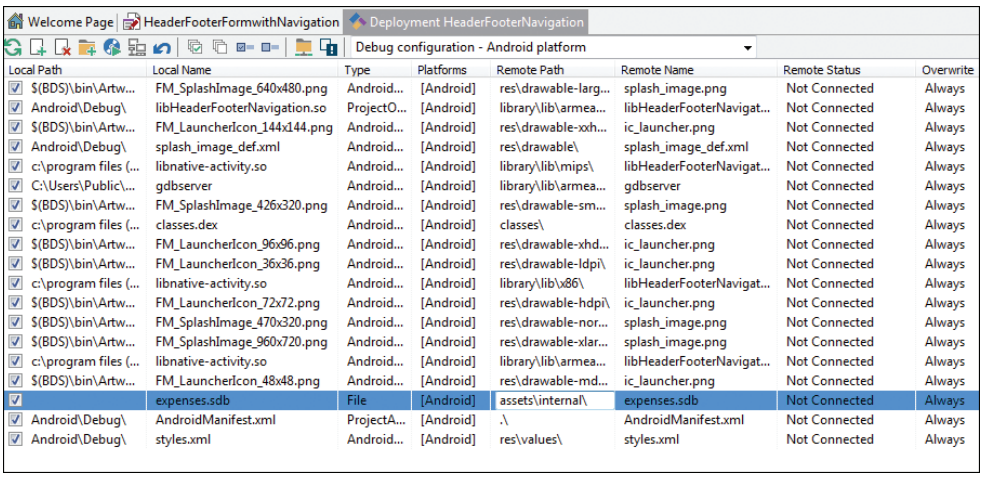


Рисунок 9.11. Сообщение о невозможности

Причина ошибки в том, что база данных находится на компьютере разработчика, а не на устройстве. Чтобы база данных появилась на устройстве, ее надо включить в пакет приложения: открыть окно **Deployment** (в меню **Project** выбрать команду **Deployment**), в появившемся окне нажать кнопку **Add Files** и указать файл базы данных. После того как файл базы данных будет добавлен в список файлов, образующих пакет, надо в поле Remote Path строки ввести путь к файлу базы данных: assets\internal\ (рис. 9.12).



Local Path	Local Name	Type	Platforms	Remote Path	Remote Name	Remote Status	Overwrite
$(BDS)\bin\Artw...	FM_SplashImage_640x480.png	Android...	[Android]	res\drawable-larg...	splash_image.png	Not Connected	Always
Android\Debug\	libHeaderFooterNavigation.so	ProjectO...	[Android]	library\lib\armea...	libHeaderFooterNavigat...	Not Connected	Always
$(BDS)\bin\Artw...	FM_LauncherIcon_144x144.png	Android...	[Android]	res\drawable-xxh...	ic_launcher.png	Not Connected	Always
Android\Debug\	splash_image_def.xml	Android...	[Android]	res\drawable\	splash_image_def.xml	Not Connected	Always
c:\program files (...	libnative-activity.so	Android...	[Android]	library\lib\mips\	libHeaderFooterNavigat...	Not Connected	Always
C:\Users\Public\...	gdbserver	Android...	[Android]	library\lib\armea...	gdbserver	Not Connected	Always
$(BDS)\bin\Artw...	FM_SplashImage_426x320.png	Android...	[Android]	res\drawable-sm...	splash_image.png	Not Connected	Always
c:\program files (...	classes.dex	Android...	[Android]	classes\	classes.dex	Not Connected	Always
$(BDS)\bin\Artw...	FM_LauncherIcon_96x96.png	Android...	[Android]	res\drawable-xhd...	ic_launcher.png	Not Connected	Always
$(BDS)\bin\Artw...	FM_LauncherIcon_36x36.png	Android...	[Android]	res\drawable-ldpi\	ic_launcher.png	Not Connected	Always
c:\program files (...	libnative-activity.so	Android...	[Android]	library\lib\x86\	libHeaderFooterNavigat...	Not Connected	Always
$(BDS)\bin\Artw...	FM_LauncherIcon_72x72.png	Android...	[Android]	res\drawable-hdpi\	ic_launcher.png	Not Connected	Always
$(BDS)\bin\Artw...	FM_SplashImage_470x320.png	Android...	[Android]	res\drawable-nor...	splash_image.png	Not Connected	Always
$(BDS)\bin\Artw...	FM_SplashImage_960x720.png	Android...	[Android]	res\drawable-xlar...	splash_image.png	Not Connected	Always
c:\program files (...	libnative-activity.so	Android...	[Android]	library\lib\armea...	libHeaderFooterNavigat...	Not Connected	Always
$(BDS)\bin\Artw...	FM_LauncherIcon_48x48.png	Android...	[Android]	res\drawable-md...	ic_launcher.png	Not Connected	Always
	expenses.sdb	File	[Android]	assets\internal\	expenses.sdb	Not Connected	Always
Android\Debug\	AndroidManifest.xml	ProjectA...	[Android]	.\	AndroidManifest.xml	Not Connected	Always
Android\Debug\	styles.xml	Android...	[Android]	res\values\	styles.xml	Not Connected	Always

Рисунок 9.12. Добавление файла базы данных в пакет приложения

Информацию о месте нахождения файла базы данных содержит параметр Database компонента FDConnection. Во время разработки приложения параметр Database

ссылается на файл, находящийся на компьютере разработчика, а во время работы программы он должен ссылаться на файл, находящийся на устройстве. Задачу динамической настройки компонента FDConnection можно возложить на процедуру обработки события BeforeConnect самого компонента (листинг 9.1).

