bhv®

Никита Культин

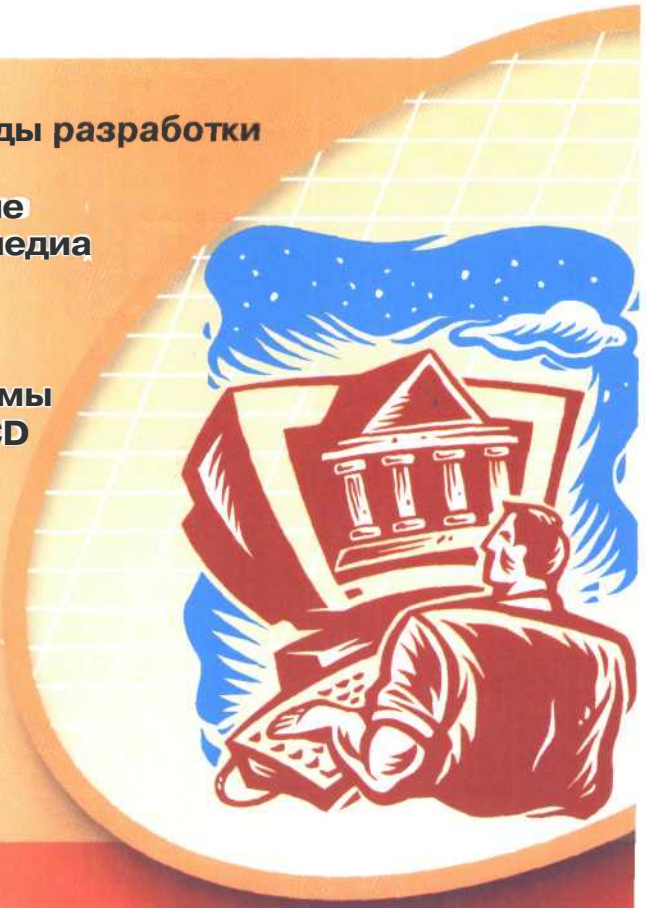
C++ Builder

САМОУЧИТЕЛЬ

Возможности среды разработки

Программирование графики, мультимедиа и баз данных

Создание справочной системы и установочного CD



+CD

Успешное освоение среды быстрой разработки приложений

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Самоучитель C++ Builder. - СПб.: БХВ-Петербург, 2004. – 320 с.: ил.

ISBN 5-94157-378-2

Книга является руководством по программированию в среде Borland C++ Builder. В ней рассматривается весь процесс разработки программы — от компоновки диалогового окна и написания функций обработки событий до отладки и создания справочной системы при помощи программы Microsoft HTML Help Workshop и установочного CD-ROM в InstallShield Express, разбираются вопросы работы с графикой, мультимедиа и базами данных, приведено описание процесса создания анимации в Macromedia Flash 5. Прилагаемый к книге компакт-диск содержит проекты, приведенные в издании в качестве примеров.

Для начинающих программистов

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Григорий Добин*
Редактор	*Галина Смирнова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 17.11.03.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 25,8.
Тираж 4000 экз. Заказ № 1252
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-378-2

Содержание

Предисловие

C++ Builder — что это?

Интерес к программированию постоянно растет. Это связано с развитием и внедрением в повседневную жизнь информационных технологий. Если человек имеет дело с компьютером, то рано или поздно у него возникает желание, а иногда и необходимость, научиться программировать.

Среди пользователей персональных компьютеров в настоящее время наиболее популярна операционная система Windows, и естественно, что тот, кто хочет программировать, хочет и писать программы, которые будут работать в Windows.

Несколько лет назад рядовому программисту оставалось только мечтать о создании своих собственных программ, работающих в Windows. Единственным средством разработки был Borland C++ for Windows, явно ориентированный на профессионалов, обладающих серьезными знаниями и опытом.

Бурное развитие вычислительной техники, потребность в эффективных средствах разработки программного обеспечения привели к появлению на рынке целого ряда систем программирования, ориентированных на так называемую "быструю разработку", среди которых особо следует отметить Microsoft Visual Basic и Borland Delphi. В основе систем быстрой разработки (RAD-систем, Rapid Application Development — среда быстрой разработки приложений) лежит технология визуального проектирования и событийного программирования, суть которой заключается в том, что среда разработки берет на себя большую часть работы по генерации кода программы, оставляя программисту работу по конструированию диалоговых окон и написанию функций обработки событий. Производительность программиста при использовании RAD систем фантастическая!

Успех и популярность Delphi вызвал желание фирмы Borland распространить методы быстрой разработки на область профессионального программирования, что и привело к появлению Borland C++ Builder.

C++ Builder — это среда быстрой разработки, в которой в качестве языка программирования используется язык *C++ Builder* (C++ Builder Language). Не вдаваясь в подробности, можно сказать, что язык C++ Builder — это расширенный C++. Например, в C++ Builder есть строковый (AnsiString) и логический (bool) типы, которых нет в классическом C++.

В настоящее время программистам стала доступна очередная, шестая версия пакета — Borland C++ Builder 6. Как и предыдущие версии, Borland C++ Builder 6 позволяет создавать различные программы: от простейших однооконных приложений до программ управления распределенными базами.

Borland C++ Builder может работать в среде операционных систем от Windows 98 до Windows XP. Особых требований, по современным меркам, к ресурсам компьютера пакет не предъявляет: процессор должен быть типа Pentium или Celeron (рекомендуется Pentium II 400 МГц); объем оперативной памяти должен составлять не менее 128 Мбайт (рекомендуется 256 Мбайт) и свободное дисковое пространство должно быть достаточным (для полной установки версии Enterprise необходимо приблизительно 750 Мбайт).

Об этой книге

Книга, которую вы держите в руках, — не описание среды разработки или языка программирования, а руководство, учебное пособие по основам программирования в C++ Builder. В ней представлена концепция визуального проектирования и событийного программирования, а также рассмотрен процесс создания программы от разработки диалогового окна и функций обработки событий до создания справочной системы и установочного CD-ROM.

Цель этой книги — познакомить читателя с технологией визуального проектирования и событийного программирования и показать на конкретных примерах возможности среды разработки, а также дать методику создания программ. Следует обратить внимание, что хотя книга ориентирована на читателя, обладающего определенными знаниями и начальным опытом в области программирования, она вполне доступна для начинающих.

часть I

Среда разработки C++ Builder

В первой части книги приводится краткое описание среды разработки C++ Builder; на примере программы вычисления силы тока в электрической цепи демонстрируется технология визуального проектирования и событийного программирования; вводятся основные понятия и термины.

ГЛАВА 1

Начало работы

Запускается C++ Builder обычным образом, т. е. выбором из меню Borland **C++Builder** 6 команды **C++Builder** 6 (рис. 1.1).

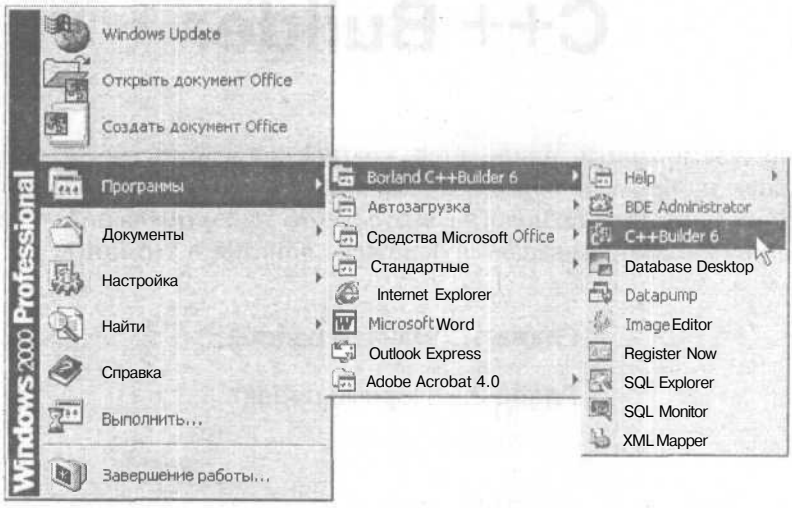


Рис. 1.1. Запуск C++ Builder

Вид экрана после запуска C++ Builder несколько необычен (рис. 1.2). Вместо одного окна на экране появляются пять:

- главное окно — C++Builder 6;
- окно стартовой формы — **Form1**;
- окно редактора свойств объектов — Object Inspector;

- окно просмотра списка объектов — **Object TreeView**;
- окно редактора кода — **Unitl.cpp.**

Окно редактора кода почти полностью закрыто окном стартовой формы.

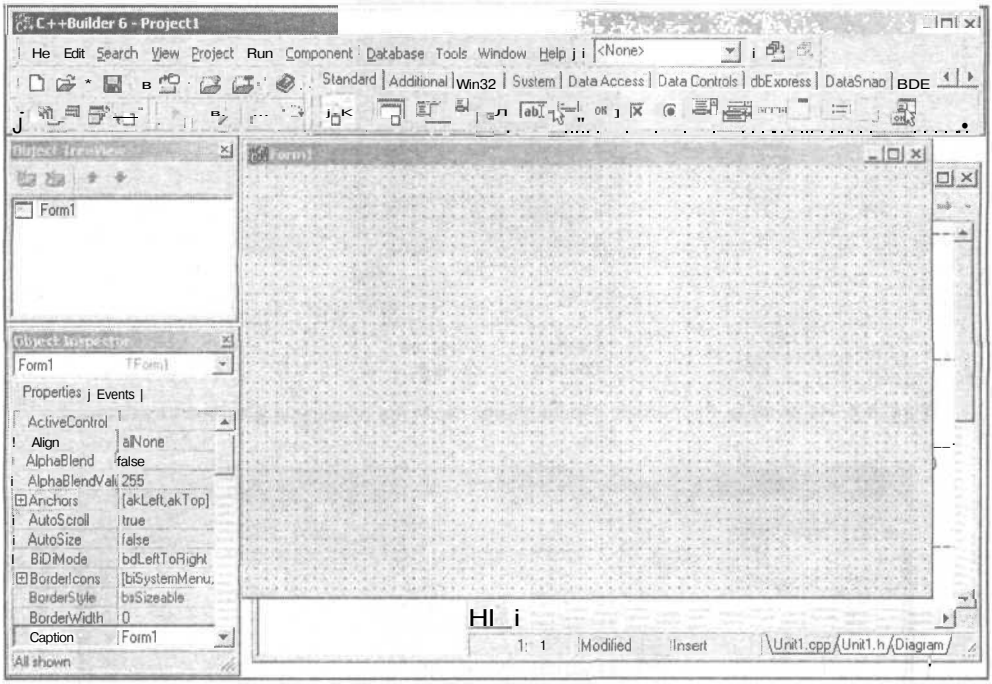


Рис. 1.2. Вид экрана после запуска C++ Builder

В главном окне (рис. 1.3) находится меню команд, панели инструментов и палитра компонентов.

Окно стартовой формы **(Forml)** представляет собой заготовку главного окна разрабатываемой программы (приложения).

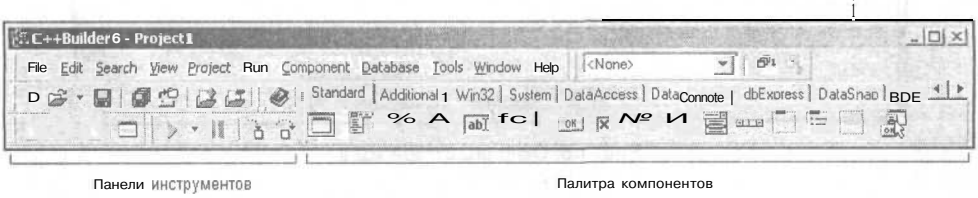


Рис. 1.3. Главное окно

Окно Object Inspector (рис. 1.4) — окно редактора свойств объектов предназначено для редактирования значений свойств объектов. В терминологии

Рис. 1.4. Ма вкладке **Properties** перечислены свойства объекта и указаны их значения

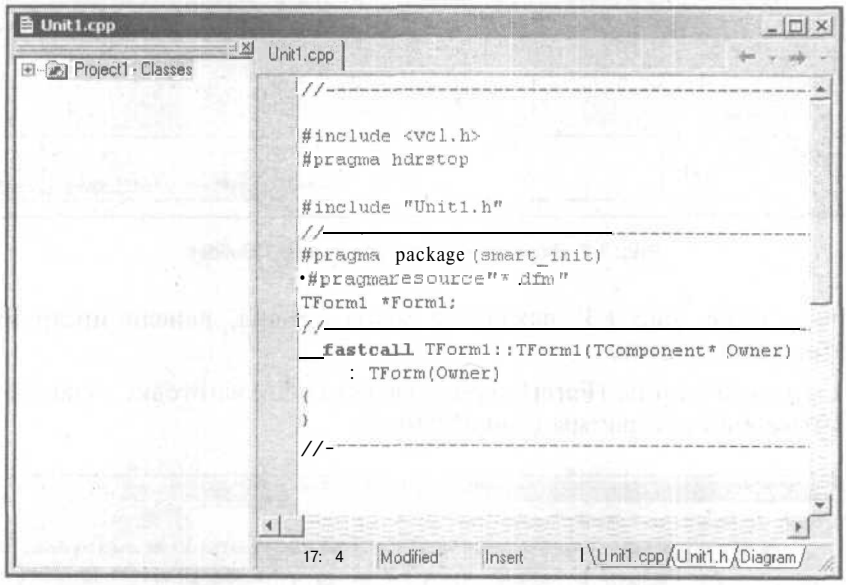


Рис. 1.5. Окно редактора кода

визуального проектирования *объекты* — это диалоговые окна и элементы управления (поля ввода и вывода, командные кнопки, переключатели и др.). *Свойства объекта* — это характеристики, определяющие вид, положение и поведение объекта. Например, свойства width и Height задают раз-

мер (ширину и высоту) формы, свойства top и Left — положение формы на экране, свойство Caption — текст заголовка. В верхней части окна указан объект (имя объекта), значения свойств которого отражены в окне **Object Inspector.**

В окне редактора кода (рис. 1.5), которое можно увидеть, отодвинув в сторону окно формы, следует набирать текст программы. В начале работы над новым проектом окно редактора кода содержит сформированный C++ Builder шаблон программы.

ГЛАВА 2

Первый проект

Для демонстрации возможностей C++ Builder и технологии визуального проектирования и событийного программирования займемся разработкой программы, используя которую можно вычислить силу тока в электрической цепи. Сила тока вычисляется по известной формуле: I = U/R, где U – напряжение источника (вольт); R -- величина сопротивления (Ом). Вид диалогового окна программы во время ее работы (после щелчка на кнопке Вычислить) приведен на рис. 2.1.

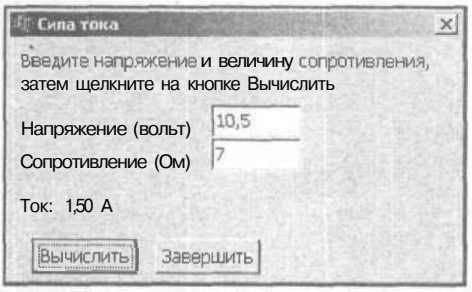


Рис. 2.1. Окно программы вычисления силы тока в электрической цепи

Чтобы начать разработку нового *приложения* (так принято называть прикладные программы), надо запустить C++ Builder или, если C++ Builder уже запущен, в меню File выбрать команду New | Application.

Форма

Работа над новым *проектом* (так в C++ Builder называется разрабатываемое приложение) начинается с создания стартовой *формы* — главного окна программы.

Стартовая форма создается путем изменения значений свойств формы Form1 (настройки формы) и добавления к форме необходимых *компонентов* (полей ввода, полей вывода текстовой информации, командных кнопок).

Основные свойства формы, которые определяют ее вид и поведение во время работы программы, приведены в табл. 2.1.

Таблица 2.1. *Свойства формы (объекта TForm)*

Свойство	Описание
Name	Имя формы. В программе имя формы используется для управления формой и доступа к компонентам формы
Caption	Текст заголовка
Width	Ширина формы
Height	Высота формы
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 при помощи 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 **Свернуть** и кнопки системного меню, 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, в том числе и формы, используется вкладка **Properties** (Свойства) диалогового окна **Object Inspector.** В левой колонке этой вкладки перечислены свойства выбранного объекта, в правой — указаны значения свойств.

При создании формы в первую очередь следует изменить значение свойства caption (Заголовок). В нашем примере надо заменить текст **Form1** на **Сила тока.** Чтобы это сделать, нужно в окне **Object Inspector** щелкнуть левой кнопкой мыши в строке **Caption** (в результате будет выделено значение свойства и появится курсор) и ввести текст: Сила тока (рис. 2.2).

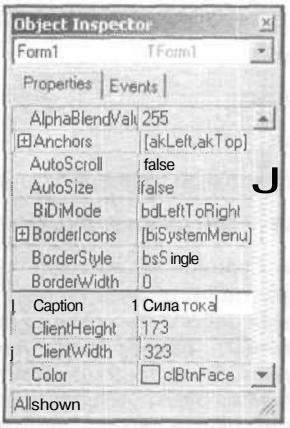


Рис. 2.2. Изменение значения свойства Caption путем ввода значения

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, т. е. точках экрана. Свойствам Height и width надо присвоить значения 200 и 330, соответственно.

Форма — это обычное окно. Поэтому размер формы можно изменить точно так же, как размер любого окна Windows, т. е. путем перетаскивания границы. По окончании перемещения границы значения свойств Height и width автоматически изменятся. Они будут соответствовать установленному размеру формы.

Положение диалогового окна на экране после запуска программы соответствует положению формы во время разработки, которое определяется значением свойств top (отступ от верхней границы экрана) и Left (отступ от левой границы экрана). Значения этих свойств также можно задать путем перемещения формы при помощи мыши.

При выборе некоторых свойств, например, Borderstyle, справа от текущего значения свойства появляется значок раскрывающегося списка. Очевидно, что значение таких свойств можно задать путем выбора из списка (рис. 2.3).

Некоторые свойства являются сложными, т. е. их значение определяется совокупностью значений других (уточняющих) свойств. Например, свойство BorderIcons определяет, какие кнопки управления окном 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 стоит значок "+", в результате щелчка на котором раскрывается список уточняющих свойств (рис. 2.4), значения которых можно задать обычным образом (ввести в поле или выбрать в списке допустимых значений).

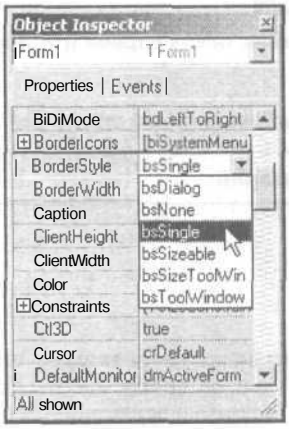


Рис. 2.3. Установка значения свойства путем выбора из списка

Рис. 2.4. Изменение значения уточняющего свойства

В результате выбора некоторых свойств (щелчка кнопкой мыши на свойстве), рядом со значением свойства появляется командная кнопка с тремя точками. Это значит, что задать значение свойства можно в дополнительном диалоговом окне, которое появится в результате щелчка на этой кнопке. Например, значение сложного свойства Font можно задать в окне **Object Inspector** путем ввода значений уточняющих свойств, а можно воспользоваться стандартным диалоговым окном **Шрифт,** которое появится в результате щелчка на кнопке с тремя точками (рис. 2.5).

В табл. 2.2 перечислены свойства формы разрабатываемой программы, которые следует изменить. Остальные свойства формы оставлены без изменения и в таблице не приведены. В приведенной таблице в именах некоторых свойств есть точка. Это значит, что надо задать значение уточняющего свойства.