Листинг 9.1. Процедура обработки события BeforeConnect компонента FDConnection

```
procedure THeaderFooterwithNavigation.FDConnection1BeforeConnect(
  Sender: TObject);
begin
   {$IF DEFINED(IOS) or DEFINED(ANDROID)}
       FDConnection1.Params.Values['Database']:=
            Tpath.Combine(TPath.GetDocumentsPath, 'expenses.sdb');
   {$ENDIF}
end;
```

Если предполагается, что изначально база данных должна быть пустой (для программы Расходы — это действительно так), то файл базы данных можно в пакет не включать, а создать базу данных при первом запуске программы на устройстве. Приведенный в листинге 9.2 код показывает, как это можно сделать. Процедура обработки события BeforeConnect компонента FDConnection проверяет, есть ли в рабочем каталоге программы файл базы данных и, если файла нет, создает его. Процедура обработки события AfterConnect направляет серверу SQL команду создания таблицы в базе данных. Так как в команде указано условие создания таблицы, то таблица создается только в том случае, если таблицы с именем expenses в базе данных нет, т.е. при первом запуске программы.

Листинг 9.2. Создание базы данных SQLite в коде

```
// перед тем, как установить соединение
procedure THeaderFooterwithNavigation.FDConnection1BeforeConnect(
  Sender: TObject);
var
  database: string;
  f: TextFile;
begin
  database := Tpath.Combine(TPath.GetDocumentsPath, 'expenses.sdb');
  if ( not System.SysUtils.FileExists(database) ) then
        begin
          AssignFile( f,database);
          Rewrite(f);
          CloseFile(f);
        end;
   FDConnection1.Params.Values['Database']:= database;
end;
// после того, как соединение установлено
procedure THeaderFooterwithNavigation.FDConnection1AfterConnect(
  Sender: TObject);
var
  SqlText: string;
begin
      // создать таблицу
      SqlText := 'CREATE TABLE IF NOT EXISTS Expenses (Date DATETIME NOT NULL, Sum MONEY
NOT NULL, Description TEXT NOT NULL)';
```

```
      FDQuery1.SQL.Text := SqlText;
      FDQuery1.ExecSQL;
      FDTable1.Active := True;
end;
```

Добавление записей в базу данных

Чтобы добавить запись в базу данных, надо направить серверу SQL команду INSERT. Сделать это можно при помощи компонента FDQuery (свойства компонента приведены в табл. 9.7).

Таблица 9.7. Свойства компонента FDTable

Свойство	Описание
Connection	Ссылка на компонент FDConnection, обеспечивающий соединение с источником данных
SQL	SQL команда, которая направляется серверу
Activate	Активизирует процесс выполнения команды

Так как команда, обеспечивающая добавление записи в таблицу базы данных, формируется во время работы программы, то настройка компонента FDQuery сводится к записи в свойство Connection имени компонента, обеспечивающего соединение с базой данных.

Рассматриваемое приложение работы с базой данных двухстраничное: на главной странице отображается список расходов, вторая страница, переход к которой выполняется по нажатию кнопки **Добавить**, используется для добавления информации в базу данных. Поэтому именно на вторую страницу надо поместить компоненты, предназначенные для ввода информации.

По умолчанию в окне конструктора отображается первая страница компонента TabControl (т.к. в качестве базы для приложения работы с базой данных Расходы был выбран шаблон **Header/Footer with Navigation**, то компонент TabControl был добавлен в форму автоматически). Чтобы начать работу со второй страницей, надо в окне **Structure** выбрать компонент TabControl и сделать щелчок на находящейся в нижней части окна **Object Inspector** ссылке **NextTab** (аналогичным образом можно вернуться к главной странице, но для этого надо будет сделать щелчок на ссылке **Previous Tab**).

Чтобы добавить в базу данных запись, надо направить серверу команду INSERT. В команде INSERT должна быть указана дата, сумма и описание, на что потрачены деньги. Для получения этой информации от пользователя будем использовать компонент DateEdit (он находится на вкладке Additional) и два компонента Edit. В качестве контейнера для компонентов взаимодействия с пользователем будем использовать компонент ListBox.

Ниже приведена последовательность шагов для создания страницы добавления записи в базу данных Расходы.

1. Поместить на страницу TabItem2 компонент ListBox.
2. Свойствам Height, Align, StyleLookup и Margins.Top присвоить соответственно значения: 160, Top, transparentlistboxstyle и 20.

3. Добавить в коллекцию Items три элемента (из контекстного меню компонента ListBox выбрать команду **Add TListBoxItem**).
4. У каждого элемента списка очистить свойство Text.
5. Поместить в ListBoxItem1 компонент DateEdit и его свойству Align присвоить значение Client.
6. Поместить в ListBox2 компонент Label, а его свойствам Text, Autosize и Align присвоить соответственно значения Сумма, True и Left.
7. Поместить в ListBox2 компонент Edit, свойствам Align и Margins.Left присвоить соответственно значения Client и 10.
8. Поместить в ListBox3 компонент Label, а его свойствам Text, Autosize и Align присвоить соответственно значения Описание, True и Left.
9. Поместить в ListBox3 компонент Edit, свойствам Align и Margins.Left присвоить соответственно значения Client и 10.
10. Поместить на страницу компонент Button, свойству Text присвоить значение Добавить.

После выполнения описанных выше шагов страница TabItem2 должна выглядеть так, как показано на рис. 9.13.