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

Таблица 2.2. *Значения свойств стартовой формы*

Свойство	Значение	Комментарий
Caption	**Сила тока**	
Height	**200**	
Width	**330**	
BorderStyle	**bsSingle**	**Тонкая граница не позволяет изменить размер окна во время работы программы путем захвата и перемещения** границы
BorderIcons.biMinimize	**False**	**В заголовке окна нет кнопки Свернуть**
BorderIcons.biMaximize	**False**	**В заголовке окна нет кнопки Развернуть**
Font.Name	Tahoma	
Font.Size	**10**	

Компоненты

Программа вычисления тока в электрической цепи должна получить от пользователя исходные данные — напряжение и величину сопротивления. Эти данные могут быть введены с клавиатуры в поля редактирования. Поэтому в форму надо добавить поле редактирования.

Поля редактирования, поля вывода текста, списки, переключатели, командные кнопки и другие элементы пользовательского интерфейса называют *компонентами*.

Для того чтобы в форму разрабатываемого приложения добавить поле редактирования, надо в палитре компонентов, на вкладке Standard, щелкнуть на значке компонента Edit (рис. 2.6), установить курсор в ту точку формы, в которой должен быть левый верхний угол компонента, и еще раз щелкнуть кнопкой мыши. В результате на форме появляется компонент Edit – поле редактирования (рис. 2.7).

Рис. 2.6. Компонент Edit — поле редактирования

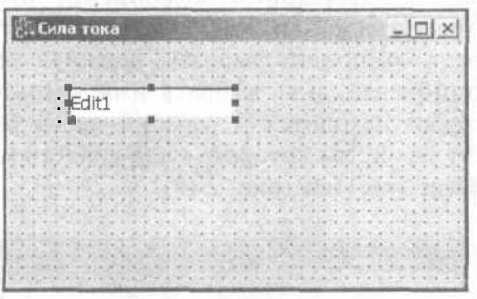


Рис. 2.7. Результат добавления в форму компонента Edit

Каждому добавленному компоненту автоматически присваивается имя, которое состоит из названия компонента и его порядкового номера. Например, если к форме добавить два компонента Edit, то их имена будут Edit1 и Edit2. Программист путем изменения значения свойства Name может изменить имя компонента. Однако в простых программах имена компонентов, как правило, не изменяют.

Основные свойства компонента Edit приведены в табл. 2.3.

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

Свойство	Определяет (задает)
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в том числе к тексту, который находится в поле редактирования
Text	Текст, который находится в поле ввода/редактирования
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы

Таблица 2.3 (окончание)

Свойство	Определяет (задает)
Height	Высоту поля
Width	Ширину поля
Font	Шрифт, используемый для отображения вводимого текста
ParentFont	**Признак**

На рис. 2.8 приведен вид формы после добавления двух полей редактирования. Один из компонентов *выбран* (выделен), окружен восемью маленькими квадратиками. Свойства выбранного компонента отображаются в окне **Object Inspector.** Чтобы увидеть и, если надо, изменить свойства другого компонента, надо этот компонент выбрать, щелкнув левой кнопкой мыши на изображении компонента, или выбрать имя компонента в раскрывающемся списке, который находится в верхней части окна **Object Inspector** (рис. 2.9). Компонент, свойства которого надо увидеть или изменить, можно выбрать и в окне **Object TreeView** (рис. 2.10).

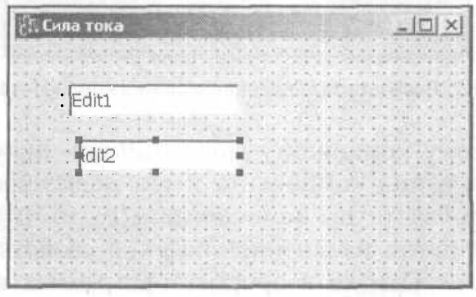


Рис. 2.8. Форма с двумя компонентами

Значения некоторых свойств компонента, определяющих, например, размер и положение компонента на поверхности формы, можно изменить при помощи мыши.

Для того чтобы изменить положение компонента, необходимо установить курсор мыши на его изображение, нажать левую кнопку мыши и, удерживая ее нажатой, переместить контур компонента в нужную точку формы, а затем отпустить кнопку мыши. Во время перемещения компонента (рис. 2.11) отображаются текущие значения координат левого верхнего угла компонента (значения свойств Left и top).

Для того чтобы изменить размер компонента, необходимо его выделить, установить указатель мыши на один из маркеров, помечающих границу

компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить положение границы компонента. Затем отпустить кнопку мыши. Во время изменения размера компонента отображаются его текущие размеры: высота и ширина (значения свойств Height и Width) (рис. 2.12).

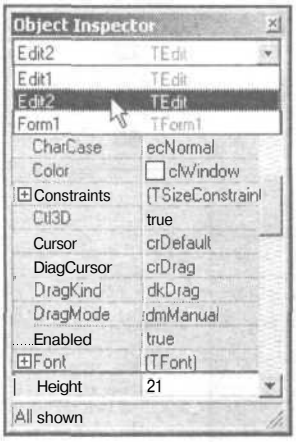


Рис. 2.9. Выбор компонента в окне **Object Inspector**

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

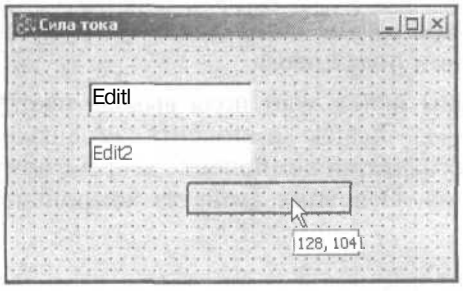


Рис. 2.11. Отображение значений свойств Left и Top при изменении положения компонента

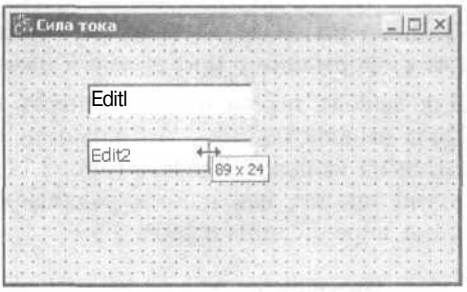


Рис. 2.12. Отображение значений свойств Height и Width при изменении размера компонента

В табл. 2.4 приведены значения свойств компонентов Edit1 и Edit2 разрабатываемого приложения (значения остальных свойств оставлены без изменения и поэтому в таблице не приведены). Компонент Edit1 предназначен для ввода величины напряжения, Edit2 — для ввода величины сопротивления. Обратите внимание, значением свойства Text обоих компонентов является пустая строка. В результате форма разрабатываемого приложения должна выглядеть так, как показано на рис. 2.13.

Таблица 2.4. *Значения свойств компонентов* `Edit1` *и* `Edit2`

Свойство	Компонент	
	Edit1	Edit2
Text		
Top	48	72
Left	144	144
Width	65	65

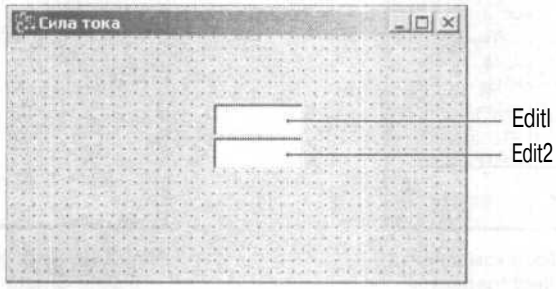


Рис. 2.13. Форма после настройки компонентов Edit

Помимо полей редактирования в окне программы находится текст — краткая информация о программе и назначении полей ввода.

Для вывода текста на поверхность формы используют поля вывода текста. Поле вывода текста — это компонент Label. Значок компонента Label находится на вкладке **Standard** (рис. 2.14). Добавляется компонент Label в форму точно так же, как и поле редактирования. Основные свойства компонента Label перечислены в табл. 2.5.

Рис. 2.14. Компонент Label — поле вывода текста

Таблица 2.5. *Свойства компонента Label*

Свойство	Определяет (задает)
Name	Имя компонента. Используется в программе для доступа к свойствам компонента

Таблица 2.5 (окончание)

Свойство	Определяет (задает)
Caption	**Отображаемый текст**
Font	Шрифт, используемый для отображения текста
ParentFont	Признак наследования шрифта родительского компонента
AutoSize	Признак того, что размер поля определяется его содержимым
Left	Расстояние от левой границы поля вывода до левой границы формы
Top	Расстояние от верхней границы поля вывода до верхней границы формы
Height	**Высоту поля вывода**
Width	**Ширину поля вывода**
Wordwrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства AutoSize должно быть false)

Если поле Label должно содержать несколько строк текста, то перед тем как ввести в поле текст (изменить значение свойства Caption), нужно присвоить свойству AutoSize значение false, а свойству wordwrap — true. Затем надо установить требуемый размер поля (при помощи мыши или вводом значений свойств Height и width) и только после этого ввести значение свойства Caption.

В форму разрабатываемого приложения надо добавить четыре компонента Label. Поле Label1 предназначено для вывода информационного сообщения, поля Label2 и Label3 — для вывода информации о назначении полей ввода, поле Label4 — для вывода результата расчета (величины тока в цепи). После добавления компонентов надо выполнить их настройку — установить значения свойств (табл. 2.6). Прочерк в таблице означает, что значение свойства оставлено без изменения или установлено автоматически — например, как результат изменения другого свойства. В результате форма разрабатываемого приложения должна выглядеть так, как показано на рис. 2.15.

Таблица 2.6. *Значения свойств компонентов Label1—Label4*

Свойство	Компонент			
	Label1	**Label2**	**Label3**	**Label4**
AutoSize	false	true	true	false
Wordwrap	true	false	false	true

Таблица 2.6 *(окончание)*

Свойство	Компонент			
	Label1	Label2	Label3	Label4
Caption	Введите напряжение и величину сопротивления, затем щелкните на кнопке Вычислить	Напряжение (вольт)	Сопротивление (Ом)	
Top	8	56	80	112
Left	8	8	8	8
Height	33	16	16	16
Width	300	120	120	200

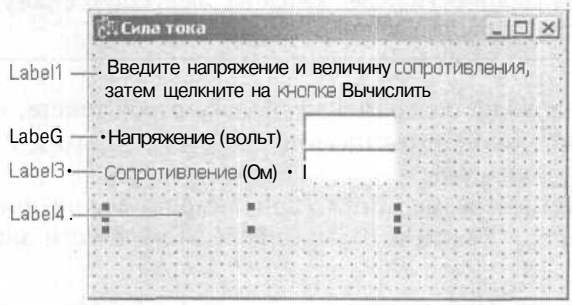


Рис. 2.15. Вид формы после добавления и настройки полей вывода текста

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

Командная кнопка — компонент Button — добавляется в форму точно так же, как и другие компоненты. Значок компонента Button находится на вкладке **Standard** (рис. 2.16). Основные свойства компонента Button приведены в табл. 2.7.

Рис. 2.16. Командная кнопка — компонент Button

После добавления к форме двух командных кнопок нужно установить значения их свойств в соответствии табл. 2.7.

Таблица 2.7. *Свойства компонента Button (командная кнопка)*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
Caption	**Текст на кнопке**
Enabled	Признак доступности кнопки. Кнопка доступна, если значение свойства равно true, и недоступна, 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

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

Окончательный вид формы разрабатываемого приложения приведен на рис. 2.17.

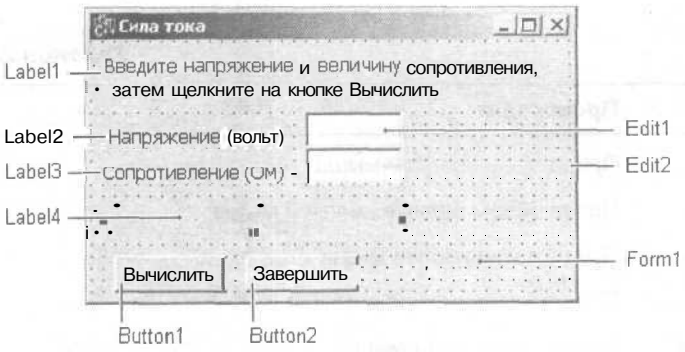


Рис. 2.17. Окончательный вид формы программы "Сила тока"

Завершив работу над формой, можно приступить к созданию программы. Но перед этим рассмотрим два важных понятия: событие и функцию обработки события.

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

Свойство	Компонент	
	Button1	Button2
Caption	Вычислить	Завершить
Top	144	144
Left	16	104
Height	25	25
Width	75	75

Событие и функция обработки события

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

Событие (Event) — это то, что происходит во время работы программы. В C++ Builder каждому событию присвоено имя. Например, щелчок кнопкой мыши -- это событие onclick, двойной щелчок мышью — событие `OnDblClick`.

В табл. 2.9 приведены некоторые события Windows.

Таблица 2.9. *События*

Событие	Происходит
Onclick	При щелчке кнопкой мыши
OnDblClick	При двойном щелчке кнопкой мыши
OnMouseDown	При нажатии кнопки мыши
OnMouseUp	При отпускании кнопки мыши
OnMouseMove	При перемещении мыши
OnKeyPress	При нажатии клавиши клавиатуры
OnKeyDown	При нажатии клавиши клавиатуры. События OnKeyDown и OnKeyPress — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие `OnKeyUp`)

Таблица 2.9 *(окончание)*

Событие	Происходит
OnKeyUp	При отпускании нажатой клавиши клавиатуры
OnCreate	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
OnPaint	При появлении окна на экране в начале работы программы; во время работы программы после появления окна (или его части), которое было закрыто другим окном или свернуто
OnEnter	При получении элементом управления фокуса
OnExit	При потере элементом управления фокуса

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

Методику создания функций обработки событий рассмотрим на примере функции обработки события onclick для командной кнопки **Вычислить.**

Чтобы приступить к созданию функции обработки события, сначала надо выбрать компонент, для которого создается функция обработки события. Выбрать компонент можно в окне **Object Inspector** или щелчком на изображении компонента в форме. После этого в окне **Object Inspector** нужно выбрать вкладку **Events** (События).

В левой колонке вкладки **Events** (рис. 2.18) перечислены события, которые может воспринимать выбранный компонент (имя и тип компонента указаны в верхней части окна). Если для события определена функция обработки, то в правой колонке рядом с именем события будет выведено имя этой функции.

Для того чтобы создать функцию обработки события, нужно сделать двойной щелчок мышью в окне **Object Inspector,** в поле функции обработки соответствующего события (рис. 2.19). В результате этого откроется окно ре-

дактора кода, в которое будет добавлен шаблон функции обработки события, а в окне **Object Inspector** рядом с именем события появится сгенерированное C++ Builder имя функции обработки события (рис. 2.19).

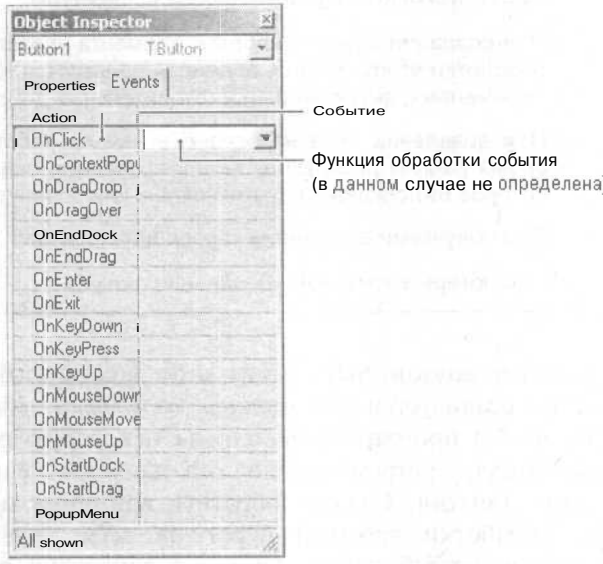


Рис. 2.18. На вкладке **Events** перечислены события, которые может воспринимать компонент (в данном случае — командная кнопка)

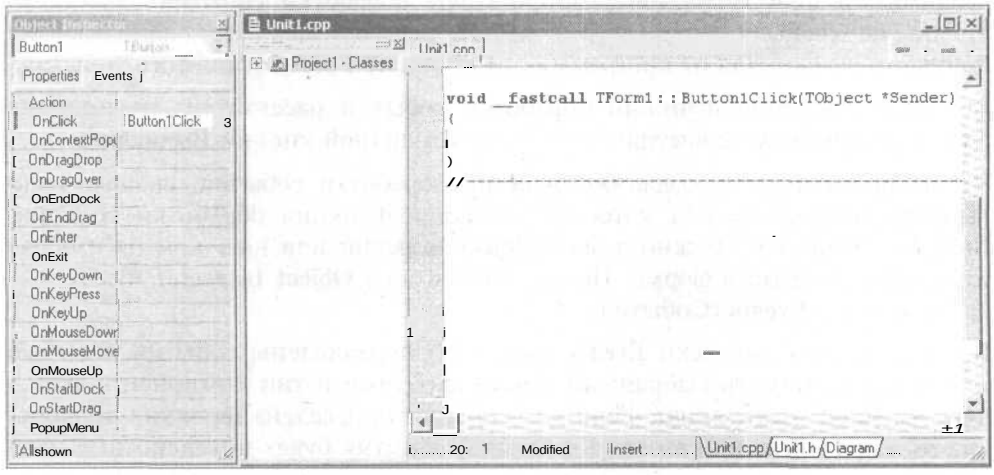


Рис. 2.19. Шаблон функции обработки события, сгенерированный C++ Builder

C++ Builder присваивает функции обработки события имя, которое состоит из двух частей. Первая часть имени идентифицирует форму, содержащую

объект (компонент), для которого создана процедура обработки события. Вторая часть имени идентифицирует сам объект и событие. В нашем примере имя формы — Form1, имя командной кнопки — Button1, а имя события — Click.

В окне редактора кода между фигурными скобками можно набирать инструкции, реализующие функцию обработки события.

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

Листинг 2.1. Простейшая обработка события Onclick на кнопке Вычислить

```
void _ fastcall TForm1: :Button1Click (TObject *Sender)
{
    float u; // напряжение
    float r; // сопротивление
    float i; // ток

    // получить данные из полей ввода
    u = StrToFloat (Edit1->Text);
    r = StrToFloat (Edit2->Text);

    // вычислить ток
    i = u/r;

    // вывести результат в поле метки
    Label4->Caption = "Ток : " +
                      FloatToStrF(i,ffGeneral,7,2) + " A";
}
```

Функция Button1Click выполняет расчет силы тока и выводит результат расчета в поле Label4. Исходные данные вводятся из полей редактирования Edit1 и Edit2 путем обращения к свойству Text. Свойство Text содержит строку символов, которую ввел пользователь. Чтобы программа работала правильно, пользователь должен ввести в каждое поле редактирования целое или дробное число в правильном формате (при вводе дробного числа для разделения целой и дробной частей надо использовать запятую). Так как поле редактирования содержит текст (свойство Text строкового типа),

необходимо выполнить преобразование строки в число. Эту задачу решает функция StrToFioat, которой в качестве параметра передается содержимое поля редактирования — значение свойства Text (Edit1->Text — это содержимое поля Edit1). Функция strToFioat проверяет символы строки, переданной ей в качестве параметра, на допустимость и, если все символы верные, возвращает значение, соответствующее строке, полученной в качестве параметра.

После того как исходные данные будут помещены в переменные и и r, выполняется расчет.

Вычисленная величина силы тока выводится в поле Label4 путем присваивания значения свойству caption. Для преобразования числа в строку символов (свойство caption - строкового типа) используется функция FloatToStrF.

В листинге 2.2 приведена процедура обработки события Onclick на командной кнопке **Завершить.** Создается она точно так же, как и процедура обработки события onclick для командной кнопки **Вычислить.** В результате щелчка на кнопке **Завершить** программа должна завершить работу. Чтобы это произошло, надо закрыть окно программы. Делает это метод close.

Листинг 2.2. Процедура **обработки события** Onclick **на кнопке Завершить**

```
void__fastcall TForm1::Button2Click(TObject *Sender)
{
    Form1->Close();
}
```

Редактор кода

Во время набора текста программы редактор кода автоматически выделяет элементы программы: полужирным шрифтом — ключевые слова языка программирования (if, else, int, float и др.), курсивом — комментарии. Это делает текст программы более выразительным, что облегчает восприятие структуры программы.