Обратите внимание, чтобы при появлении страницы в поле компонента DataEdit отображалась текущая дата, в конструктор формы надо добавить инструкцию:

```
DataEdi1 := System.SysUtils.Date;
```

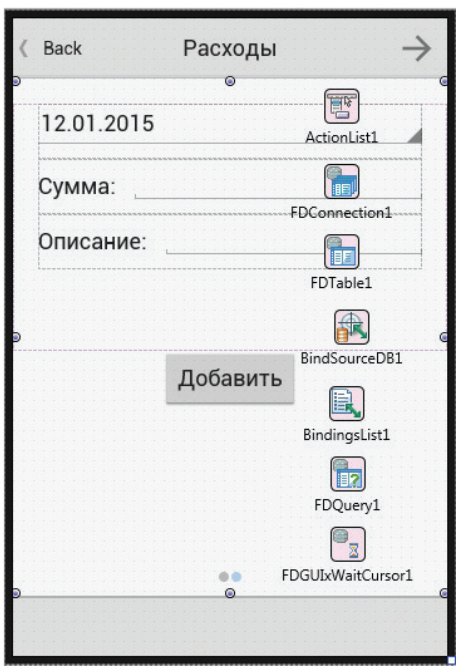


Рисунок 9.13. Страница TabItem2

Процедура обработки события Click на кнопке **Добавить** приведена в листинге 9.3. Обратите внимание, чтобы при появлении страницы

Листинг 9.3. Процедура обработки события Click на кнопке Добавить

```
procedure TForm1.Button1Click(Sender: TObject);
var
    SqlText: string;
    st: string;
begin

    // System.SysUtils.Date - текущая дата

    if (Edit1.Text.Trim.Length = 0)  Or (Edit2.Text.Trim.Length = 0)
       then exit;

    DateTimeToString(st, 'YYYY-MM-DD', DateEdit1.Date);

    SqlText := 'INSERT INTO Expenses Values("' +
           st + '", "' +
           Edit1.Text + '", "' +
           Edit2.Text + '")';

    FDQuery1.SQL.Text := SqlText;
    FDQuery1.ExecSQL;

    Edit1.Text := '';
    Edit2.Text := '';

    FDTable1.Refresh();
    LinkFillControlToField1.BindList.FillList;
end;
```

Глава 10. Работа с сенсорами

В этой главе рассказывается, как работать с сенсорами – датчиками, позволяющими программе получить информацию об устройстве, например, о его положении в пространстве (географические координаты) и состоянии (движение или покой).

Компоненты доступа к сенсорам

Компоненты, обеспечивающие доступ к сенсорам устройства, находятся на вкладке Sensors палитры компонентов (рис. 10.1).

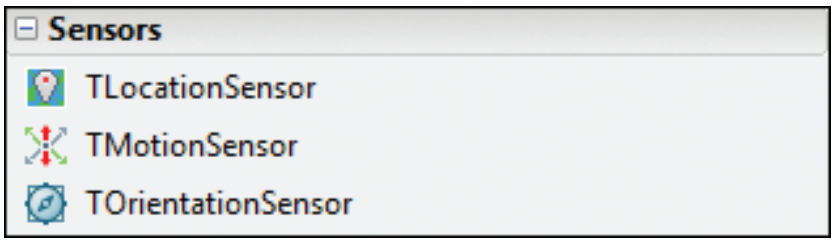


Рисунок 10.1. Компоненты доступа к сенсорам устройства

Компонент `LocationSensor` позволяет получить географические координаты устройства, компонент `MotionSensor` – информацию о движении устройства, компонент `OrientationSensor` – об ориентации в пространстве.

Датчик положения

При помощи компонента `LocationSensor` программа может получить информацию о положении устройства в пространстве – географические координаты: широту (Latitude) и долготу (Longitude). Информацию о положении устройства компонент получает от GPS приемника устройства, от оператора сотовой связи или определяет по координатам Wi-Fi точек доступа. В любом случае устройство, на котором работает приложение, должно разрешать программе получать информацию о местоположении.

Процесс работы с сенсором положения рассмотрим на примере программы (рис. 10.2), позволяющей получить текущие координаты устройства и загрузить с сайта Google карту местности, в которой находится устройство.

Форма приложения приведена на рис. 10.3.