В процессе разработки программы часто возникает необходимость переключения между окном редактора кода и окном формы. Сделать это можно при помощи командной кнопки **Toggle Form/Unit,** которая находится на панели инструментов **View** (рис. 2.20), или нажав клавишу <F12>. На панели инструментов View находятся командные кнопки **View Unit и View Form,** используя которые можно выбрать нужный модуль или форму в случае, если проект состоит из нескольких модулей или форм.

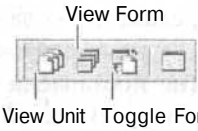


Рис. 2.20. Кнопка **Toggle Form/Unit** позволяет быстро переключаться между формой и редактором кода

Система подсказок

Редактор кода поддерживает функцию контекстно-зависимой подсказки, которая во время набора текста программы автоматически выводит краткую справочную информацию о свойствах и методах объектов, о параметрах функций.

Например, после того как будет набрано имя объекта (компонента) и символы ->, редактор кода автоматически выведет список свойств и методов объекта (рис. 2.21). Программисту останется только выбрать из списка нужный элемент и нажать клавишу <Enter> (быстро перейти к нужному элементу списка или к области, где этот элемент находится, можно, нажав клавишу, соответствующую первому символу этого элемента).

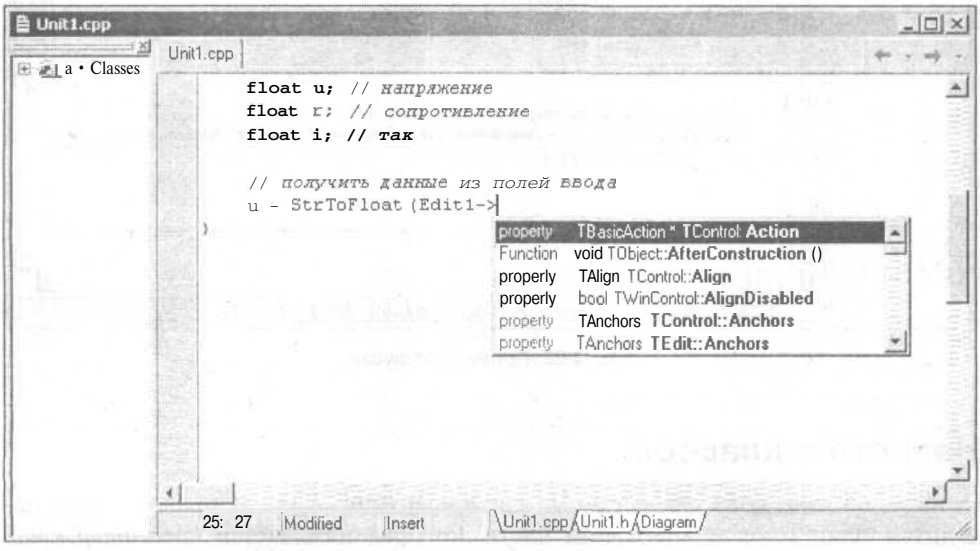


Рис. 2.21. Редактор кода автоматически выводит список свойств и методов объекта (компонента)

Следует обратить внимание, что если список свойств и методов не появляется, то это значит, что в программе обнаружена ошибка (C++ Builder контролирует правильность набираемого программистом текста в "фоновом"

режиме). Например, если в окне редактора кода набрать if Edit1->, то список свойств и методов объекта Edit1 не появится, т. к. инструкция if в данном случае записана с ошибкой (не поставлена открывающая скобка после if). C++ Builder информирует программиста об обнаруженной ошибке сообщением Unable to invoke Code Completion due to errors in source code, которое появляется в нижней части окна редактора кода.

После набора имени встроенной или объявленной программистом функции редактор кода также выводит подсказку: список параметров. Параметр, который в данный момент вводит программист, в подсказке выделен полужирным. Например, если набрать слово FloatToStrF, которое является именем функции преобразования дробного числа в строку символов, и открывающую скобку, то на экране появится окно, в котором будет указан список параметров функции (рис. 2.22).

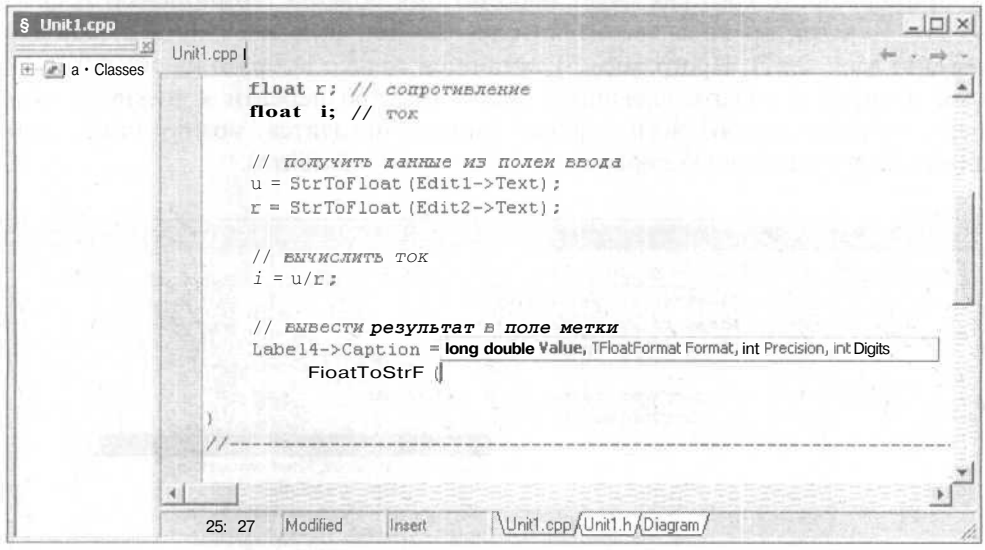


Рис. 2.22. Пример подсказки

Навигатор классов

Окно редактора кода разделено на две части (рис. 2.23). В правой части находится текст программы. Левая часть, которая называется *навигатор классов* (ClassExplorer), облегчает навигацию по тексту (коду) программы. В иерархическом списке, структура которого зависит от проекта, над которым идет работа, перечислены объекты проекта (формы и компоненты) и функции обработки событий. Выбрав элемент списка, можно быстро перейти к нужному фрагменту кода, например к функции обработки события.

Рис. 2.23. Окно **ClassExplorer** облегчает навигацию по тексту программы

Окно навигатора классов можно закрыть обычным образом. Если окно навигатора классов не доступно, то для того чтобы оно появилось на экране, нужно в меню **View** выбрать команду **ClassExplorer.**

Шаблоны кода

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

```
if  O
{

}
else
{

}
```

Редактор кода предоставляет программисту большой набор шаблонов: объявления классов, ФУНКЦИЙ, ИНСТРУКЦИЙ Выбора (if, switch), ЦИКЛОВ (for,

while). Для некоторых инструкций, например для if и while, есть несколько вариантов шаблонов.

Для того чтобы в процессе набора текста программы воспользоваться шаблоном кода и вставить его в текст программы, нужно нажать комбинацию клавиш <Ctrl>+<J> и из появившегося списка выбрать нужный шаблон (рис. 2.24). Выбрать шаблон можно обычным образом, прокручивая список, или вводом первых букв имени шаблона (имена шаблонов в списке выделены полужирным). Выбрав в списке шаблон, нужно нажать клавишу <Enter>, шаблон будет вставлен в текст программы.

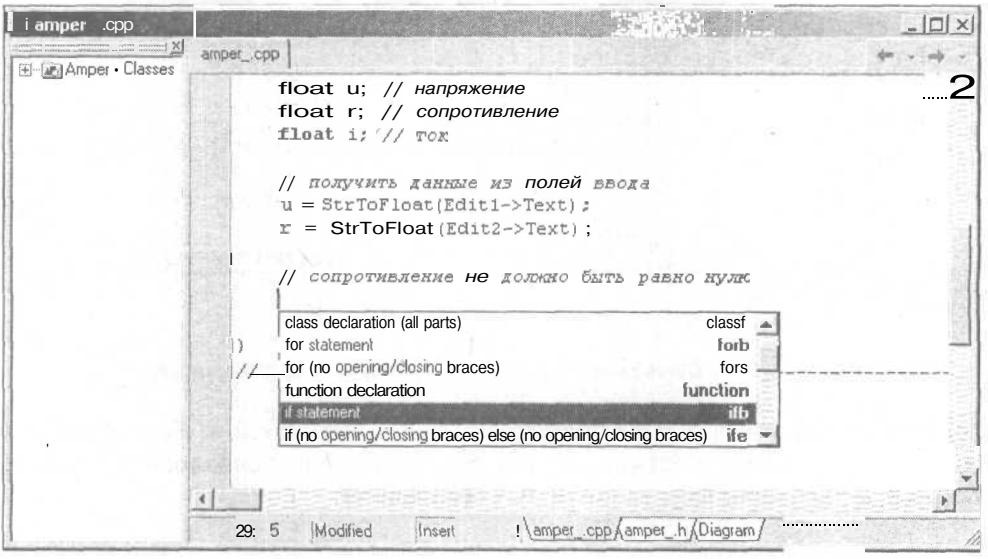


Рис. 2.24. Список шаблонов кода отображается в результате нажатия <Ctrl>+<J>

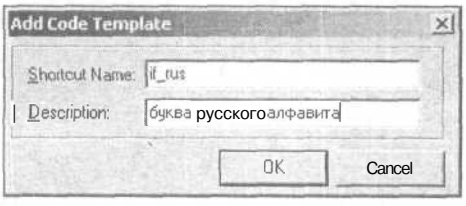


Рис. 2.25. В поля диалогового окна надо ввести имя шаблона и его краткое описание

Программист может создать свой собственный шаблон кода и использовать его точно так же, как и стандартный. Для того чтобы создать шаблон кода, нужно в меню **Tools** выбрать команду **Editor Options** и в окне **Code Insight** щелкнуть на кнопке Add. В появившемся окне **Add Code Template** (рис. 2.25) надо задать имя шаблона (**Shortcut Name)** и его краткое описание (De-

scription). Затем, после щелчка на кнопке OK, в поле **Code** надо ввести шаблон (рис. 2.26).

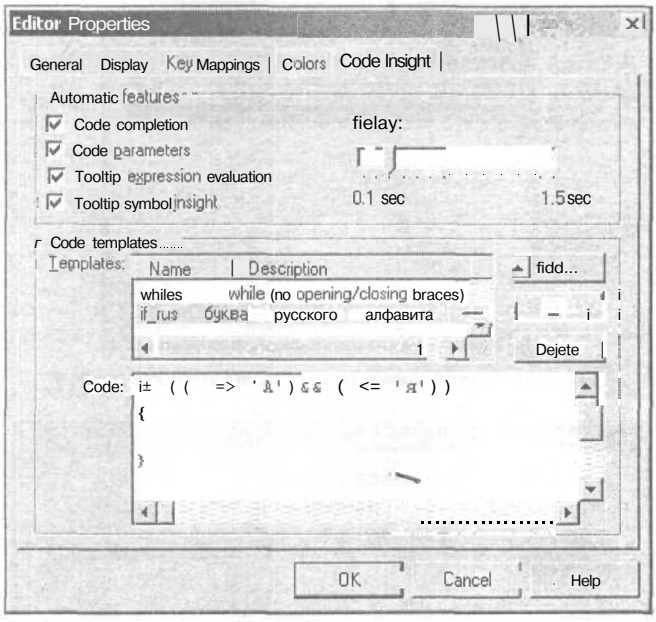


Рис. 2.26. Пример шаблона кода программиста

Справочная система

В процессе набора текста программы можно получить справку о конструкции языка, типе данных, классе или функции. Для этого нужно в окне редактора кода набрать слово, о котором надо получить справку (например, имя функции), и нажать клавишу <F1>. Так как с запрашиваемой темой в справочной системе может быть связано несколько разделов, на экране, как правило, появляется окно **Найденные разделы** (рис. 2.27), в котором можно выбрать нужный раздел. Следует обратить внимание на то, что после имени функции может быть указано имя библиотеки, к которой эта функция относится: VCL или CLX (вспомните: библиотека VCL используется при разработке приложений для Windows, а CLX — при разработке кроссплатформенных приложений). Поэтому, выбирая раздел справочной системы, надо обращать внимание на то, к какой библиотеке он относится.

Справочную информацию можно получить также, выбрав из меню **Help** команду C++ **Builder Help.** В этом случае на экране появится стандартное окно справочной системы. В этом окне на вкладке **Предметный указатель** нужно ввести ключевое слово, определяющее тему, по которой нужна

справка. В качестве ключевого слова можно ввести, например, первые несколько букв имени функции, свойства или метода (рис. 2.28).

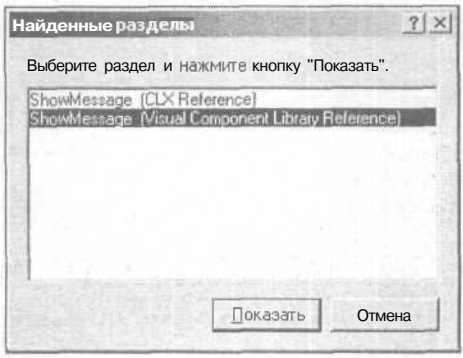


Рис. 2.27. В диалоговом окне следует уточнить раздел

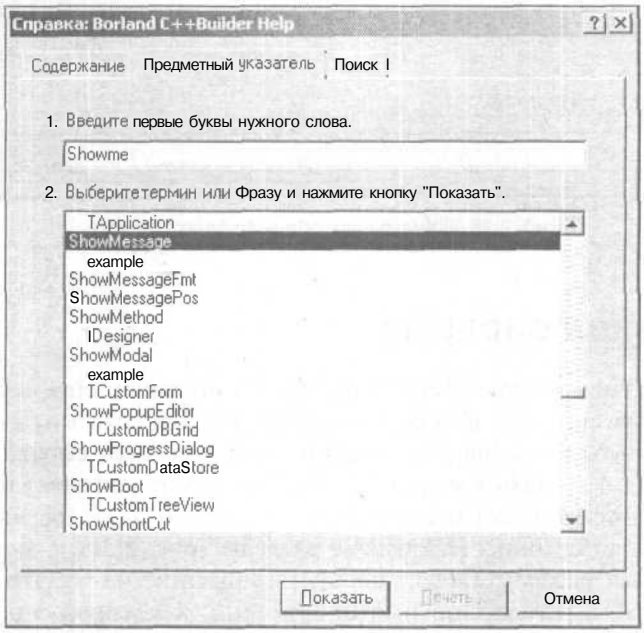


Рис. 2.28. Поиск справочной информации по ключевому слову

Сохранение проекта

Проект — это набор файлов, используя которые компилятор создает выполняемый файл программы (exe-файл). В простейшем случае проект составляют: файл описания проекта (bpr-файл), файл главного модуля (cpp-файл),

файл ресурсов (res-файл), файл описания формы (dfm-файл), заголовочный файл формы (h-файл) и файл описания функций формы (cpp-файл).

Чтобы сохранить проект, нужно в меню **File** выбрать команду **Save Project** As. Если проект еще ни разу не был сохранен, то C++ Builder сначала предлагает сохранить модуль (содержимое окна редактора кода) и поэтому на экране появляется окно **Save** Unit1 As. В этом окне (рис. 2.29) надо выбрать папку, предназначенную для проектов, создать в ней папку для сохраняемого проекта, открыть ее и ввести имя модуля. В результате щелчка на кнопке

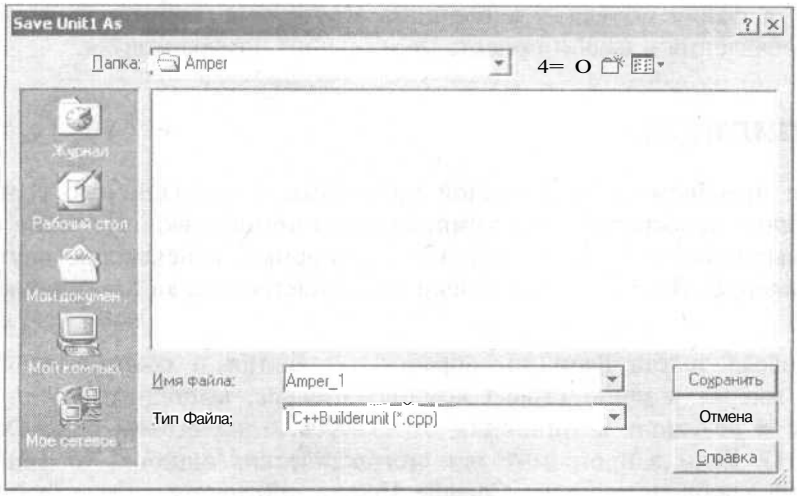


Рис. 2.29. Сохранение модуля

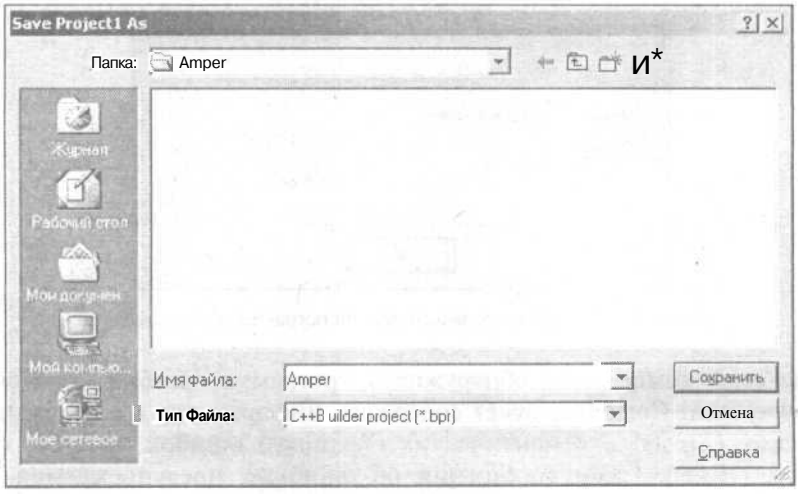


Рис. 2.30. Сохранение проекта

OK в указанной папке будут созданы три файла: cpp, h и dfm, и на экране появится диалоговое окно **Save Projectl** As (рис. 2.30), в которое надо ввести имя проекта.

Обратите внимание, что имена файла модуля (cpp) и файла проекта (bpr) должны быть разными, т. к. C++ Builder в момент сохранения файла проекта создает одноименный cpp-файл (файл главного модуля). Кроме того, надо учесть, что имя генерируемого компилятором выполняемого файла совпадает с именем проекта. Поэтому файлу проекта следует присвоить такое имя, которое, по вашему мнению, должен иметь выполняемый файл программы, а файлу модуля — какое-либо другое имя, например, полученное путем добавления к имени проекта порядкового номера модуля.

Компиляция

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

После ввода текста функции обработки события и сохранения проекта можно, выбрав в меню **Project** команду **Compile**, выполнить компиляцию. Процесс и результат компиляции отражается в диалоговом окне **Compiling** (рис. 2.31). Если ,в программе нет синтаксических ошибок, то окно будет содержать сообщение: **Done: Compile Unit,** в противном случае будет выведено сообщение **Done: There are errors.**

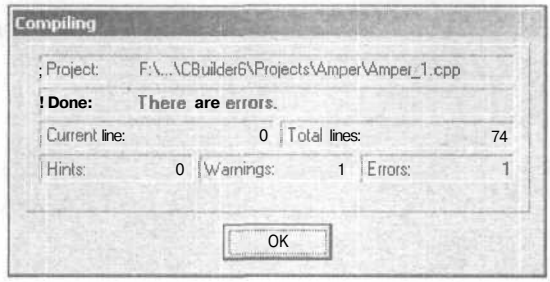


Рис. 2.31. Результат компиляции: в программе есть ошибки

В случае если компилятор обнаружит в программе ошибки и неточности, диалоговое окно **Compiling** будет содержать информацию о количестве синтаксических (Errors) и семантических (Warnings) ошибок, а также о числе подсказок (Hints). Сами сообщения об ошибках, предупреждения и подсказки находятся в нижней части окна редактора кода.