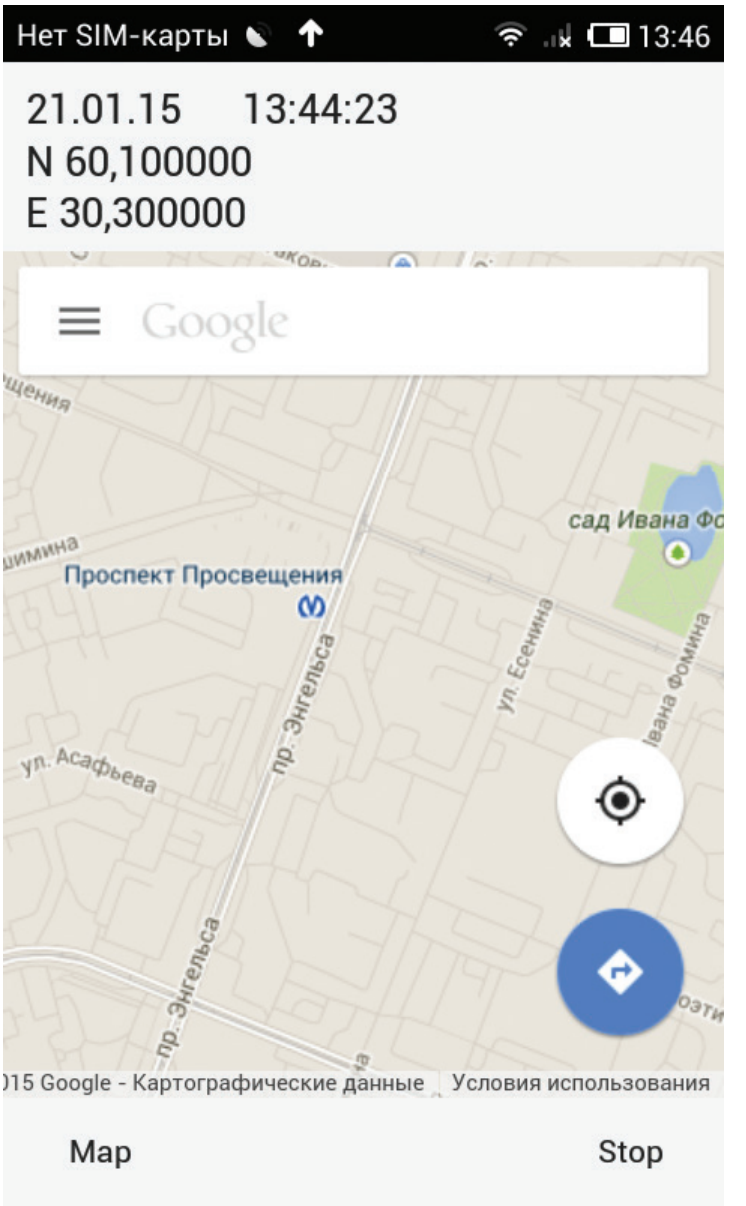


Рисунок 10.2. Окно программы

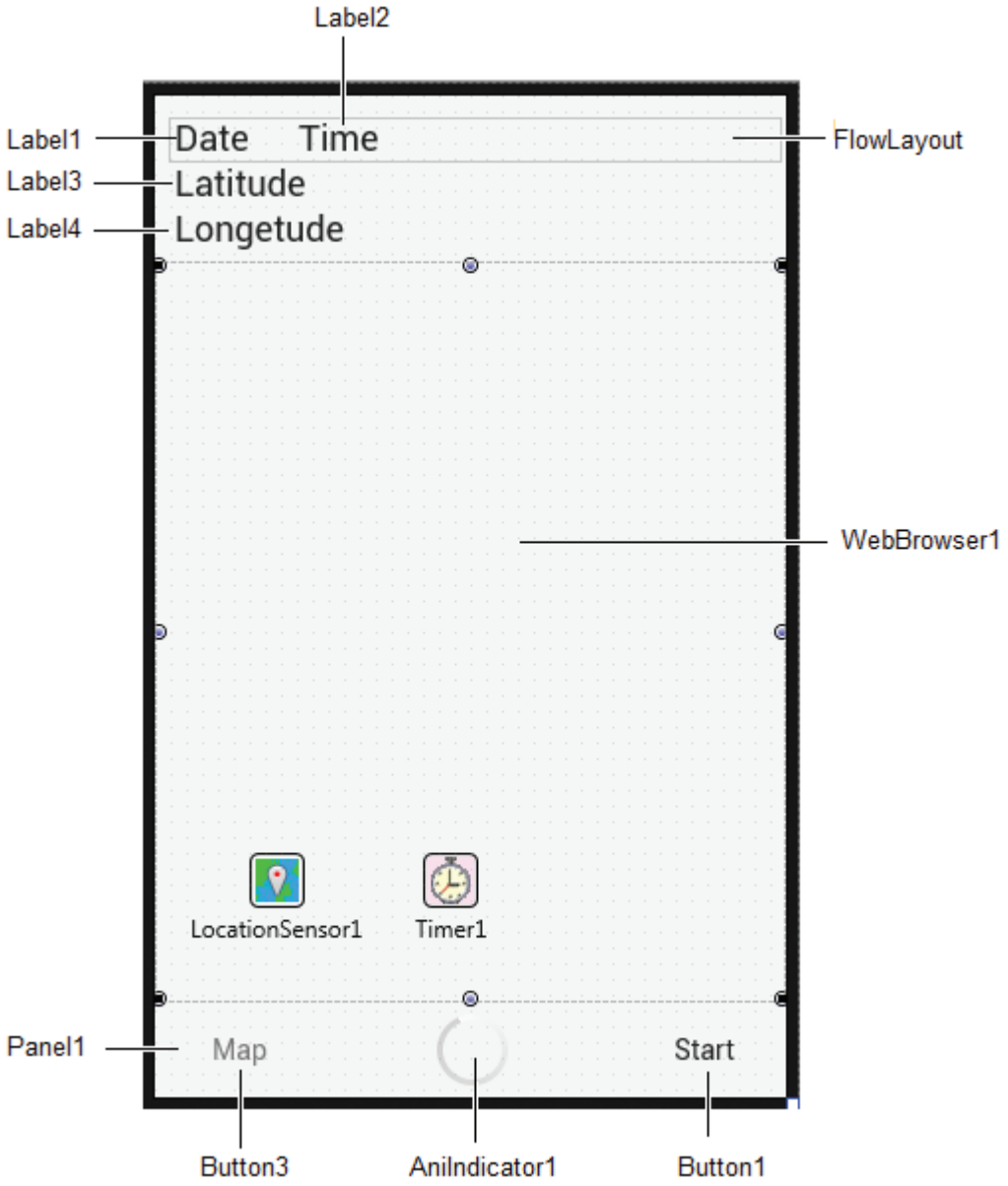


Рисунок 10.3. Форма программы

Компоненты `Label1` и `Label2`, находящиеся внутри контейнера `FlowLayout1`, используются для отображения текущей даты и времени. В поле компонентов `Labe3` и `Label4`, отображаются широта и долгота точки, в которой находится устройство. В нижней части формы, на панели (компонент `Panel`) находятся командные кнопки включения сенсора (`Button1`) и активизации процесса загрузки карты (`Button2`). Там же находится индикатор процесса загрузки карты. Центральную часть формы занимает компонент `WebBrowser`. Значения свойств компонентов, определяющих их положение на форме, приведены в табл. 10.1. Чтобы форма выглядела так, как показано на рисунке, компоненты надо добавлять на форму в том порядке, в котором они следуют в таблице.