Чтобы перейти к фрагменту кода, который, по мнению компилятора, содержит ошибку, надо выбрать сообщение об ошибке (щелкнуть в строке сообщения левой кнопкой мыши) и из контекстного меню (рис. 2.32) выбрать команду **Edit Source.**

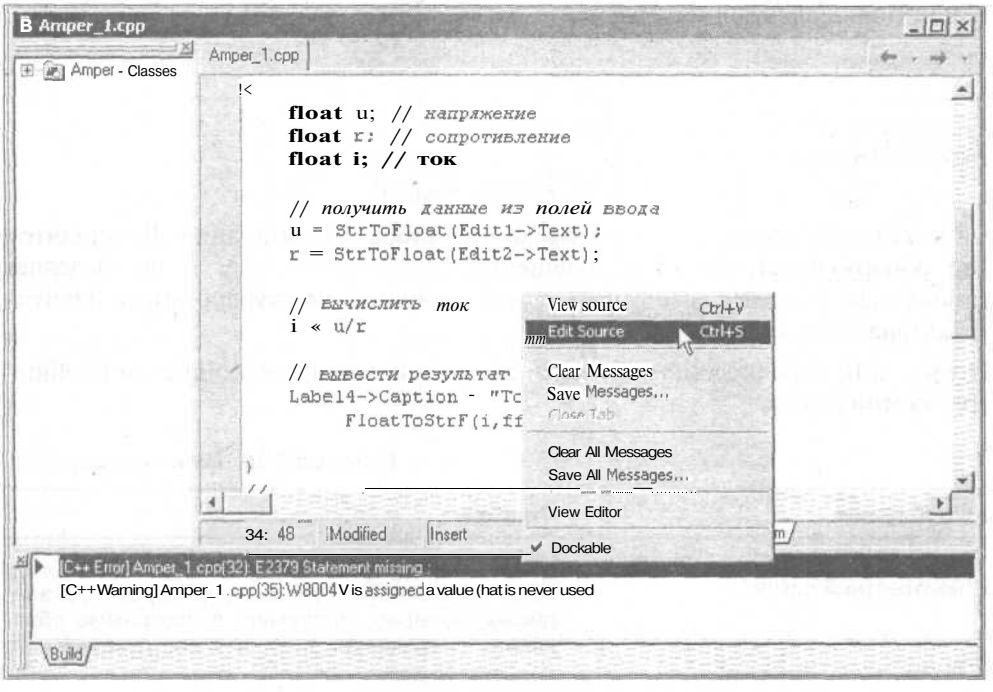


Рис. 2.32. Переход к фрагменту программы, который содержит ошибку

Процесс компиляции можно активизировать, выбрав в меню **Run** команду **Run,** которая запускает разрабатываемое приложение. Если будет обнаружено, что с момента последней компиляции в программу были внесены изменения или программа еще ни разу не компилировалась, то будет выполнена компиляция, затем — компоновка, и после этого программа будет запущена (естественно, только в том случае, если в программе нет ошибок).

Ошибки

Компилятор переходит ко второму этапу генерации выполняемой программы только в том случае, если исходный текст не содержит синтаксических ошибок. В большинстве случаев в только что набранной программе есть ошибки. Программист должен их устранить. Процесс устранения ошибок носит итерационный характер. Обычно сначала устраняются наиболее очевидные ошибки, например, объявляются необъявленные переменные. После очередного внесения изменений в текст программы выполняется повторная

компиляция. Следует обратить внимание на то, что компилятор не всегда может точно локализовать ошибку. Поэтому, анализируя фрагмент программы, который, по мнению компилятора, содержит ошибку, нужно обращать внимание не только на тот фрагмент кода, на который компилятор установил курсор, но и на тот, который находится в предыдущей строке. Например, в следующем фрагменте кода:

```
// вычислить ток
i = u/r
// вывести результат в поле метки
Label4->Caption = "Ток : " +
                  FloatToStrF(i,ffGeneral,7,2) + " A";
```

не поставлена точка с запятой после оператора присваивания. Компилятор это обнаруживает, выводит сообщение statement missing ;, но выделяет строку `Label4->Caption = "ток : " +` и устанавливает курсор после идентификатора Label4.

В табл. 2.10 перечислены типичные ошибки и соответствующие им сообщения компилятора.

Таблица 2.10. *Типичные ошибки*

Сообщение	Ошибка
Undefined symbol **(неизвестный символ)**	Используется необъявленная переменная Имя переменной, функции или параметра записано неверно. Например, в программе объявлена переменная Summ, а в инструкциях используется sum
Statement missing ; **(отсутствует точка с запятой)**	После инструкции не поставлена точка с запятой
Unterminated string or character constant **(незаконченная строковая или символьная константа)**	В конце строковой константы, например, текста сообщения, нет двойных кавычек
) expected (ожидается закрывающая скобка)	При записи арифметического выражения, содержащего скобки, нарушен баланс открывающих и закрывающих скобок
if statement missing (**(в инструкции** if **нет открывающей скобки)**	В инструкции if условие не заключено в скобки
Compound statement missing }	Нарушен баланс открывающих и закрывающих фигурных скобок. Вероятно, не поставлена закрывающая фигурная скобка отмечающая конец функции или группы инструкций, например, после условия или слова else в инструкции **if**

Таблица 2.10 (окончание)

Сообщение	Ошибка
Extra parameter in call to (лишний параметр при вызове функции)	Неверно записана инструкция вызова функции, указан лишний параметр

Если компилятор обнаружил достаточно много ошибок, то просмотрите все сообщения и устраните сначала наиболее очевидные ошибки и выполните повторную компиляцию. Вполне вероятно, что после этого количество ошибок значительно уменьшится. Это объясняется особенностями синтаксиса языка, когда одна незначительная ошибка может "тащить" за собой довольно большое количество других.

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 (... is declared but never used.). Действительно, зачем объявлять переменную и не использовать ее?

В табл. 2. И приведены предупреждения, наиболее часто выводимые компилятором.

Таблица 2.11. Предупреждения компилятора

Предупреждение	Вероятная причина
... is declared but never used	
Possibly incorrect assignment. (вероятно, инструкция присваивания некорректная)	В условии, например, инструкции if, вместо оператора сравнения (==) использован оператор присваивания (=)
Possibly use of ... before definition. (вероятно, используется неинициализированная переменная)	Не присвоено начальное значение переменной

Компоновка

Если в программе нет ошибок, то можно выполнить компоновку. Для этого надо в меню **Compile** выбрать команду **Make** или **Build.** Разница между командами **Make и Build** заключается в следующем. Команда **Make** обеспе-

чивает компоновку файлов проекта, а команда **Build** — принудительную перекомпиляцию, а затем — компоновку.

На этапе компоновки также могут возникнуть ошибки. Чаще всего причина ошибок во время компоновки состоит в недоступности файлов библиотек или других ранее откомпилированных модулей. Устраняются эти ошибки путем настройки среды разработки и включением в проект недостающих модулей. В простых проектах ошибки времени компиляции, как правило, не возникают.

Запуск программы

Пробный запуск программы можно выполнить непосредственно из среды разработки, не завершая работу с C++ Builder. Для этого нужно в меню **Run** выбрать команду **Run** или щелкнуть на командной кнопке **Run** (рис. 2.33).

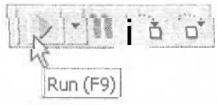


Рис. 2.33. Запуск программы из среды разработки

Ошибки времени выполнения

Во время работы приложения могут возникать ошибки, которые называются *ошибками времени выполнения* (run time errors) или *исключениями* (exceptions). В большинстве случаев причинами исключений являются неверные исходные данные.

Например, если во время работы программы вычисления силы тока в поле **Напряжение** ввести `10.5`, т. е. разделить целую и дробную часть точкой, то в результате щелчка на кнопке Вычислить на экране появится окно с сообщением об ошибке (рис. 2.34).

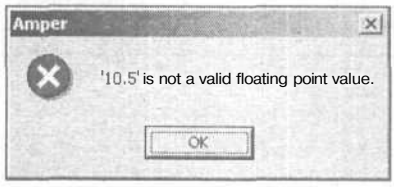


Рис. 2.34. Пример окна с сообщением об ошибке времени выполнения (программа запущена из Windows)

Причина возникновения ошибки в следующем. В тексте программы дробная часть числа от целой отделяется точкой. При вводе данных в поле редактирования пользователь может отделить дробную часть числа от целой точкой или запятой. Какой из этих двух символов является правильным, зависит от настройки Windows.

Если в настройке Windows указано, что разделитель целой и дробной частей числа — запятая (для России это стандартная установка), *а* пользователь использовал точку (ввел в поле **Напряжение** (`Edit1`) строку `10.5`), то при выполнении инструкции

```
u = StrToFloat(Edit1->Text);
```

возникнет исключение, т. к. при указанной настройке Windows содержимое поля редактирования и, следовательно, аргумент функции `StrToFloat` не является изображением дробного числа.

Если программа запущена из среды разработки, то при возникновении исключения выполнение программы приостанавливается и на экране появляется окно с сообщением об ошибке и ее типе. В качестве примера на рис. 2.35 приведено окно сообщения о возникновении исключения, причина которого заключается в том, что строка, введенная пользователем в поле редактирования, не является дробным числом.

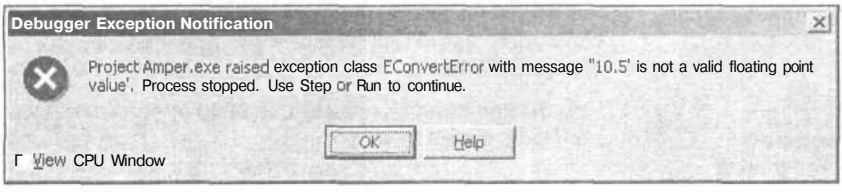


Рис. 2.35. Пример сообщения о возникновении исключения (программа запущена из C++ Builder)

После возникновения исключения и щелчка на кнопке OK в диалоговом окне **Debugger Exception Notification** (рис. 2.35) выполнение программы можно прервать или, несмотря на возникшую ошибку, продолжить. Чтобы прервать выполнение программы, надо в меню **Run** выбрать команду **Program Reset,** чтобы продолжить — команду **Step Over.**

Обработку исключений берет на себя автоматически добавляемый в выполняемую программу код, который обеспечивает, в том числе, и вывод информационного сообщения. Вместе с тем C++ Builder дает возможность программе самой выполнить обработку исключения.

Инструкция обработки исключения выглядит так:

```
try
{
    // здесь инструкции, выполнение которых может вызвать исключение
}
```

```
catch ( Тип &e)
{
    // здесь инструкции обработки исключения
}
```

где:

- try -- ключевое слово, обозначающее, что далее следуют инструкции, при выполнении которых возможно возникновение исключений, и что обработку этих исключений берет на себя программа;
- catch -- ключевое слово, обозначающее начало секции обработки исключения. Инструкции этой секции будут выполнены, если в программе возникнет исключение указанного типа.

Основной характеристикой исключения является его тип. В табл. 2.12 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 2.12. Типичные исключения

Исключение	Возникает
EConvertError — ошибка преобразования	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число
EDivByZero — целочисленное деление на ноль	При выполнении операции целочисленного деления, если делитель равен нулю
EZeroDivide — деление на ноль	При выполнении операции деления над дробными операндами, если делитель равен нулю
EInOutError — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частой причиной является отсутствие требуемого файла или, в случае использования сменного диска, отсутствие диска в накопителе

В программе вычисления силы тока исключения могут возникнуть при выполнении преобразования строк, введенных в поля редактирования, в числа и при вычислении величины тока. Исключение EConvertError возникнет, если пользователь неправильно введет числа в поля редактирования: например, разделит целую и дробную части точкой. Исключение EZeroDivide возникнет, если пользователь задаст величину сопротивления равной нулю.

В листинге 2.3 приведена функция обработки события Onclick на командной кнопке **Вычислить.** В функцию включены инструкции обработки исключений.

\ Листинг 2.3. Обработка исключений

```
void __fastcall TForm1 : :Button1Click(TObject *Sender)
{
    float u; // напряжение
    float r; // сопротивление
    float i; // ток

    // получить данные из полей ввода
    // возможно исключение — ошибка преобразования строки в число
    try
    {
    u = StrToFloat(Edit1->Text) ;
    r = StrToFloat(Edit2->Text) ;
    }
    catch (EConvertError &e)
    {
        ShowMessage ("При вводе дробных чисел используйте запятую.");
        return;
    }

    // вычислить ток
    // возможно исключение — деление на ноль
    try
    {
    i = u/r;
    }
    catch (EZeroDivide &e)
    {
        ShowMessage ("Величина сопротивления не должна быть равна нулю") ;
        Edit2->SetFocus (); // курсор в поле Сопротивление
        return;
    }

    // вывести результат в поле метки
    Label4->Caption = "Ток : " + FloatToStrF(i, ffGeneral, 7, 3) ;
}
```

В приведенной функции для вывода сообщений в случае возникновения исключений использована функция ShowMessage, которая выводит на экран окно с текстом и командной кнопкой OK.

Инструкция вызова функции ShowMessage выглядит так:

```
ShowMessage(Сообщение);
```

Где *сообщение* — строковая константа (текст, который надо вывести). На рис. 2.36 приведен вид окна сообщения, полученного в результате выполнения инструкции:

```
ShowMessage("Величина сопротивления не должна быть равна нулю.");
```

Рис. 2.36. Сообщение, выведенное функцией ShowMessage

Следует обратить внимание на то, что в заголовке окна сообщения, выводимого функцией ShowMessage, указывается название приложения. Название приложения задается на вкладке **Application** окна **Project Options.** Если название приложения не задано, то в заголовке будет имя исполняемого файла.

Для вывода сообщений можно использовать функцию MessageDlg. Функция MessageDlg позволяет поместить в окно с сообщением один из стандартных значков, например "Внимание", задать количество и тип командных кнопок и определить, какую из кнопок нажал пользователь. На рис. 2.37 приведено окно, выведенное в результате выполнения инструкции

```
MessageDlg("Файл c:\\temp\\test.txt будет удален.",
           mtWarning, TMsgDlgButtons() << mbOK << mbCancel, 0);
```

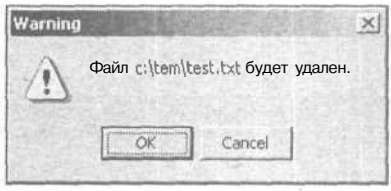


Рис. 2.37. Пример окна сообщения

Значение функции MessageDlg — число, проверив значение которого можно определить, выбором какой командной кнопки был завершен диалог.

В общем виде обращение к функции MessageDlg выглядит так:

```
Выбор:= MessageDlg(Сообщение, Тип, Кнопки, КонтекстСправки)
```

где:

- *сообщение* — текст сообщения;
- *тип* — тип сообщения. Сообщение может быть информационным, предупреждающим или сообщением о критической ошибке. Каждому типу со-

общения соответствует определенный значок. Тип сообщения задается именованной константой (табл. 2.13);

- *кнопки* — кнопки, отображаемые в окне сообщения. Задаются операцией включения в множество элементов — констант (табл. 2.14).
- *КонтекстСправки* — параметр, который определяет раздел справочной информации, который появится на экране, если пользователь нажмет клавишу <F1>. Если вывод справочной информации не предусмотрен, то значение параметра должно быть равно нулю.

Таблица 2.13. *Константы, определяющие тип сообщения*

Константа	Типсообщения	Значок
mtWarning	Внимание	
mtError	Ошибка	
mtInformation	Информация	
mtConfirmation	Подтверждение	
MtCustom	Обычное	Без значка

Таблица 2.14. *Константы, определяющие кнопки в окне сообщения*

Константа	Кнопка	Константа	Кнопка
mbYes	Yes	mbAbort	Abort
mbNo	No	mbRetry	Retry
mbOK	OK	mbIgnore	Ignore
mbCancel	Cancel	mbAll	All
mbHelp	Help		

Кроме приведенных констант можно использовать константы mbOkCancel, mbYesNoCancel и mbAbortRetryIgnore. Эти константы определяют наиболее часто используемые в диалоговых окнах комбинации командных кнопок.

Значение, возвращаемое функцией MessageDlg (табл. 2.15), позволяет определить, какая из командных кнопок была нажата пользователем.

Таблица 2.15. Значения функции MessageDig

Значение функции MessageDig	Диалог завершен нажатием кнопки
mrAbort	Abort
mrYes	Yes
mrOk	Ok
mrRetry	Retry
mrNo	No
mrCancel	Cancel
mrIgnore	Ignore
mrAll	All

Внесение изменений

После нескольких запусков программы "Сила тока" возникает желание усовершенствовать программу, внести в нее изменения. Например, такие, чтобы после ввода напряжения в результате нажатия клавиши <Enter> курсор переходил в поле **Сопротивление,** а после ввода сопротивления в результате нажатия этой же клавиши выполнялся расчет. Кроме того, было бы неплохо, чтобы пользователь мог вводить в поля редактирования только числа.

Чтобы внести изменения в программу, нужно запустить C++ Builder и открыть соответствующий проект. Сделать это можно обычным способом, выбрав в меню **File** команду **Open Project.** Можно также воспользоваться командой **Reopen** из меню **File.** При выборе команды **Reopen** открывается список проектов, над которыми работал программист в последнее время.

В листинге 2.4 приведена программа "Сила тока", в которую внесены изменения: добавлены функции обработки событий onKeypress для компонентов Edit1 и Edit2. Чтобы добавить в программу функцию обработки события, надо в окне **Object Inspector** выбрать компонент, для которого нужно создать функцию обработки события, на вкладке **Events** выбрать событие и сделать двойной щелчок в поле рядом с именем события. C++ Builder сформирует шаблон функции обработки события. После этого можно вводить инструкции, реализующие функцию.

Листинг 2.4. Функции обработки событий на компонентах формы программы "Сила тока"

```
// щелчок на кнопке Вычислить
void__fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
    float u; // напряжение
    float r; // сопротивление
    float i; // ток

    // проверим, введены ли данные в поля Напряжение и Сопротивление
    if ( ( (Edit1->Text).Length() == 0) || ( (Edit2->Text).Length() == 0) )
    {
        MessageDlg ("Надо ввести напряжение и сопротивление",
                    mtInformation, TMsgDlgButtons() << mbOK, 0) ;
        if ( (Edit1->Text).Length() == 0)
            Edit1->SetFocus(); // курсор в поле Напряжение
        else
            Edit2->SetFocus(); // курсор в поле Сопротивление
        return;
    };

    // получить данные из полей ввода
    u = StrToFloat (Edit1->Text);
    r = StrToFloat (Edit2->Text);

    // вычислить силу тока
    try
    {
    i = u/r;
    }
    catch (EZeroDivide &e)
    {
        ShowMessage ("Величина сопротивления не должна быть равна нулю") ;
        Edit2->SetFocus ( ) ;   // курсор в поле Сопротивление
        return;
    }

    // вывести результат в поле Label4
   Label4->Caption = "Ток : " +
                     FloatToStrF(i,ffGeneral,7,2) + " A";
}

// нажатие клавиши в поле Напряжение
void __fastcall TForm1 : :Edit1KeyPress(TObject *Sender, char &Key)
{
    // коды запрещенных клавиш заменим нулем, в результате
    // символы этих клавиш в поле редактирования не появятся

    // Key — код нажатой клавиши
    // проверим, является ли символ допустимым
```

```
    if ( ( Key >= '0') && ( Key <= '9')) // цифра
        return;

    // Глобальная переменная Decimalseparator
    // содержит символ, используемый в качестве разделителя
    // при записи дробных чисел
    if ( Key == Decimalseparator)
    {
       if ( (Edit1->Text).Pos(DecimalSeparator) != 0)
           Key = 0; // разделитель уже введен
    return;
    }

    if (Key == VK_BACK) // клавиша <Backspace>
        return;

    if ( Key == VK_RETURN) // клавиша <Enter>
    {
       Edit2->SetFocus();
       return;
    };

    // остальные клавиши запрещены
    Key = 0; // не отображать символ
}

// нажатие клавиши в поле Сопротивление
void _ fastcall TForm1 : :Edit2KeyDown (TObject *Sender, WORD &Key,
      TShiftState Shift)
{
    if ( ( Key >= '0') && ( Key <= '9')) // цифра
        return;

    if ( Key == Decimalseparator) {
       if ( (Edit2->Text).Pos(DecimalSeparator) != 0)
           Key = 0; // разделитель уже введен
       return;
    }

    if (Key == VK_BACK) // клавиша <Backspace>
        return;

    if ( Key == VK_RETURN) // клавиша <Enter>
    {
       Button1->SetFocus ( ) ; // переход к кнопке Вычислить
                               // повторное нажатие клавиши <Enter>
                               // активизирует процесс вычисления тока
```

```
        return;
    };

    // остальные клавиши запрещены
    Key = 0; // не отображать символ
}

// щелчок на кнопке Завершить
void _ fastcall TForm1: :Button2Click(TObject *Sender)
{
    Form1->Close ( ) ; // закрыть форму приложения
}
```

Настройка приложения

После того как программа отлажена, необходимо выполнить ее окончательную настройку: задать название программы и значок, который будет изображать исполняемый файл приложения в папке, на рабочем столе и на панели задач, во время работы программы.