Таймер в рассматриваемой программе используется для обновления индикатора времени: функция обработки сигала от таймера обновляет содержимое поля `Label2`. Индикатор `AniIndicator` отображается только в процессе загрузки карты с сервера Google.

Таблица 10.1. Значения свойств компонентов

Компонент	Свойство	Значение
FlowLayout1	Align	Top
	Margins.Left	10
Label1	Align	Left
Label2	Align	Left
	Margins.Left	10
Label3	Align	Top
	Margins.Left	10
Label4	Align	Top
	Margins.Left	10
Panel1	Align	Bottom
Button1	StyleLookup	speedbuttonstyle
	Align	Right
	Margins.Right	5
	Margins.Bottom	5
Button2	StyleLookup	speedbuttonstyle
	Align	Left
	Margins.Leftt	5
	Margins.Bottom	5
AniIndicator1	Align	Center
WebBrowser1	Align	Client

Чтобы получить координаты устройства, надо включить сенсор положения – присвоить свойству `Active` компонента `LocationSensor` значение `True`. Следует обратить внимание, сенсор не может моментально определить местоположение устройства, для определения координат необходимо время, иногда, значительное. Когда сенсор определит положение устройства или, если координаты уже известны, обнаружит изменение координат, компонент `LocationSensor` генерирует событие `LocationChanged` (положение изменилось). Это событие и должна обрабатывать программа, чтобы, например, отобразить координаты.

Текст процедур обработки событий компонентов приведен в листинге.

Листинг 10.1. Работа с датчиком положения

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

```
      Label1.Text:= DateToStr(System.SysUtils.Date);
      Label2.Text := TimeToStr(System.SysUtils.Time);
end;
// нажатие Button1
// Включить/выключить сенсор
procedure TForm1.Button1Click(Sender: TObject);
var
  button: TButton;
begin
  button := Sender as TButton;
  if button.Tag = 0 then
    begin
      // команда Старт
      button.Tag := 1;
      button.Text := 'Stop';
      Label3.Text := '';
      Label4.Text := '';
      Label5.Text := '';
      // включить сенсор
      LocationSensor1.Active := True;
    end
  else
    begin
      // команда Стоп
      button.Tag := 0;
      button.Text := 'Start';
      // выключить сенсор
      LocationSensor1.Active := False;
    end;
end;
// изменилось положение (координаты) устройства
procedure TForm1.LocationSensor1LocationChanged(Sender: TObject; const OldLocation, New-
Location: TLocationCoord2D);
var
    latitude,longitude: double;
    stLat, stLong: string;
begin
      Timer1.Enabled := false;
      Button2.Enabled := true;
       // показать координаты
       latitude := NewLocation.Latitude;
       longitude := NewLocation.Longitude;
       if (latitude > 0) then
          stLat := 'N '    // N - Nord, северная широта
       else
          stLat := 'S ';   // S - South, южная широта
       if (longitude > 0) then
          stLong := 'E '    // E - East, восточная долгота
       else
          stLong := 'W ';   // W - West, западная долгота
       Label3.Text := stLat +  FloatToStrF(latitude, ffFixed,3,6);
       Label4.Text :=  stLong   + FloatToStrF(longitude, ffFixed,3,6);
end;
// показать точку на карте
procedure TForm1.Button2Click(Sender: TObject);
```

```
const
  LGoogleMapsURL: String = 'https://maps.google.com/maps?q=%s,%s';
  GoogleMapsURL: String = 'https://www.google.com/maps/@%s,%s,16z';
var
    ENUSLat, ENUSLong: String;
    url: string;
begin
       // показать точку на карте
       ENUSLat := FloatToStrF(p0.Y,ffGeneral, 5, 6, TFormatSettings.Create('en-US'));
       ENUSLong := FloatToStrF(p0.X,  ffGeneral, 5, 6, TFormatSettings.Create('en-US'));
       url := Format(GoogleMapsURL, [ENUSLat, ENUSLong]);
       //Label6.Text := url;
       AniIndicator1.Enabled := True;
       AniIndicator1.Visible := True;
       // загрузить карту
       WebBrowser1.Navigate(url);
end;
// начало загрузки страницы
procedure TForm1.WebBrowser1DidStartLoad(ASender: TObject);
begin
      AniIndicator1.Enabled := True;
      AniIndicator1.Visible := True;
end;
// страница загружена
procedure TForm1.WebBrowser1DidFinishLoad(ASender: TObject);
begin
      AniIndicator1.Enabled := False;
      AniIndicator1.Visible := False;
end;
// сигнал от таймера, обновить индикатор
procedure TForm1.Timer1Timer(Sender: TObject);
begin
      Label2.Text := TimeToStr(System.SysUtils.Time);
end;
// завершение работы приложения
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
         LocationSensor1.Active := False;
end;
```

Датчик движения

Компонент `MotionSensor` – датчик движения, который позволяет получить информацию об изменении ориентации и движении устройства в пространстве в трех плоскостях, вдоль привязанных к устройству осей X, Y и Z.

Ось X направлена слева направо вдоль экрана устройства, ось Y - снизу вверх, ось Z направлена перпендикулярно лицевой стороне устройства.

Доступные во время работы приложения свойства `AccelerationX`, `AccelerationY` и `AccelerationZ` содержат значения измеряемого в Галах текущего ускорения вдоль соответствующих осей. Один Гал соответствует ускорению 9.8 м/сек^2.

Следующая программа, ее форма приведена на рис. 10.4 позволяет поэкспериментировать с датчиком движения (акселерометром), понять, как с ним работать.

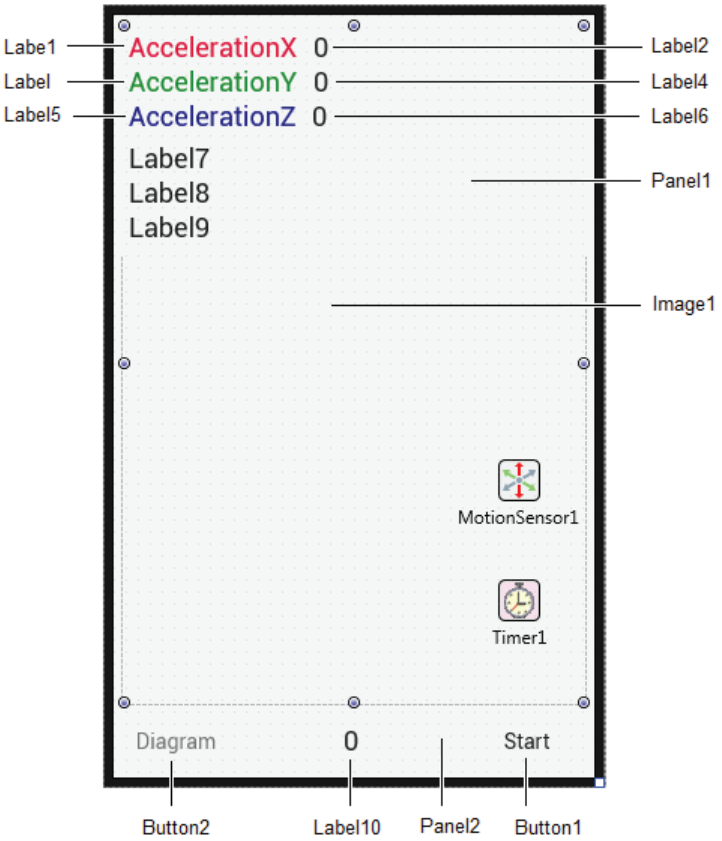


Рисунок 10.4. Форма приложения **Датчик движения**

Значения свойств компонентов приведены в табл. 10.2. Обратите внимание, компоненты `Label1` и `Label2`, `Label3` и `Label4`, а также компоненты `Label5` и `Label6` размещены на подложке – компонентах `FlowLayout`. Чтобы форма выглядела так как показано на рисунке, компоненты нужно добавлять на форму в том порядке, в котором они следуют в таблице.

Таблица 10.2. Значения свойств компонентов

Компонент	Свойство	Значение
Panel1	Align	Top
FlowLayout1	Align	Top
	Margins.Left	10
Label1	Align	Left
	TextSettings.FontColor	Crimson
Label2	Align	Left
	Margins.Left	10
FlowLayout2	Align	Top
	Margins.Left	10
Label3	Align	Left
	TextSettings.FontColor	Forestgreen

Компонент	Свойство	Значение
Label4	Align	Left
	Margins.Left	10
FlowLayout3	Align	Top
	Margins.Left	10
Label5	Align	Left
	TextSettings.FontColor	Darkblue
Label6	Align	Left
	Margins.Left	10
Pane2	Align	Bottom
Button1	StyleLookup	speedbuttonstyle
	Text	Start
	Align	Right
	Margins.Right	5
	Margins.Bottom	5
Button2	StyleLookup	speedbuttonstyle
	Text	Diagram
	Align	Left
	Margins.Leftt	5
	Margins.Bottom	5
Label10	Align	Center
Image1	Align	Contents

В окне приложения, после активизации акселерометра нажатием кнопки **Start**, отображаются текущие значения ускорений (рис. 10.5). Частота обновления значений ускорений в окне программы определяется частотой прерываний от таймера (событий Timer). Процедура обработки события Timer отображает значения ускорений и записывает их в массивы для дальнейшей обработки и отображения в виде графиков. После остановки акселерометра кнопкой **Stop** и нажатия кнопки Diagram в окне программы отображаются графики изменения ускорений (рис. 10.6). Вид линий позволяет понять, как меняются значения свойств AccelerationX, AccelerationY и AccelerationZ при перемещении устройства в пространстве.