Название программы

Название программы отображается во время ее работы в панели задач Windows, а также в заголовках окон сообщений, выводимых функцией `ShowMessage`.

Название программы надо ввести в поле **Title** (рис. 2.38) вкладки **Application** диалогового окна **Project Options,** которое появляется в результате выбора в меню **Project** команды **Options.**

Значок приложения

Чтобы назначить приложению значок, отличный от стандартного, нужно в меню **Project** выбрать команду **Options** и в открывшемся окне на вкладке **Application** щелкнуть на кнопке **Load Icon.** В результате этих действий откроется стандартное окно, используя которое можно просмотреть каталоги и найти подходящий значок (значки хранятся в файлах с расширением ico).

В состав C++ Builder входит утилита Image Editor (Редактор изображений), при помощи которой программист может создать для своего приложения уникальный значок. Запустить Image Editor можно из C++ Builder, выбрав в меню **Tools** команду **Image Editor,** или из Windows — командой **Пуск | Программы | Borland C++ Builder | Image Editor.**

Чтобы начать работу по созданию нового значка, нужно в меню **File** выбрать команду New | **Icon File** (рис. 2.39).

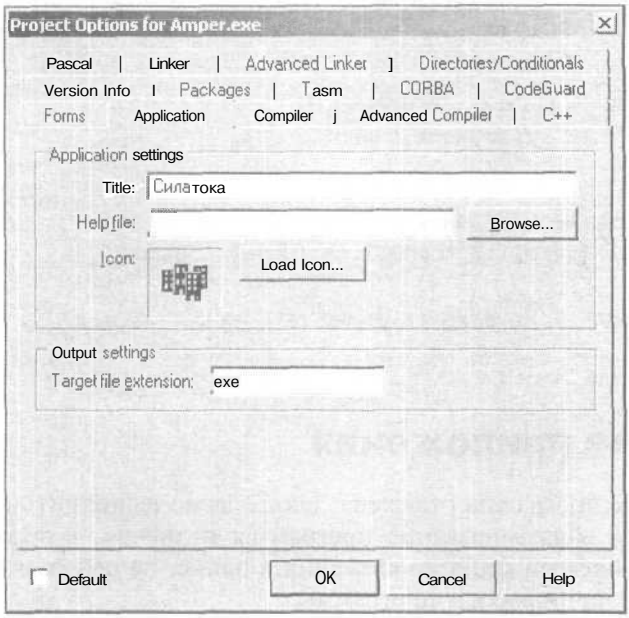


Рис. 2.38. Название программы надо ввести в поле **Title**

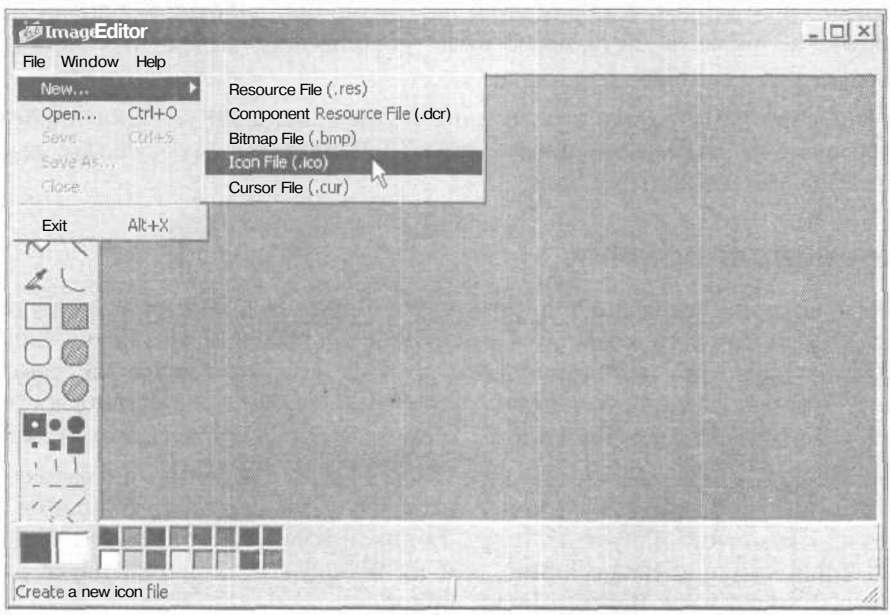


Рис. 2.39. Начало работы над новым значком

После выбора типа создаваемого файла открывается окно **Icon Properties** (рис. 2.40), в котором необходимо выбрать характеристики создаваемого значка: **Size** (Размер) -- 32x32 (стандартный размер значков Windows) и **Colors** (Палитра) — 16 цветов. В результате нажатия кнопки **OK** открывается окно **Iconl.ico** (рис. 2.41), в котором можно, используя стандартные инструменты и палитру, нарисовать нужный значок.

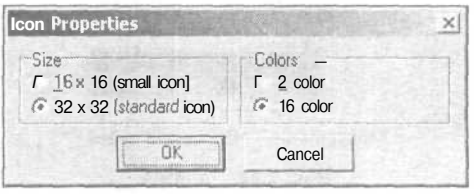


Рис. 2.40. Стандартные характеристики значка

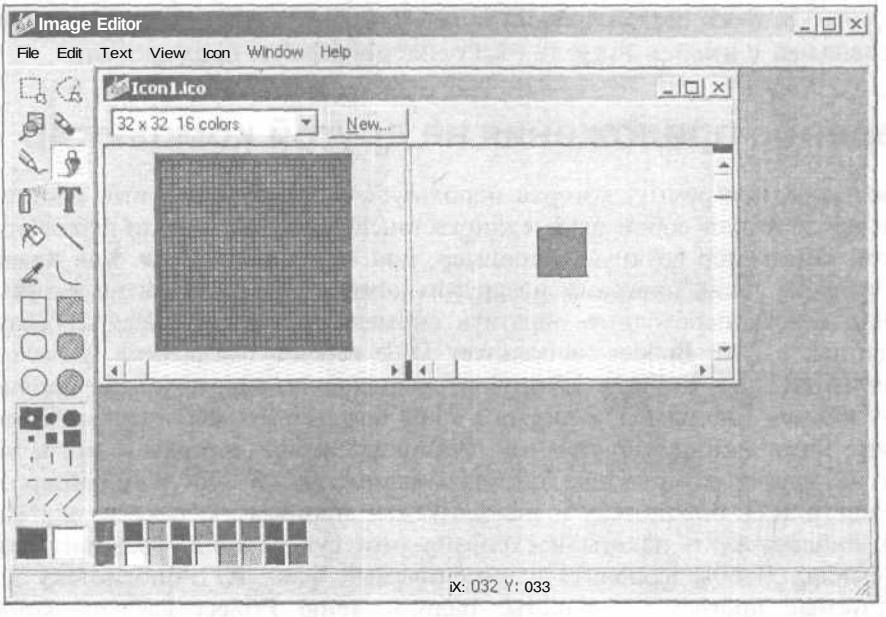


Рис. 2.41. Начало работы над новым значком

Процесс рисования в Image Editor практически ничем не отличается от процесса создания картинки в обычном графическом редакторе, например в Microsoft Paint. Однако есть одна тонкость. Первоначально поле изображения закрашено "прозрачным" цветом. Если значок нарисовать на этом фоне, то при отображении значка части поля изображения, закрашенные "прозрачным" цветом, примут цвет фона, на котором будет находиться значок.

В процессе создания картинки можно удалить (стереть) ошибочно нарисованные элементы, закрасив их прозрачным цветом, которому на палитре соответствует левый квадрат в нижнем ряду (рис. 2.42).

Кроме "прозрачного" цвета в палитре есть "инверсный" цвет. Нарисованные этим цветом части рисунка при выводе на экран будут окрашены инверсным относительно цвета фона цветом.

Рис. 2.42. Палитра

Чтобы сохранить нарисованный значок, надо в меню **File** выбрать команду **Save,** в открывшемся диалоговом окне раскрыть папку проекта (приложения, для которого создан значок) и задать имя файла значка, которое обычно совпадает с именем проекта (выполняемого файла приложения).

Перенос приложения на другой компьютер

Небольшую программу, которая использует только стандартные компоненты и представляет собой один-единственный exe-файл, можно перенести на другой компьютер вручную, например, при помощи дискеты. Как правило, при запуске таких программ на другом компьютере проблем не возникает. Вместе с тем, необходимо обратить внимание на следующее. Программа, созданная в C++ Builder, использует DLL версию *библиотеки времени выполнения* (RTL — Runtime Library) и специальные динамические библиотеки — *пакеты* (например, в пакете VCL60 находятся наиболее часто используемые компоненты и системные функции). Чтобы программа могла работать на другом компьютере, помимо exe-файла на этот компьютер надо перенести RTL-библиотеку и используемые программой пакеты или включить библиотеку и пакеты в exe-файл (что существенно увеличит размер exe-файла). Чтобы включить в выполняемый файл RTL-библиотеку и используемые программой пакеты, надо в меню **Project** выбрать команду **Options** и во вкладках **Linker** (рис. 2.43) и **Packages** (рис. 2.44) сбросить соответственно флажки **Use dynamic RTL и Build with runtime packages.** После этого нужно выполнить перекомпоновку программы.

Сложные программы, например те, которые используют компоненты доступа к базам данных, перенести на другой компьютер вручную проблематично. Для таких программ лучше создать установочную дискету (CD-ROM). Сделать это можно, например, при помощи пакета InstallShield Express, который входит в комплект поставки C++ Builder.

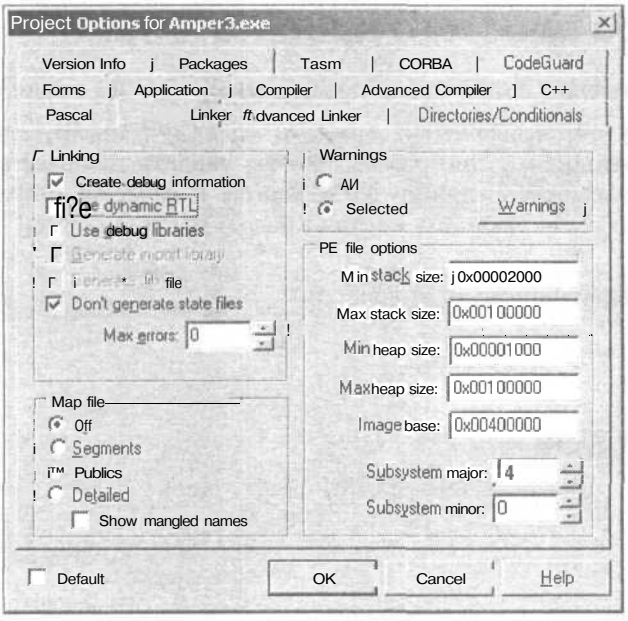


Рис. 2.43. Чтобы включить в выполняемый файл RTL-библиотеку, сбросьте флажок **Use dynamic RTL**

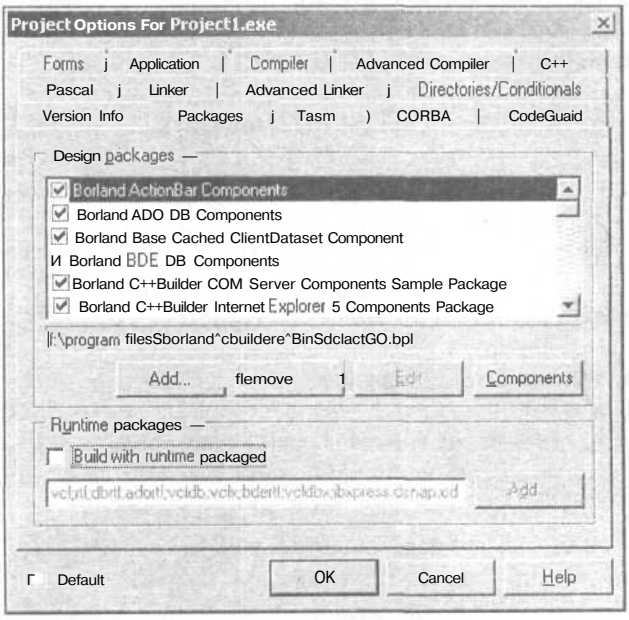


Рис. 2.44. Чтобы включить в выполняемый файл используемые программой пакеты (специальные DLL-библиотеки), сбросьте флажок **Build with runtime packages**

Структура простого проекта

Проект представляет собой набор программных единиц — модулей.

Один из модулей, называемый *главным,* содержит инструкции, с которых начинается выполнение программы. Чтобы увидеть главный модуль, нужно в меню **Project** выбрать команду **View Source.** В качестве примера в листинге 2.5 приведен текст главного модуля программы "Сила тока".

Листинг 2.5. Главный модуль (Amper.cpp)

```
#include <vcl.h>
#pragma hdrstop

USEFORM("Amper_1.cpp", Form1);

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->Title = "Сила тока";
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
        }
    }
    return 0;
}
```

Начинается главный модуль директивами компилятору (точнее, препроцессору). Директива `#include <vcl.h>` информирует компилятор, что перед тем

как приступить непосредственно к компиляции, в текст главного модуля нужно включить заголовочный файл библиотеки визуальных компонентов — vcl.h. Строка USEFORM("Amper_1.cpp", Form1) указывает, что в проект нужно включить файл модуля формы Amper_1.cpp, который содержит функции обработки событий для формы Form1. Далее следует описание главной функции программы — WinMain. Функция WinMain инициализирует внутренние структуры программы, создает форму Forml и запускает программу, что приводит к появлению на экране стартовой формы. Так как в проекте "Сила тока" только одна форма, то на экране именно она и появляется. Инструкция обработки исключений catch выполняется, если в программе возникает ошибка. Таким образом, главный модуль обеспечивает вывод стартовой формы программы, дальнейшее поведение которой определяют функции обработки событий стартовой формы.

Помимо главного модуля в состав проекта входят *модули формы*. Для каждой формы C++ Builder создает отдельный модуль, который состоит из двух файлов: *заголовочного файла* и *файла кода* (содержимое этих файлов отражается в окне редактора кода). Заголовочный файл содержит описание формы (листинг 2.6), файл кода (модуль формы) - - описание (текст) функций, в том числе и обработки событий (листинг 2.7).

Листинг 2.6. Заголовочный файл модуля формы (Amper_1.h)

```
#ifndef Amper_1H
ftdefineAmper_1H

#include <Classes.hpp>
#include <Controls.hpp>
ttinclude<StdCtrls.hpp>
#include <Forms.hpp>

class TForm1 : public TForm
{
__published:  // IDE-managed Components
    TLabel *Label1;
    TLabel *Label2;
    TLabel *Label3;
    TEdit *Edit1;
    TEdit *Edit2;
    TButton *Button1;
    TButton *Button2;
    TLabel *Label4;
    void__fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
```

```
private:        // User declarations
public:         // User declarations
    __fastcall TForm1(TComponent* Owner);
};

extern PACKAGE TForm1 *Form1;
#endif
```

Г Листинг 2.7. Модуль формы (Amper_1.cpp)

```
#include <vcl.h>
#pragma hdrstop

iinclude "Amper_1.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent*Owner)
    : TForm(Owner)
{
}

// щелчок на кнопке Вычислить
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    float u; // напряжение
    float r; // сопротивление
    float i; // сила тока

    // получить данные из полей ввода
    // возможно исключение
    try
    {
    u = StrToFloat(Edit1->Text);
    r = StrToFloat(Edit2->Text);
    }
    catch (EConvertError se)
    {
        ShowMessage("При вводе дробных чисел используйте запятую.");
        return;
    }
```

```
    // вычислить силу тока
    // возможно исключение
    try
    {
    i = u/r;
    }
    catch (EZeroDivide &e)
    {
        ShowMessage ("Сопротивление не должно быть равно нулю");
        Edit1->SetFocus ();   // курсор в поле Сопротивление
        return;
    }

    // вывести результат в поле метки
    Label4->Caption = "Ток : " +
         FloatToStrF(i,ffGeneral,7,3);

}

// щелчок на кнопке Завершить
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Form1->Close(); // закрыть окно программы
}
```

Следует отметить, что значительное количество работы по генерации программного кода выполнил C++ Builder. Он полностью сформировал главный модуль (Amper.cpp), заголовочный файл модуля формы (Amper_1.h), значительную часть модуля формы (Amper_1.cpp). Кроме того, C++ Builder, анализируя действия программиста, сформировал описание формы, файл проекта и файл ресурсов проекта.

часть II

Практикум программирования

Во второй части книги демонстрируется назначение и возможности базовых компонентов, на конкретных примерах показано, как, используя эти базовые компоненты, создать программу, обеспечивающую отображение графики, воспроизведение звука и анимации. Уделено внимание разработке приложений работы с базами данных. Показано, как создать справочную систему и установочный CD.

ГЛАВА 3

Графика

C++ Builder позволяет программисту разрабатывать программы, которые работают с графикой. В этой главе рассказывается, что надо сделать, чтобы на поверхности формы появилась картинка, сформированная из графических примитивов, или иллюстрация, созданная в графическом редакторе или полученная в результате сканирования фотографии.

Холст

Программа может вывести графику на поверхность формы (или компонента image), которой соответствует свойство canvas (Canvas — холст для рисования). Для того чтобы на поверхности формы или компонента image появилась линия, окружность, прямоугольник или другой графический элемент *(примитив),* необходимо к свойству canvas применить соответствующий метод (табл. 3.1).

Например, оператор

```
Form1->Canvas->Rectangle(10,10,50,50);
```

рисует на поверхности формы прямоугольник.

Таблица 3.1. *Методы вычерчивания графических примитивов*

Метод	Действие
`LineTo(x,y)`	Рисует линию из текущей точки в точку с указанными координатами
`Rectangle(x1,y1,x2,y2)`	Рисует прямоугольник. `x1, y1` и x2, y2 — координаты левого верхнего и правого нижнего углов прямоугольника. Цвет границы и внутренней области прямоугольника могут быть разными

Таблица 3.1 *(окончание)*

Метод	Действие
`FillRect(x1,y1,x2,y2)`	Рисует закрашенный прямоугольник. xl, yl, x2, y2 — определяют координаты диагональных углов
`FrameRect(x1,y1,x2,y2)`	Рисует контур прямоугольника. xl, yl, x2, y2 — определяют координаты диагональных углов
`RounRect(x1,y1,x2,y2,x3,y3)`	Рисует прямоугольник со скругленными углами
`Ellipse(x1,y1,x2,y2)`	Рисует эллипс или окружность (круг). xl, yl, x2, y2 — координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, окружность
`Polyline(points,n)`	Рисует ломаную линию. *points* — массив типа `TPoint`. Каждый элемент массива представляет собой запись, поля x и y которой содержат координаты точки перегиба ломаной; n — количество звеньев ломаной. Метод **Polyline** вычерчивает ломаную линию, последовательно соединяя прямыми отрезками точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д.

Методы вывода графических примитивов рассматривают свойство canvas как некоторый абстрактный холст, на котором они могут *рисовать* (Canvas переводится как "поверхность", "холст для рисования"). Холст состоит из отдельных точек — пикселов. Положение пиксела на поверхности холста характеризуется горизонтальной (X) и вертикальной (Y) координатами. Координаты возрастают сверху вниз и слева направо (рис. 3.1). Левый верхний пиксел поверхности формы (клиентской области) имеет координаты (О, О), правый нижний — (`ClientWidth`, `ClientHeight`). Доступ к отдельному пикселу осуществляется через свойство Pixels, представляющее собой двумерный массив, элементы которого содержат информацию о цвете точек холста.

Следует обратить внимание на важный момент. Изображение, сформированное на поверхности формы, может быть испорчено, например, в результате полного или частичного перекрытия окна программы другим окном. Поэтому программист должен позаботиться о том, чтобы в момент появления окна программа перерисовала испорченное изображение. К счастью, операционная система Windows информирует программу о необходимости перерисовки окна, посылая ей соответствующее сообщение, в результате чего возникает событие OnPaint. Событие `OnPaint` возникает и в момент за-

пуска программы, когда окно появляется на экране в первый раз. Таким образом, инструкции, обеспечивающие вывод графики на поверхность формы, надо поместить в функцию обработки события onPaint.

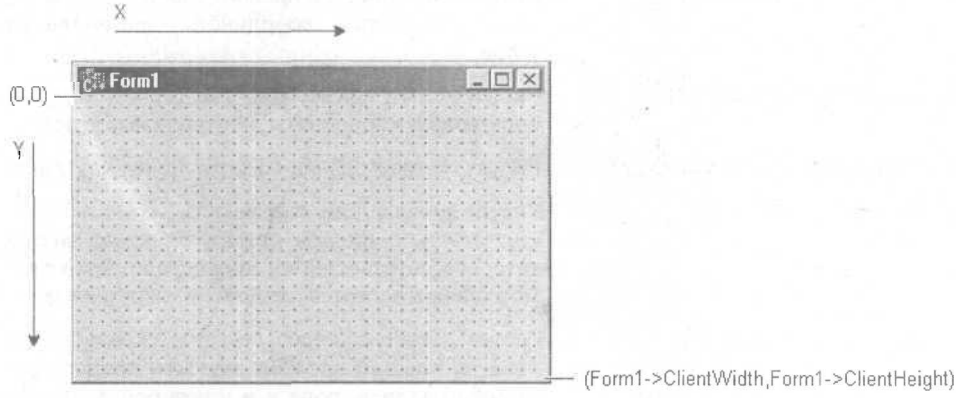


Рис. 3.1. Координаты точек поверхности формы (холста)

Карандаш и кисть

Методы вычерчивания графических примитивов обеспечивают только вычерчивание. Вид графического элемента определяют свойства Реп (карандаш) и Brush (кисть) той поверхности (Canvas), на которой рисует метод.

Карандаш и кисть, являясь свойствами объекта canvas, в свою очередь представляют собой объекты Реп и Brush. Свойства объекта Реп (табл. 3.2) задают цвет, толщину и тип линии или границы геометрической фигуры. Свойства объекта Brush (табл. 3.3) задают цвет и способ закраски области внутри прямоугольника, круга, сектора или замкнутого контура.

Таблица 3.2. *Свойства объекта Реп (карандаш)*

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

Таблица 3.3. *Свойства объекта Brush (кисть)*

Свойство	Определяет
Color	Цвет закрашивания замкнутой области
style	Стиль заполнения области (bsSolid — сплошная запивка. Штриховка: bsHorizontal — горизонтальная; bsVertical — вертикальная; bsFDiagonal — диагональная с наклоном линий вперед; bsBDlagonal — диагональная с наклоном линий назад; bsCross — в клетку; bsDiagCross — диагональная клетка

Ниже приведена функция обработки события onPain, которая рисует на поверхности формы олимпийский флаг.

```
void _ fastcall TForm1: :FormPaint (TObject *Sender)
{
    // полотнище флага
    Canvas->Pen->Width = 1;
    Canvas->Pen->Color = clBlack;
    Canvas->Brush->Color = clCream;
    Canvas->Rectangle(30,30,150,150);

    Canvas->Pen->Width = 2; // ширина колец
    Canvas->Brush->Style = bsClear; // чтобы круг, нарисованный
                                    // методом Ellipse, не был закрашен
    // рисуем кольца
    Canvas->Pen->Color = clBlue;
    Canvas->Ellipse(40,40,80,80);

    Canvas->Pen->Color = clBlack;
    Canvas->Ellipse(70,40,110,80);

    Canvas->Pen->Color = clRed;
    Canvas->Ellipse(100,40,140,80);

    Canvas->Pen->Color = clYellow;
    Canvas->Ellipse(55,65,95,105);

    Canvas->Pen->Color = clGreen;
    Canvas->Ellipse(85,65,125,105);
}
```

Графические примитивы

Любая картинка, чертеж или схема могут рассматриваться как совокупность графических *примитивов:* точек, линий, окружностей, дуг и др. Таким образом, для того чтобы на экране появилась нужная картинка, программа

должна обеспечить вычерчивание (вывод) графических элементов — *примитивов,* составляющих эту картинку.

Вычерчивание графических примитивов на поверхности (формы или компонента image — области вывода иллюстрации) осуществляется применением соответствующих методов к свойству Canvas этой поверхности.

Линия

Вычерчивание прямой линии выполняет метод LineTo. Метод рисует линию из той точки, в которой в данный момент находится карандаш (эта точка называется текущей позицией карандаша или просто "текущей"), в точку, координаты которой указаны в инструкции вызова метода.

Например, оператор

```
Canvas->LineTo(100,200)
```

рисует линию в точку с координатами (100, 200), после чего текущей становится точка с координатами (100, 200).

Начальную точку линии можно задать, переместив карандаш в нужную точку графической поверхности. Сделать это можно при помощи метода MoveTo, указав в качестве параметров координаты точки начала линии. Например, операторы

```
Canvas->MoveTo(10,10); // установить карандаш в точку (10,10)
Canvas->LineTo(50,10); // линия из точки (10,10) в точку (50,10)
```

рисуют горизонтальную линию из точки (10, 10) в точку (50, 10).

Используя свойство текущей точки, можно нарисовать ломаную линию. Например, операторы

```
Canvas->MoveTo(10,10);
Canvas->LineTo(50,10);
Canvas->LineTo(10,20);
Canvas->LineTo(50,20);
```

рисуют линию, похожую на букву Z.

Ломаная линия

Метод Polyline вычерчивает ломаную линию. В качестве параметров методу передается массив типа TPoint, содержащий координаты узловых точек линии, и количество звеньев 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.

Например, приведенный ниже фрагмент кода рисует ломаную линию, состоящую из трех звеньев.

```
TPoint p[4]; // координаты начала, конца и точек перегиба

// задать координаты точек ломаной
p[0].x = 100; p[0].y = 100;   // начало
p[1].x = 100; p[1].y = 150;   // точка перегиба
p[2].x = 150; p[2].y = 150;   // точка перегиба
p[3].x = 150; p[3].y = 100;   // конец

Canvas->Polyline(p,3); // ломаная из трех звеньев
```

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. В инструкции вызова метода надо указать координаты двух точек — углов прямоугольника. Например, оператор

```
Canvas->Rectangle(10,10,50,50)
```

рисует квадрат, левый верхний угол которого находится в точке (10, 10), а правый нижний в точке (50, 50).

Цвет, вид и ширину линии контура прямоугольника определяют значения свойства Pen, а цвет и стиль заливки области внутри прямоугольника — значения свойства Brush той поверхности, на которой метод рисует прямоугольник. Например, следующие операторы рисуют флаг Российской Федерации.

```
Canvas->Brush->Color = clWhite; // цвет кисти — белый
Canvas->Rectangle(10,10,90,30);
Canvas->Brush->Color = clBlue; // цвет кисти — синий
Canvas->Rectangle(10,30,90,50);
Canvas->Brush->Color = clRed; // цвет кисти — красный
Canvas->Rectangle(10,50,90,70);
```

Вместо четырех параметров — координат двух диагональных углов прямоугольника — методу Rectangle можно передать один параметр — структуру типа TRect, поля которой определяют положение диагональных углов прямоугольной области. Следующий фрагмент кода демонстрирует использование структуры TRect в качестве параметра метода Rectangle.

```
TRect rct; // прямоугольная область
rct.Top  = 10;
rct.Left = 10;
rct.Bottom = 50;
rct.Right = 50;
Canvas->Rectangle(rct); // нарисовать прямоугольник
```

Есть еще два метода, которые вычерчивают прямоугольник. Метод FillRect вычерчивает закрашенный прямоугольник, используя в качестве инструмента только кисть (Brush), а метод FrameRect — только контур и использует только карандаш (Pen). У этих методов только один параметр — структура типа TRect. Поля структуры TRect содержат координаты прямоугольной области. Значения полей структуры TRect можно задать при помощи функции Rect.

Например:

```
TRect rct; // область, которую надо закрасить
rct = Rect(10,10,30,50);          // координаты области
Canvas->Brush->Color = clRed; // цвет закраски
Canvas->FillRect (rct);
```

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

```
Canvas->RoundRec(x1,y1,x2,y2,x3,y3)
```

Параметры xl, yl, x2, y2 определяют положение углов прямоугольника, а параметры x3 и y3 — размер эллипса, одна четверть которого используется для вычерчивания скругленного угла (рис. 3.2).

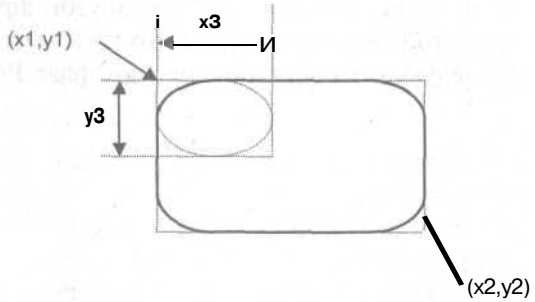


Рис. 3.2. Метод RoundRec вычерчивает прямоугольник со скругленными углами

Многоугольник

Метод Polygon вычерчивает многоугольник. Инструкция вызова метода в общем виде выглядит так:

```
Canvas->Polygon(p,п)
```

где p — массив записей типа TPoint, который содержит координаты вершин многоугольника; n — количество вершин.

Метод Polygon чертит многоугольник, соединяя прямыми линиями точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Вид границы многоугольника определяют значения свойства Реп, а вид заливки области, ограниченной линией границы, — значения свойства Brush той поверхности, на которой метод рисует.

Ниже приведен фрагмент кода, который, используя метод Polygon, рисует ромб,

```
TPoint p[4]; // четыре вершины

// координаты вершин
p[0].x =  50; p[0].y = 100;
p[1].x = 150; p[1].y =  75;
p[2].x = 250; p[2].y = 100;
p[3].x = 150; p[3].y = 125;

Canvas->Brush->Color = clRed;
Canvas->Polygon(p,3);
```

Окружность и эллипс

Нарисовать эллипс или окружность (частный случай эллипса) можно при помощи метода Ellipse. Инструкция вызова метода в общем виде выглядит следующим образом:

```
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. 3.3).

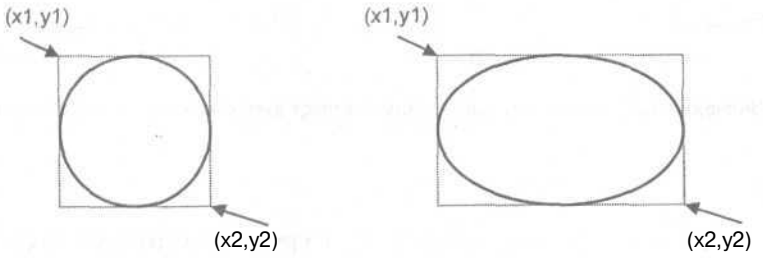


Рис. 3.3. Значения параметров метода Ellipse определяют вид геометрической фигуры

Вместо четырех параметров — координат диагональных углов прямоугольника — методу Ellipse можно передать один — объект типа TRect. Следующий фрагмент кода демонстрирует использование объекта TRect в качестве параметра метода Ellipse.

```
TRect rec = Rect(10,10,50,50);
Canvas->Ellipse(rec);
```

Как и в случае вычерчивания других примитивов, вид контура эллипса (цвет, толщину и стиль линии) определяют значения свойства Реп, а цвет и стиль заливки области внутри эллипса — значения свойства Brush той поверхности (canvas), на которой метод чертит.

Дуга

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

```
Canvas->Arc(x1,y1,x2,y2,x3,y3,x4,y4)
```

Параметры x1, y1, x2, y2 определяют эллипс (окружность), частью которого является дуга. Параметры x3 и y3 задают 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. 3.4).

Цвет, толщина и стиль линии, которой вычерчивается дуга, определяются значениями свойства Реп поверхности (canvas), на которую выполняется вывод.

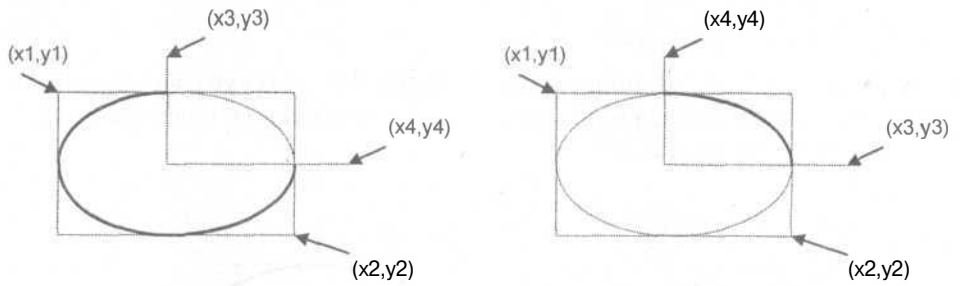


Рис. 3.4. Значения параметров метода Arc определяют дугу как часть эллипса (окружности)

Сектор

Метод pie вычерчивает сектор эллипса или круга. Инструкция вызова метода в общем виде выглядит следующим образом:

```
Canvas->Pie(x1,y1,x2,y2,x3,y3,x4,y4)
```

Параметры x1, y1, x2, y2 определяют эллипс (круг), частью которого является сектор; x3, уз, 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, уз), к прямой, заданной точкой с координатами (x4, y4) (рис. 3.5).

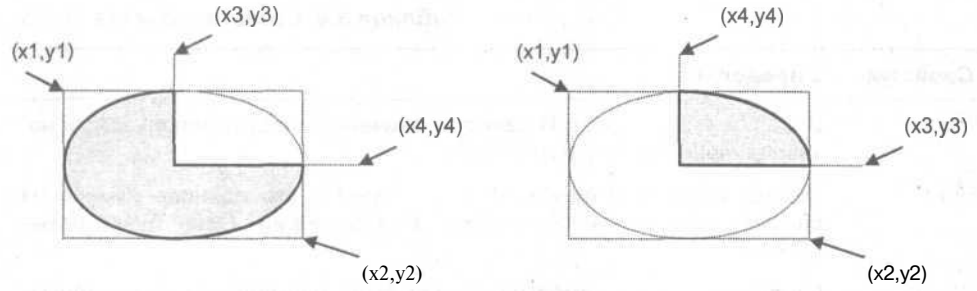


Рис. 3.5. Значения параметров метода Pie определяют сектор как часть эллипса (окружности)

Текст

Вывод текста (строк типа AnsiString) на поверхность графического объекта обеспечивает метод TextoutA. Инструкция вызова метода TextoutA в общем виде выглядит следующим образом:

```
Canvas->TextOutA(x,y,Текст)
```

Параметр текст задает выводимый текст. Параметры x и y определяют координаты точки графической поверхности, от которой выполняется вывод текста (рис. 3.6).

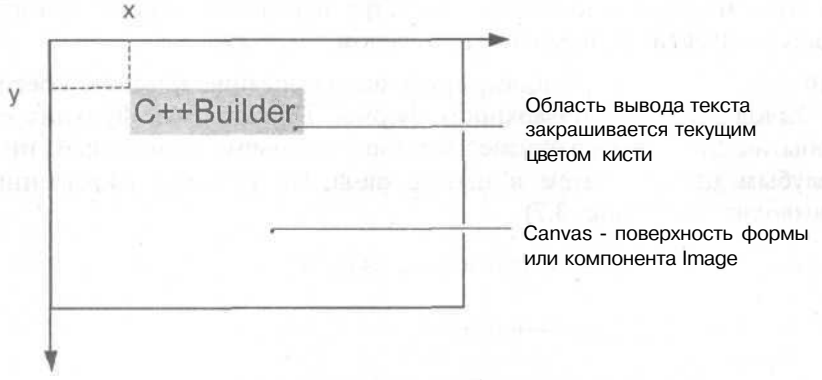


Рис. 3.6. Координаты области вывода текста

Шрифт, который используется для вывода текста, определяется значением свойства Font соответствующего объекта canvas. Свойство Font представляет собой объект типа TFont. В табл. 3.4 перечислены свойства объекта TFont, определяющие характеристики шрифта, используемого методом TextOutA для вывода текста.

Таблица 3.4. *Свойства объекта TFont*

Свойство	Определяет
Name	Используемый шрифт. В качестве значения следует использовать название шрифта (например, Arial)
size	Размер шрифта в пунктах (points). Пункт— это единица измерения размера шрифта, используемая в полиграфии. Один пункт равен 1/72 дюйма
style	Стиль начертания символов. Может быть: нормальным, полужирным, курсивным, подчеркнутым, перечеркнутым. Стиль задается при помощи следующих 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: `Canvas->Font->Style = TFontStyles () <<fsBold<<fsUnderline`
Color	Цвет символов. В качестве значения можно использовать константу типа TColor

При выводе текста весьма полезны методы TextWidth и TextHeight, значениями которых являются соответственно ширина и высота области вывода текста, которые, очевидно, зависят от характеристик используемого шрифта. Обоим этим методам в качестве параметра передается строка, которую предполагается вывести на поверхность методом TextOutA.

Следующий фрагмент кода демонстрирует использование методов, обеспечивающих вывод текста на поверхность формы. Приведенная функция обработки события OnPaint закрашивает верхнюю половину окна белым, нижнюю — голубым цветом, затем в центре окна, по границе закрашенных областей, выводит текст (рис. 3.7).