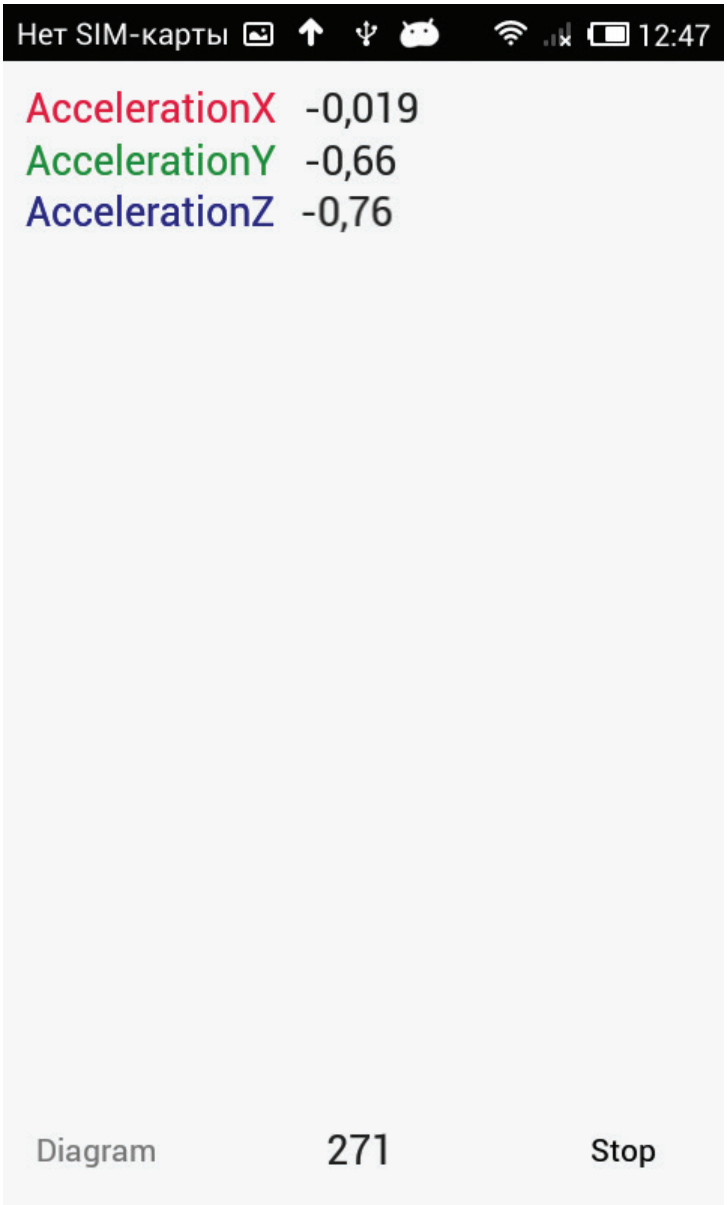


Рисунок 10.5. Запись значений ускорений

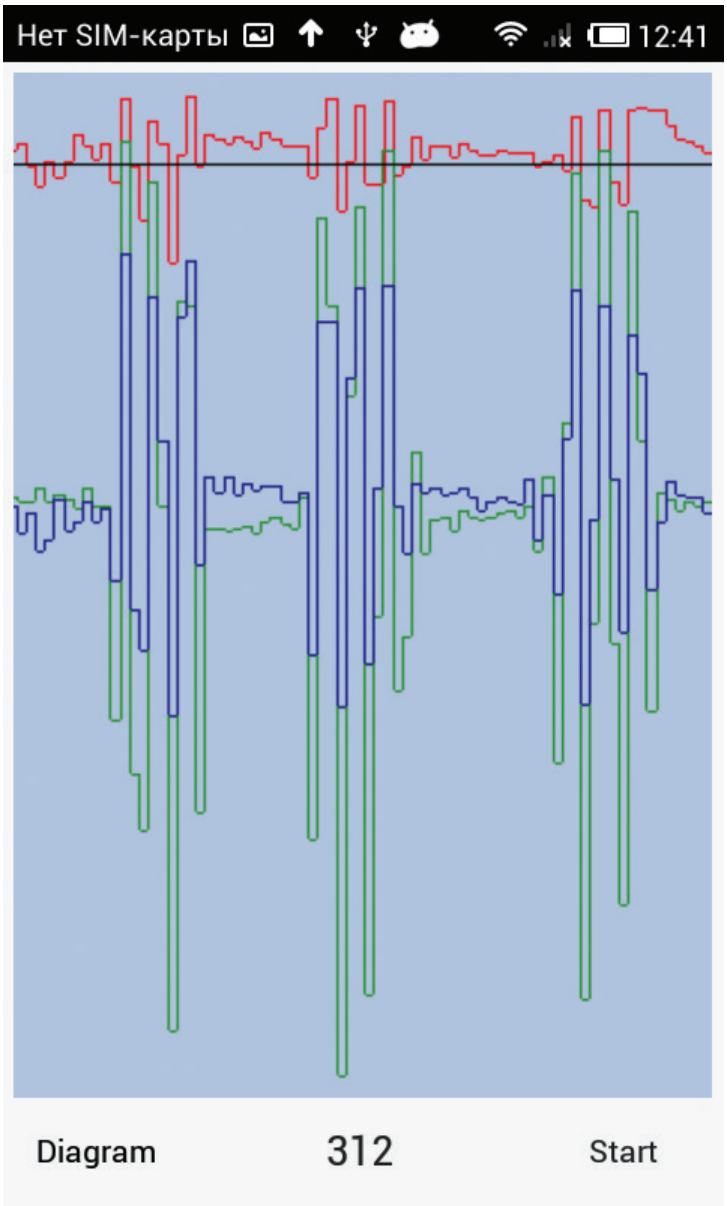


Рисунок 10.6. Графики изменения ускорений

Код программы **Датчик движения** приведен в листинге 10.2.

Листинг 10.2. Датчик движения

```
const
  HB = 500; // количество циклов записи информации с датчика
var
  // диапазон изменения значений
  minX, maxX: single;
  minY, maxY: single;
  minZ, maxZ: single;
```

```
  // ряд значений, полученных от сенсора
  arX: array[1..HB] of single;
  arY: array[1..HB] of single;
  arZ: array[1..HB] of single;
  p: integer; // указатель записи в массивы
  mode: integer; // 1 - запись данных; 0 - просмотр (анализ)
procedure TForm1.FormCreate(Sender: TObject);
begin
      label7.Text := '';
      label8.Text := '';
      label9.Text := '';
end;
 // Пуск/Стоп
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
begin
        if (Sender as TButton).Tag = 0 then
        begin
            // начать запись
            Button1.Text := 'Stop';
            Button1.Tag := 1;
            Button2.Enabled := False;
            Image1.Visible := False;
            for i:=1 to HB do
            begin
              arX[i] := 0;
              arY[i] := 0;
              arZ[i] := 0;
            end;
            p:=0;
            Panel1.Visible := True;
            Label7.Text := '';
            Label8.Text := '';
            Label9.Text := '';
            // запуск сенсора
            MotionSensor1.Active := true;
            Timer1.Enabled := True;
            mode:= 1;  // режим записи
        end
        else
          begin
            // выключить сенсор
            MotionSensor1.Active := False;
            Timer1.Enabled := False;
            Button1.Text := 'Start';
            Button1.Tag := 0;
            Button2.Enabled := True;
            // показать диапазон изменения Acceleration
            Label7.Text := Format('minX=%f maxX=%f',[minx, maxX]);
            Label8.Text := Format('minY=%f maxY=%f',[minY, maxY]);
            Label9.Text := Format('minZ=%f maxZ=%f',[minZ, maxZ]);
            mode := 0;
          end;
end;
```

```
// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  ax,ay,az: single;
begin
      ax:= MotionSensor1.Sensor.AccelerationX;
      ay:= MotionSensor1.Sensor.AccelerationY;
      az := MotionSensor1.Sensor.AccelerationZ;
      // запомнить значения для построения графика
      if p < HB then
      begin
        p := p+1;
        arX[p] := ax;
        arY[p] := ay;
        arZ[p] := az;
        Label10.Text := IntToStr(p);
      end;
      Label2.Text := FloatToStrF(ax, ffGeneral,2,9);
      Label4.Text := FloatToStrF(ay, ffGeneral,2,9);
      Label6.Text := FloatToStrF(az, ffGeneral,2,9);
      // определить max и min значения
      if ax > maxX then maxX := ax;
      if ax < minX then minX := ax;
      if ay > maxY then maxY := ay;
      if ay < minY then minY := ay;
      if az > maxZ then maxZ := az;
      if az < minZ then minZ := az;
end;
// рисует график значений массива ar
procedure TForm1.DrawAcceleration(ar  : array of single; bl, bh: single; color: TAlpha-
Color);
var
    min, max: single;
    range: single;  // диапазон изменения значений ряда
    k: single;      // коэф. масштабирования
    y0: integer;    // Y координата гор. оси
    x: integer;
    dx: integer;
    p1,p2: TPoint;
     i: integer;
begin
    if ((bh = 0) and (bl = 0))  then
    begin
      // найти min и max значения ряда
      min:= ar [1];
      max:= ar [1];
      for i:=1 to p-1 do
      begin
       if ar [i] < min then
          min := ar [i];
       if ar [i] > max then
          max := ar [i];
      end;
    end
    else begin
```

```
      min := bl;
      max := bh;
    end;
     dx := Round(Image1.Bitmap.Width / p);
     // диапазон изменения ряда данных
     if max >=0 then
      range := max - min
     else
      range := abs(min-max);
     k:= (Image1.Height-20)/range;    // 20 = отступ_сверху + отступ_снизу
     if max > 0 then
      y0 := Round(max * k)+ 10
     else
      y0:= 0;
    x:=0;
    p1.X := 0;
    p1.Y := y0 - Round(ar [1]*k);
    // строим график
    Image1.Bitmap.Canvas.BeginScene;
    Image1.Bitmap.Canvas.Stroke.Color := color;
      for i:= 2 to p-1 do
      begin
          x:= x+ dx;
          p2.X := x;
          p2.Y := y0 - Round(ar [i]*k);
          Image1.Bitmap.Canvas.DrawLine(p1, p2,1);
          p1.X := p2.X;
          p1.Y:= p2.Y;
      end;
      // показать ось X
      p1.X :=0;
      p1.Y :=y0;
      p2.X := Round(Image1.Width);
      p2.Y := y0;
      Image1.Bitmap.Canvas.Stroke.Color := TAlphaColors.Black;
      Image1.Bitmap.Canvas.DrawLine(p1, p2,1);
    image1.Bitmap.Canvas.EndScene;
end;
// показать графики Acceleration
procedure TForm1.ShowAcceleration;
var
    min,max : single;
    i: integer;
begin
      Panel1.Visible := False;
      // найти диапазон изменения
      // значений Acceleration
      min:= arx[1];
      max:= arx[1];
      for i:=1 to p do
      begin
       if arx[i] < min then
          min := arx[i];
       if arx[i] > max then
          max := arx[i];
```

```
        if arY[i] < min then
           min := arY[i];
        if arY[i] > max then
           max := arY[i];
        if arZ[i] < min then
           min := arZ[i];
        if arx[i] > max then
           max := arZ[i];
      end;
      Image1.Bitmap := TBitmap.Create(round(Image1.Width), round(Image1.Height));
      Image1.Bitmap.Clear(TAlphaColors.Lightsteelblue );
      Image1.Visible := True;
      // построить графики: AccelerationX, AccelerationY, AccelerationZ
      DrawAcceleration(arX,min,max,TAlphaColors.Red);
      DrawAcceleration(arY,min,max,TAlphaColors.Forestgreen);
      DrawAcceleration(arZ,min,max,TAlphaColors.Darkblue);
end;
// нажатие кнопки Graphics
procedure TForm1.Button2Click(Sender: TObject);
begin
     ShowAcceleration;
end;
// изменилась ориентация устройства
procedure TForm1.Image1Resize(Sender: TObject);
begin
        if mode = 0 then
            ShowAcceleration;
end;
```

Заключение

Вы познакомились с возможностями и технологией программирования в Delphi, узнали, как создать «классическое» приложение для Windows на основе библиотеки VCL, как разработать Multi-Device приложение на основе библиотеки FireDAC и развернуть его на Android устройство, узнали, как создать базу данных на мобильном устройстве. Очевидно, что дать исчерпывающее руководство по программированию на все случаи жизни невозможно. Узнать все возможности среды разработки, научиться их эффективно использовать можно только на практике, решая практические задачи. Поэтому, решайте практические задачи, чем больше вы сделаете самостоятельно, тем большему научитесь!

Для заметок