```
void__fastcall TForm1::FormPaint(TObject *Sender)
{
    AnsiString ms = "Borland C++Builder";
    TRect aRect;
    int x,y; // точка, от которой будет выведен текст
```

```
    // верхнюю половину окна красим белым
    aRect = Rect(0,0,ClientWidth,ClientHeight/2);
    Canvas->Brush->Color = clWhite;
    Canvas->FillRect(aRect);

    // нижнюю половину окна красим голубым
    aRect = Rect(0,ClientHeight/2,ClientWidth,ClientHeight);
    Canvas->Brush->Color = clSkyBlue;
    Canvas->FillRect(aRect);


    Canvas->Font->Name = "Times New Roman";
    Canvas->Font->Size = 24;
    // Canvas->Font->Style = TFontStyles()<< fsBold << fsItalic;

    // текст разместим в центре окна
    x = (ClientWidth - Canvas->TextWidth(ms)) /2;
    y = ClientHeight/2 - Canvas->TextHeight(ms) /2;
    Canvas->Brush->Style = bsClear; // область вывода текста
                                    // не закрашивать
    Canvas->Font->Color = clBlack;
    Canvas->TextOutA(x,y,ms); // вывести текст
}
```

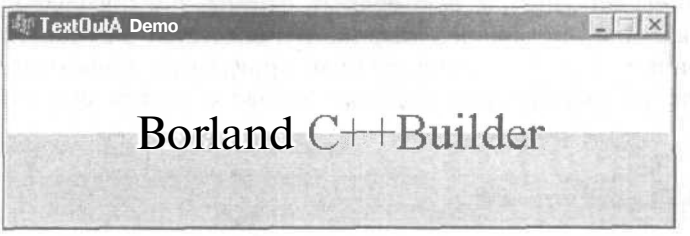


Рис. 3.7. Вывод текста

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

Следующий фрагмент кода демонстрирует возможность вывода строки текста ПрИ ПОМОЩИ ДВУХ ИНСТРУКЦИЙ TextOutA:

```
Canvas->TextOutA(10,10,"Borland ");
Canvas->TextOutA(Canvas->PenPos.x, Canvas->PenPos.y, "C++Builder");
```

Точка

Поверхности, на которую программа может осуществлять вывод графики, соответствует объект canvas. Свойство pixels, представляющее собой двумерный массив типа TColor, содержит информацию о цвете каждой точки графической поверхности. Используя свойство pixels, можно задать цвет любой точки графической поверхности, т. е. "нарисовать" точку. Например, инструкция

```
Canvas->Pixels[10][10] = clRed
```

окрашивает точку поверхности формы в красный цвет.

Размерность массива Pixels определяется реальным размером графической поверхности. Размер графической поверхности формы (рабочей области, которую также называют *клиентской)* определяют свойства `ClientWidth` и `ClientHeight`, а размер графической поверхности компонента image – свойства `Width` и Height. Левой верхней точке рабочей области формы соответствует элемент `pixels[0][0]`, а правой нижней – `Pixels[ClientWidth- 1][ClientHeight -1]`.

Следующая программа (листинг 3.1), используя свойство pixels, строит график функции $y = 2\sin(x)e^{x/5}$. Границы диапазона изменения аргумента функции являются исходными данными. Диапазон изменения значения функции вычисляется во время работы программы. На основании этих данных программа вычисляет масштаб, позволяющий построить график таким образом, чтобы он занимал всю область формы, предназначенную для вывода графика. Для построения графика используется вся доступная область формы, причем если во время работы программы пользователь изменит размер окна, то график будет выведен заново с учетом реальных размеров окна.

Листинг 3.1. График функции

```
// обработка события OnPaint
void __fastcall TForm1::FormPaint(TObject *Sender)\
{
    Grafik();
}

// обработка события OnResize
void __fastcall TForm1::FormResize(TObject *Sender)
{
    TRect rct = Rect(0,0,ClientWidth,ClientHeight);

    Canvas->FillRect(rct);  // стереть
    Grafik();
}
```

```
#include "math.h" // для доступа к sin и exp

// функция, график которой надо построить
float f(float x)
{
   return 2*sin(x)*exp(x/5);
}

void TForm1::Grafik()
{
 float x1, x2;   // границы изменения аргумента функции
 float y1, y2;   // границы изменения значения функции
 float x;        // аргумент функции
 float y;        // значение функции в точке x
 float dx;       // приращение аргумента
 int l, b;       // левый нижний угол области вывода графика
 int w, h;       // ширина и высота области вывода графика
 float mx, my;   // масштаб по осям X и Y
 int x0, y0;     // начало координат

 // область вывода графика
 l = 10;                          // X — координата левого верхнего угла
 b = Form1->ClientHeight-20;      // Y — координата левого нижнего угла
 h = Form1->ClientHeight-40;      // высота
 w = Form1->Width — 20;           // ширина


 x1 = 0;      // нижняя граница диапазона аргумента
 x2 = 25;     // верхняя граница диапазона аргумента
 dx = 0.01;   // шаг аргумента

 // найдем максимальное и минимальное значение
 // функции на отрезке [x1,x2]
 x = x1;
 y1 = f(x); // минимум
 y2 = f(x); // максимум
 do {
   y = f(x);
   if ( y < y1) y1 = y;
   if ( y > y2) y2 = y;
   x += dx;
 } while (x <= x2);

 // вычислим масштаб
 ту = (float)h/abs(y2-y1); // масштаб по оси Y
 mx = w/abs(x2-x1);        // масштаб по оси X
```

```
// оси
x0 = l+abs(x1*mx);
y0 = b-abs(y1*my);
Canvas->MoveTo(x0,b);Canvas->LineTo(x0,b-h);
Canvas->MoveTo(l,y0);Canvas->LineTo(l+w,y0);
Canvas->TextOutA(x0+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
Canvas->TextOutA(x0+5,b,FloatToStrF(y1,ffGeneral,6,3));

// построение графика
x = x1;
do {
    y = f(x);
    Canvas->Pixels[x0+x*mx][y0-y*my] = clRed;
    x += dx;
} while (x <= x2);
}
```

Основную работу выполняет функция Grafik (ее объявление надо поместить в раздел private объявления формы в заголовочном файле программы). Функция Grafik сначала вычисляет максимальное (y2) и минимальное (y1) значение функции на отрезке [xl, x2]. Затем, используя информацию о ширине и высоте области вывода графика, она вычисляет коэффициенты масштабирования по осям X и Y. После этого вычисляет координату Y горизонтальной оси, координату X вертикальной оси и вычерчивает координатные оси. Затем выполняется непосредственное построение графика (рис. 3.8).

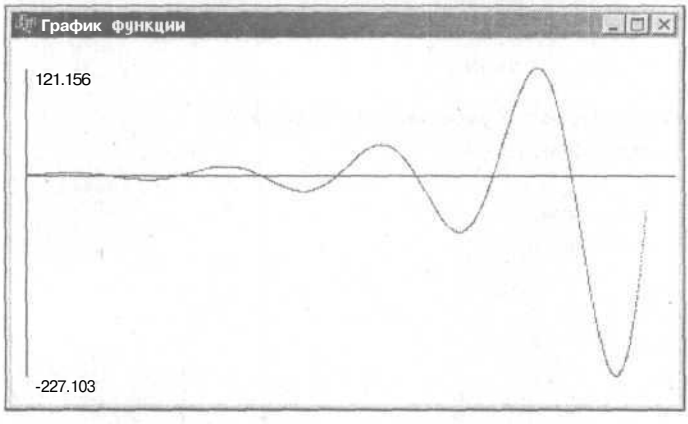


Рис. 3.8. График, построенный поточкам

ВЫЗОВ фуНКЦИИ Grafik ВЫПОЛНЯЮТ фуНКЦИИ обработки событий OnPaint И OnResize. Функция TForm1::FormPaint обеспечивает вычерчивание графика

после появления формы на экране в результате запуска программы, а также после появления формы во время работы программы — например, в результате удаления или перемещения других окон, полностью или частично перекрывающих окно программы. Функция `TForm1::FormResize` обеспечивает вычерчивание графика после изменения размера формы.

Приведенная программа универсальна. Заменив инструкции в теле функции `f(x)`, можно получить график другой функции. Причем независимо от вида функции ее график будет занимать всю область, предназначенную для вывода. Следует обратить внимание на то, что приведенная программа работает корректно, если функция, график которой надо построить, принимает как положительные, так и отрицательные значения. Если функция во всем диапазоне только положительная или только отрицательная, то в программу необходимо внести изменения. Какие — пусть это будет упражнением для читателя.

Иллюстрации

Наиболее просто вывести иллюстрацию, которая находится в файле с расширением bmp, jpg или ico, можно при помощи компонента image, значок которого находится на вкладке **Additional** палитры компонентов (рис. 3.9). Основные свойства компонента приведены в табл. 3.5.

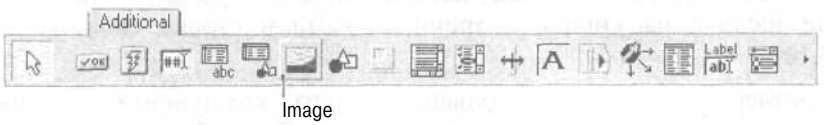


Рис. 3.9. Значок компонента Image

Таблица 3.5. *Свойства компонента image*

Свойство	Описание
Picture	Иллюстрация, которая отображается в поле компонента
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации и значение свойств AutoSize, Strech и Proportional равно `false`, то отображается часть иллюстрации
Proportional	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства `AutoSize` ДОЛЖНО быть `false`
Strech	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена

Таблица 3.5 *(окончание)*

Свойство	Описание
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
Center	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно false, то картинка прижата к правой границе компонента, если true — то картинка располагается по центру
visible	Признак указывает, отображается ли компонент и, соответственно, иллюстрация на поверхности формы
Canvas	Поверхность, на которую можно вывести графику

Иллюстрацию, которая будет выведена в поле компонента image, можно задать как во время разработки формы приложения, так и во время работы программы.

Во время разработки формы иллюстрация задается установкой значения свойства Picture путем выбора файла иллюстрации в стандартном диалоговом окне, которое становится доступным в результате щелчка на командной кнопке **Load** окна **Picture Editor,** которое, в свою очередь, появляется в результате щелчка на кнопке с тремя точками в строке свойства Picture (рис. 3.10).

Если размер иллюстрации больше размера компонента, то свойству Proportional нужно присвоить значение true. Тогда будет выполнено масштабирование иллюстрации в соответствии с реальными размерами компонента.

Чтобы вывести иллюстрацию в поле компонента image во время работы программы, нужно применить метод LoadFromFile к свойству picture, указав в качестве параметра метода файл иллюстрации. Например, инструкция

```
Image1->Picture->LoadFromFile("e:\\temp\\bart.bmp")
```

загружает иллюстрацию из файла bart.bmp и выводит ее в поле компонента вывода иллюстрации (Image1).

По умолчанию компонент image можно использовать для отображения иллюстраций форматов BMP, ICO и WMF. Чтобы использовать компонент для отображения иллюстраций в формате JPEG (файлы с расширением jpg), надо подключить соответствующую библиотеку (поместить в текст программы директиву ttinciude <jpeg.hpp>). Обратите внимание, что если указанной директивы в тексте программы не будет, то компилятор не выведет сообщения об ошибке. Но во время работы программы при попытке загру-

зить jpg-файл при помощи метода LoadFromFile возникнет ошибка — исключение EInvalidGraphic.

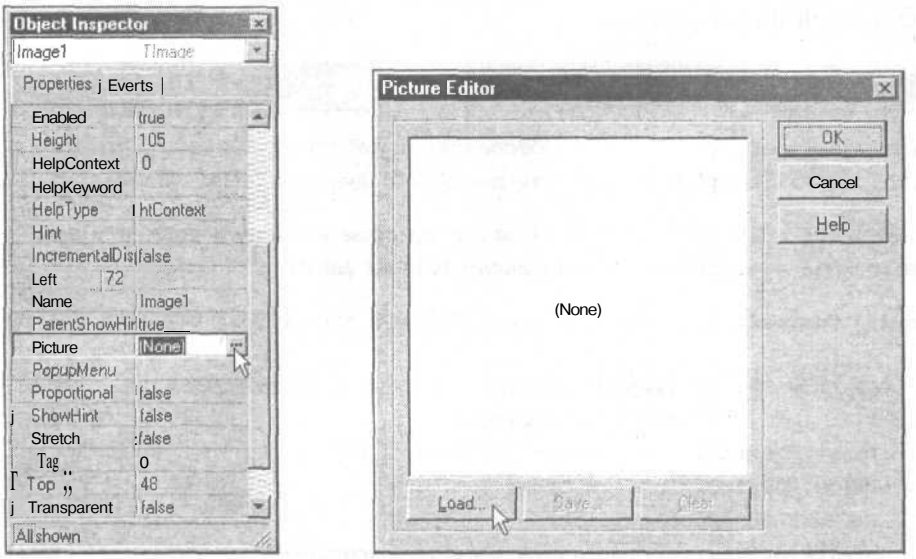


Рис. 3.10. Чтобы выбрать иллюстрацию, щелкните в строке **Picture** на кнопке с тремя точками, **затем** в окне **Picture Editor** — на кнопке **Load**

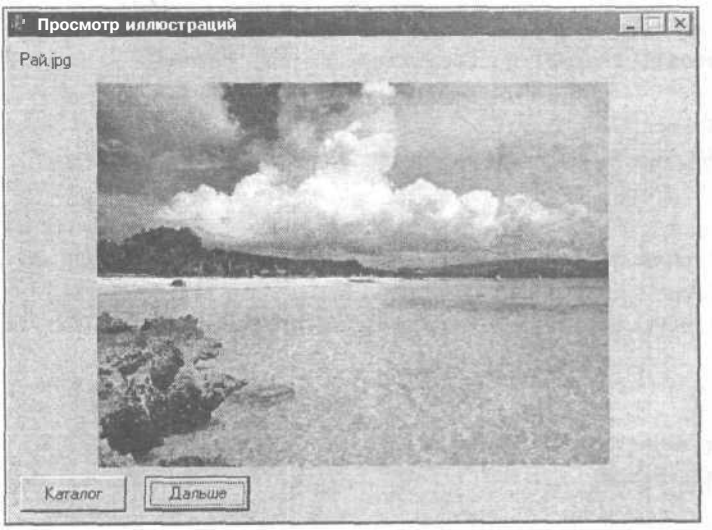


Рис. 3.11. Диалоговое окно программы **Просмотр иллюстраций**

Следующая программа (вид ее окна приведен на рис. 3.11, а текст — в листинге 3.2) использует компонент image для отображения JPG-иллюстраций.

Кнопка **Каталог,** в результате щелчка на которой появляется стандартное диалоговое окно **Выбор папки,** позволяет пользователю выбрать каталог, в котором находятся иллюстрации. Кнопка **Дальше** обеспечивает отображение следующей иллюстрации.

! Листинг 3.2. Просмотр иллюстраций

```
#include <jpeg.hpp>        // обеспечивает работу с JPEG-иллюстрациями
#include <FileCtrl.hpp>  // для доступа к функции SelectDirectory

AnsiString aPath;       // каталог, в котором находится иллюстрация
TSearchRec aSearchRec; // результат поиска файла

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    aPath = ""; // текущий каталог — каталог, из которого
                // запущена программа
    Image1->AutoSize = false;
    Image1->Proportional = true;
    Button2->Enabled = false;
    FirstPicture(); // показать картинку, которая
                    // есть в каталоге программы
}

// щелчок на кнопке Каталог
void __fastcall TForm1::Button1Click(TObject *Sender)
{
     if (SelectDirectory(
          "Выберите каталог, в котором находятся иллюстрации",
          "",aPath) != 0)
     {
        // пользователь выбрал каталог и щелкнул на кнопке OK
        aPath = aPath + "\\";
        FirstPicture();      // вывести иллюстрацию
     }
}

// найти и вывести первую картинку
void TForm1::FirstPicture()
{
    Image1->Visible = false;    // скрыть компонент Image1
    Button2->Enabled = false;  // кнопка Дальше недоступна
    Label1->Caption = "";
    if ( FindFirst(aPath+"*.jpg", faAnyFile, aSearchRec) == 0)
```

```
    {
        Image1->Picture->LoadFromFile(aPath+aSearchRec.Name);
        Image1->Visible = true;
        Label1->Caption = aSearchRec.Name;
        if ( FindNext(aSearchRec) == 0)  // найти след. иллюстрацию
        {
            // иллюстрация есть
            Button2->Enabled = true; // теперь кнопка Дальше доступна
        }
    }
}

// щелчок на кнопке Дальше
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Image1->Picture->LoadFromFile(aPath+aSearchRec.Name);
    Label1->Caption = aSearchRec.Name;
    if ( FindNext(aSearchRec) != 0)  // найти след. иллюстрацию
    {
        // иллюстраций больше нет
        Button2->Enabled = false; // теперь кнопка Дальше недоступна
    }
}
```

Загрузку и вывод первой и остальных иллюстраций выполняют соответственно функции FirstPicture И NextPicture.

Функция FirstPicture вызывает функцию `FindFirst` для того, чтобы получить имя файла первой иллюстрации. В качестве параметров функции FindFirst передаются:

- имя каталога, в котором должны находиться иллюстрации;
- структура aSearchRec, поле Name которой, в случае успеха, будет содержать имя файла, удовлетворяющего критерию поиска;
- маска файла иллюстрации.

Если в указанном при вызове функции FindFirst каталоге есть хотя бы один файл с указанным расширением, то значение функции будет равно нулю. В этом случае метод `LoadFromFile` загружает файл иллюстрации. После загрузки первой иллюстрации функция FirstPicture вызывает функцию FindNext для поиска следующего файла иллюстрации. Если файл будет найден, то кнопка **Дальше** будет сделана доступной.

Функция обработки события Onclick на кнопке **Дальше** загружает следующую иллюстрацию, имя файла которой было найдено функцией FindNext в процессе обработки предыдущего щелчка на кнопке **Дальше,** и снова вы-

зывает функцию FindNext для поиска следующей иллюстрации. Если файл иллюстрации не будет найден, то кнопка **Дальше** станет недоступной.

Необходимо обратить внимание на следующее. Для того чтобы иллюстрации отображались без искажения, свойству AutoSize компонента Image1 надо присвоить значение false, а свойству Proportional — значение true. Сделать это можно во время создания формы в среде разработки (установить значения свойств в окне **Object Inspector)** или возложить задачу настройки компонента на саму программу. В последнем случае в функцию **обработки события** OnCreate ДЛЯ **формы** (TForm1::FormCreate) надо **Добавить** следующие инструкции:

```
Image1->AutoSize = false;
Image1->Proportional = true;
```

Кроме того, во время создания формы свойству Enabled кнопки **Дальше** (Button2) надо присвоить значение false. Это обеспечит корректную работу программы в том случае, если в каталоге, из которого запускается программа, нет иллюстраций. Настройку кнопки Button2 можно возложить на **ФУНКЦИЮ** TForm1::FormCreate. **ДЛЯ ЭТОГО В фунКЦИЮ** надо **ДобаВИТЬ** оператор

```
Button2->Enabled = false
```

Битовые образы

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

Сформировать битовый образ можно путем загрузки из bmp-файла или из ресурса, а также путем копирования фрагмента из другого битового образа, в том числе и с поверхности формы.

Картинку битового образа (иногда говорят просто "битовый образ") можно подготовить при помощи графического редактора или, если предполагается, что битовый образ будет загружен из ресурса программы, — при помощи редактора ресурсов (например, Borland Resource Workshop). В последнем случае надо создать файл ресурсов и поместить в него битовый образ. Файл ресурсов можно создать и при помощи утилиты Image Editor.

В программе битовый образ — это объект типа TBitmap. Некоторые свойства объекта TBitmap приведены в табл. 3.6.

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

```
Graphics::TBitmap *Plane = new Graphics::TBitmap();
Plane->LoadFromFile("plane.bmp");
```

В результате выполнения приведенного выше фрагмента, битовый образ Plane представляет собой изображение самолета (предполагается, что в файле plane.bmp находится изображение самолета).

Таблица 3.6. *Свойства объекта 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 выводится. По умолчанию значение TransparentColor определяет цвет левого нижнего пиксела
TransparentColor	Задает прозрачный цвет. Элементы картинки, окрашенные этим цветом, методом Draw не выводятся
Canvas	Поверхность битового образа, на которой можно рисовать точно так же, как на поверхности формы или компонента image

После того как битовый образ сформирован (загружен из файла или из ресурса), его можно вывести, например, на поверхность формы или компонента Image. Сделать ЭТО МОЖНО, Применив метод Draw К СВОЙСТВУ Canvas. В качестве параметров методу Draw надо передать координаты точки, от которой будет выведен битовый образ. Например, оператор

```
Canvas->Draw(10,20,Plane);
```

выводит на поверхность формы битовый образ Plane — изображение самолета.

Если перед применением метода Draw свойству Transparent битового образа присвоить значение true, то фрагменты рисунка, цвет которых совпадает с цветом левой нижней точки рисунка, не будут выведены. Такой прием используется для создания эффекта прозрачного фона. "Прозрачный" цвет можно задать и принудительно, присвоив соответствующее значение свойству TransparentColor.

Следующая программа демонстрирует работу с битовыми образами. После запуска программы в ее окне появляется изображение неба и двух самоле-

тов (рис. 3.12). И небо, и самолеты — это битовые образы, загруженные из файлов во время работы программы. Загрузку и вывод битовых образов на поверхность формы выполняет функция обработки события OnPaint, текст которой приведен в листинге 3.3. Белое поле вокруг левого самолета показывает истинный размер битового образа Plane. Белого поля вокруг правого самолета нет, т. к. перед тем как вывести битовый образ второй раз, свойству Transparent было присвоено значение true.

Рис. 3.12. Присвоив свойству Transparent значение true, можно скрыть фон

Листинг 3.3. Загрузка и вывод битовых образов на поверхность формы

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    // битовые образы: небо и самолет
    Graphics::TBitmap *sky = new Graphics::TBitmap();
    Graphics::TBitmap *plane = new Graphics::TBitmap();

    sky->LoadFromFile("sky.bmp");
    plane->LoadFromFile("plane.bmp");

    Canvas->Draw(0,0,sky);      // фон — небо

    Canvas->Draw(20,20,plane); // левый самолет

    plane->Transparent = true;
    /* теперь элементы рисунка, цвет которых совпадает
       с цветом левой нижней точки битового образа,
       не отображаются */
    Canvas->Draw(120,20,plane); // правый самолет

    // уничтожить объекты
    sky->Graphics::~TBitmap();
    plane->Graphics::~TBitmap();
}
```

Небольшие по размеру битовые образы часто используют при формировании фоновых рисунков по принципу кафельной плитки (рис. 3.13).

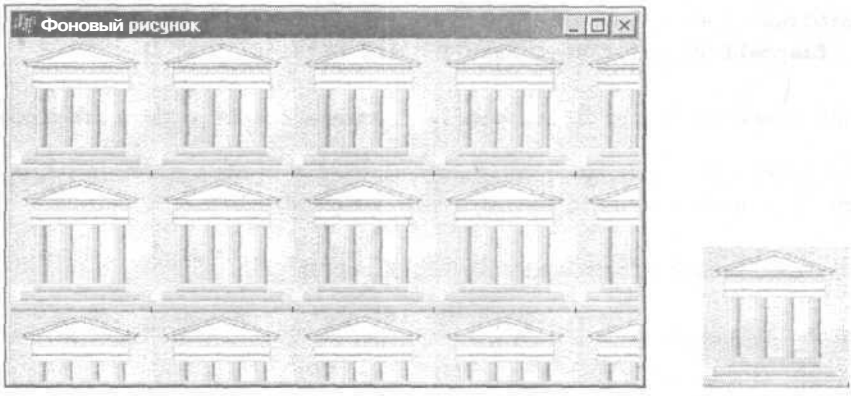


Рис. 3.13. Фоновый рисунок и битовый образ-плитка, из которого он составлен

Следующая программа показывает, как можно получить фоновый рисунок путем многократного вывода битового образа на поверхность формы. Формирование фонового рисунка, многократный вывод битового образа на поверхность формы выполняет функция Background. Ее объявление (прототип), а также объявление битового образа (объекта типа TBitmap) надо поместить в секцию private объявления класса формы (листинг 3.4), которая находится в заголовочном файле. Создание битового образа и загрузку картинки из файла выполняет функция обработки события OnCreate. Функция обработки события OnPaint путем вызова функции Background обеспечивает вывод фонового рисунка на поверхность формы (листинг 3.5).

[Листинг 3.4. Объявление битового образа и функции `Background`

```
class TForm1 : public TForm
{
__published:
    void__fastcall FormCreate(TObject *Sender);
    void__fastcall FormPaint(TObject *Sender);
    void__fastcall FormResize(TObject *Sender);
private:
  Graphics::TBitmap *back;       // элемент фонового рисунка
  void__fastcall Background(); // формирует фоновый рисунок на
                               // поверхности формы
public:
    __fastcall TForm1(TComponent* Owner);
};
```

Листинг 3.5. Функции, обеспечивающие формирование и вывод фонового рисунка

```
// обработка события OnCreate
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    back = new Graphics::TBitmap(); // создать объект — битовый образ

     // загрузить картинку
    try // в процессе загрузки картинки возможны ошибки
    {
        Form1->back->LoadFromFile("Legal.bmp");
    }
    catch (EFOpenError &e)
    {
        return;
    }
}

// формирует фоновый рисунок
void __fastcall TForm1::Background()
{
    int x=0,y=0; // координаты левого верхнего угла битового образа

    if ( back->Empty) // битовый образ не был загружен
        return;

    do {
      do {
        Canvas->Draw(x,y,back);
        x += back->Width;
      }
      while (x < ClientWidth);
      x = 0;
      y += back->Height;
    }
    while (y < ClientHeight);
}

// обработка события OnPaint
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    Background(); // обновить фоновый рисунок
}
```

Мультипликация

Под мультипликацией обычно понимается движущийся и меняющийся рисунок. В простейшем случае рисунок может только двигаться или только меняться.

Обеспечить перемещение рисунка довольно просто: надо сначала вывести рисунок на экран, затем через некоторое время стереть его и снова вывести этот же рисунок, но уже на некотором расстоянии от его первоначального положения. Подбором времени между выводом и удалением рисунка, а также расстояния между старым и новым положением рисунка (шага перемещения), можно добиться того, что у наблюдателя будет складываться впечатление, что рисунок равномерно движется по экрану.

Метод базовой точки

Следующая простая программа показывает, как можно заставить двигаться изображение, сформированное из графических примитивов. Окно и форма программы приведены на рис. 3.14.

Рис. 3.14. Окно и форма программы

На поверхности формы находится один-единственный компонент Timer, который используется для генерации последовательности событий, функция обработки которых обеспечивает вывод и удаление рисунка. Значок компонента Timer находится на вкладке **System** (рис. 3.15). Следует обратить внимание, что компонент Timer является *невизуальным*. Это значит, что во время работы программы компонент в диалоговом окне не отображается. Поэтому компонент Timer можно поместить в любую точку формы. Свойства компонента Timer приведены в табл. 3.7.

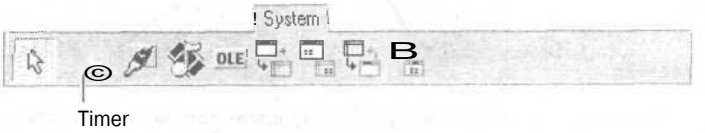


Рис. 3.15. Значок компонента Timer

Таблица 3.7. *Свойства компонента Timer*

Свойство	Определяет
Name	Имя компонента. Используется для доступа к свойствам компонента
Interval	Период возникновения события OnTimer. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение true) или запрещает (значение false) возникновение события OnTimer

Компонент Timer генерирует событие OnTimer. Период возникновения события OnTimer измеряется в миллисекундах и определяется значением свойства interval. Следует обратить внимание на свойство Enabled. Оно дает возможность программе "запустить" или "остановить" таймер. Если значение свойства Enabled равно false, то событие OnTimer не возникает.

В рассматриваемой программе вывод изображения выполняет функция ship, которая рисует на поверхности формы кораблик. В качестве параметров функция ship получает координаты *базовой тонки*. Базовая точка (x_0, y_0) определяет положение графического объекта в целом; от нее отсчитываются координаты графических примитивов, образующих объект (рис. 3.16). Координаты фафических примитивов можно отсчитывать от базовой точки не в пикселах, а в относительных единицах. Такой подход позволяет легко выполнить масштабирование изображения.

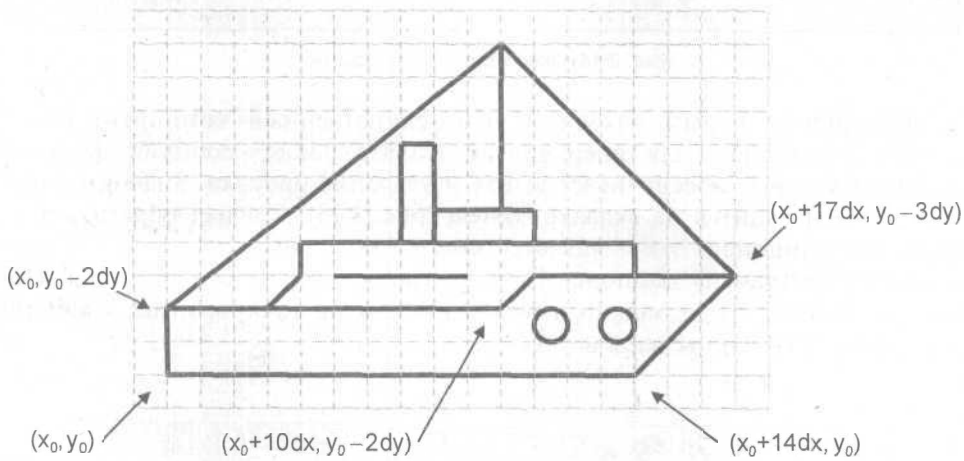


Рис. 3.16. Базовая точка (x_0, y_0) определяет положение объекта

Перед тем как нарисовать кораблик на новом месте, функция обработки события от таймера стирает кораблик, нарисованный в процессе обработки предыдущего события OnTimer. Изображение кораблика стирается путем вывода прямоугольника, перекрывающего его.

Функции Обработки события OnTimer, **функция** Ship **и функция** FormCreate, обеспечивающая настройку таймера, приведены в листинге 3.6

Листинг 3.6. Простая мультипликация

```
int x = -68, y = 50; // начальное положение базовой точки

// рисует на поверхности формы кораблик
void __fastcall TForm1::Ship(int x, int y)
{
  int dx=4,dy=4; // шаг сетки
  // корпус и надстройку будем рисовать
  // при помощи метода Polygon
  TPoint p1[7]; // координаты точек корпуса
  TPoint p2[8]; // координаты точек надстройки

  TColor pc,bc; // текущий цвет карандаша и кисти

  // сохраним текущий цвет карандаша и кисти
  pc = Canvas->Pen->Color;
  bc = Canvas->Brush->Color;

  // установим нужный цвет карандаша и кисти
  Canvas->Pen->Color = clBlack;
  Canvas->Brush->Color = clWhite;

  // рисуем . . .
  // корпус
  p1[0].x = x;       p1[0].y = y;
  p1[1].x = x;       p1[1].y = y-2*dy;
  p1[2].x = x+10*dx; p1[2].y = y-2*dy;
  p1[3].x = x+11*dx; p1[3].y = y-3*dy;
  p1[4].x = x+17*dx; p1[4].y =y-3*dy;
  p1[5].x = x+14*dx; p1[5].y =y;
  p1[6].x = x;       p1[6].y =y;
  Canvas->Polygon(p1,6);

  // надстройка
  p2[0].x = x+3*dx;   p2[0].y = y-2*dy;
  p2[1].x = x+4*dx;   p2[1].y = y-3*dy;
```

```
  p2[2].x = x+4*dx;   p2[2].y = y-4*dy;
  p2[3].x = x+13*dx;  p2[3].y = y-4*dy;
  p2[4].x = x+13*dx;  p2[4].y = y-3*dy;
  p2[5].x = x+11*dx;  p2[5].y = y-3*dy;
  p2[6].x = x+10*dx;  p2[6].y = y-2*dy;
  p2[7].x = x+3*dx;   p2[7].y = y-2*dy;
  Canvas->Polygon(p2,7);

  Canvas->MoveTo(x+5*dx,y-3*dy);
  Canvas->LineTo(x+9*dx,y-3*dy);

  // капитанский мостик
  Canvas->Rectangle(x+8*dx,y-4*dy,x+11*dx,y-5*dy);

  // труба
  Canvas->Rectangle(x+7*dx,y-4*dy,x+8*dx,y-7*dy);

  // иллюминаторы
  Canvas->Ellipse(x+11*dx,y-2*dy,x+12*dx,y-1*dy);
  Canvas->Ellipse(x+13*dx,y-2*dy,x+14*dx,y-1*dy);

  // мачта
  Canvas->MoveTo(x+10*dx,y-5*dy);
  Canvas->LineTo(x+10*dx,y-10*dy);

  // оснастка
  Canvas->Pen->Color = clWhite;
  Canvas->MoveTo(x+17*dx,y-3*dy);
  Canvas->LineTo(x+10*dx,y-10*dy);
  Canvas->LineTo(x,y-2*dy);

  // восстановим цвет карандаша и кисти
  Canvas->Pen->Color = pc;
  Canvas->Brush->Color = bc;
}

// обработка события OnTimer
void _ fastcall TForm1: :Timer1Timer (TObject *Sender)
{
    // стереть кораблик — закрасить цветом, совпадающим
    // с цветом фона (формы)
    Canvas->Brush->Color = Form1->Color;
    Canvas->FillRect(Rect(x-1,y+1,x+68,y-40));

    // вычислить координаты базовой точки
    x+=3;
```

```
    if (x > ClientWidth) {
        // кораблик "уплыл" за правую границу формы
        x= -70; // чтобы кораблик "выплывал" из-за левой границы формы
        y=random(Form1->ClientHeight);
    }
    // нарисовать кораблик на новом месте
    Ship(x,y);
}

// обработка события OnCreate для формы
void__fastcall TForm1::FormCreate(TObject *Sender)
{
    /* Таймер можно настроить во время разработки программы
       (в процессе создания формы) или во время работы программы. */

    // настройка и запуск таймера
    Timer1->Interval = 100; // период события OnTimer - 0.1 сек.
    Timer1->Enabled = true; // пуск таймера
}
```

Использование битовых образов

В последнем примере изображение формировалось из графических примитивов. Теперь рассмотрим, как можно реализовать перемещение заранее подготовленного при помощи графического редактора изображения.

Как и в предшествующей программе, эффект перемещения объекта (картинки) достигается за счет периодической перерисовки картинки с некоторым смещением относительно ее прежнего положения. Перед выводом картинки в новой точке предыдущее изображение должно быть удалено, а фоновый рисунок, который был перекрыт — восстановлен. Удалить (стереть) картинку и одновременно восстановить фон можно путем перерисовки всей фоновой картинки или только той ее части, которая была перекрыта объектом. В рассматриваемой программе используется второй подход. Изображение объекта выводится применением метода Draw к свойству canvas формы, а стирается путем копирования (метод copyRect) нужной части фона из буфера в битовый образ, соответствующий поверхности формы.

Форма программы приведена на рис. 3.17, а текст — в листинге 3.6. Компонент Timer используется для организации цикла удаления и вывода изображения самолета.

Рис. 3.17. Форма программы "Полет над городом"

! Листинг 3.7. "Полет над городом"

```
void__fastcall TForm1::FormCreate(TObject *Sender)
{
    // загрузить фоновый рисунок из bmp-файла
    back = new Graphics::TBitmap();
    back->LoadFromFile("factory.bmp");

    // установить размер клиентской (рабочей) области формы
    // в соответствии с размером фонового рисунка
    ClientWidth = back->Width;
    ClientHeight = back->Height;

    // загрузить картинку
    sprite = new Graphics::TBitmap();
    sprite->LoadFromFile("aplane.bmp");
    sprite->Transparent = true;

    // исходное положение самолета
    x=-20; // чтобы самолет "вылетал" из-за левой границы окна
    y=20;
}

void__fastcall TForm1::FormPaint(TObject *Sender)
{
    Canvas->Draw(0,0,back);     // фон
    Canvas->Draw(x,y,sprite);   // рисунок
}

void__fastcall TForm1::Timer1Timer(TObject *Sender)
{
    TRect badRect;  // положение и размер области фона,
                    // которую надо восстановить
```

```
    badRect = Rect(x,y,x+sprite->Width,y+sprite->Height);

    // стереть самолет (восстановить "испорченный" фон)
    Canvas->CopyRect(badRect,back->Canvas,badRect);

    // вычислим новые координаты спрайта (картинки)
    x +=2;
    if (x > ClientWidth)
    {
        // самолет улетел за правую границу формы
        // изменим высоту и скорость полета
        x = -20;
        y = random(ClientHeight — 30);  // высота полета"
        // скорость полета определяется периодом возникновения
        // события OnTimer, который, в свою очередь, зависит
        // от значения свойства Interval
        Timer1->Interval = random(20) + 10; // скорость "полета" меняется
                                            // от 10 до 29
    }
    Canvas->Draw(x,y,sprite);
}
```

Для хранения битовых образов (картинок) фона и самолета используются **два объекта ТИПа** TBitmap, **Которые создает** функция TForml::FormCreate (объявления этих объектов надо поместить в заголовочный файл проекта). Эта же функция загружает из файлов картинки фона (factory.bmp) и самолета (aplane.bmp).

Восстановление фона выполняется при помощи метода CopyRect, который позволяет выполнить копирование прямоугольного фрагмента одного битового образа в другой. Объект, к которому применяется метод CopyRect, является приемником копии битового образа. В качестве параметров методу передаются: координаты и размер области, куда должно быть выполнено копирование; поверхность, с которой должно быть выполнено копирование; положение и размер копируемой области. Информация о положении и размере восстанавливаемой (копируемой на поверхность формы) области фона находится в структуре badRect типа TRect.

Следует обратить внимание на то, что начальное значение переменной x, которая определяет положение левой верхней точки битового образа (движущейся картинки) — отрицательное число, равное ширине битового образа картинки. Поэтому в начале работы программы самолет не виден и картинка отрисовывается за границей видимой области. С каждым событием OnTimer 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.

Загрузка битового образа из ресурса программы

В программе "Полет над городом" (листинг 3.7) картинки (битовые образы) фона и объекта (самолета) загружаются из файлов. Такой подход не всегда удобен. C++ Builder позволяет поместить нужные битовые образы в исполняемый файл программы и по мере необходимости загружать их непосредственно оттуда.

Битовый образ, находящийся в выполняемом файле программы, называется *ресурсом, а* операция загрузки битового образа из выполняемого файла – *загрузкой* битового образа *из ресурса*.

Для того чтобы воспользоваться возможностью загрузки картинки из ресурса, сначала надо создать *файл ресурсов* и поместить в него нужные картинки.

Создание файла ресурсов

Файл ресурсов можно создать при помощи утилиты Image Editor, которая поставляется вместе с C++ Builder. Запустить Image Editor можно из C++ Builder, выбрав в меню **Tools** команду **Image Editor,** или из Windows, выбрав команду **Пуск | Программы | Borland C++Builder | Image Editor.**

Для того чтобы создать файл ресурсов, надо в меню **File** выбрать команду New, а затем в появившемся подменю — команду **Resource File** (рис. 3.18). В результате выполнения команды будет создан файл ресурсов Untitled1.res (рис. 3.19), в который надо поместить необходимые ресурсы.

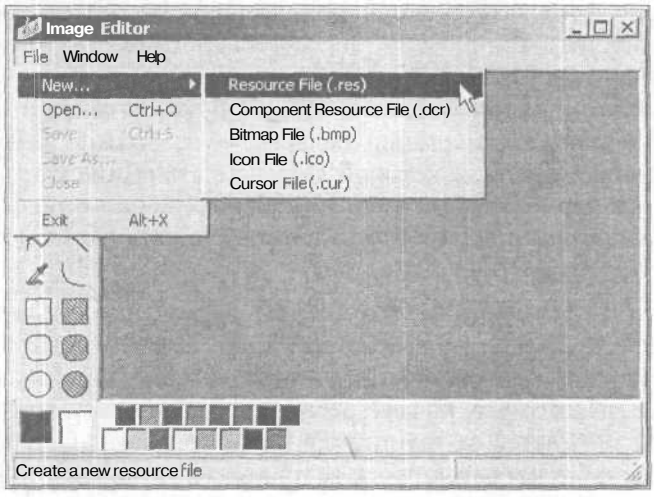


Рис. 3.18. Чтобы создать файл ресурсов, выберите команду **File | New | Resource File**

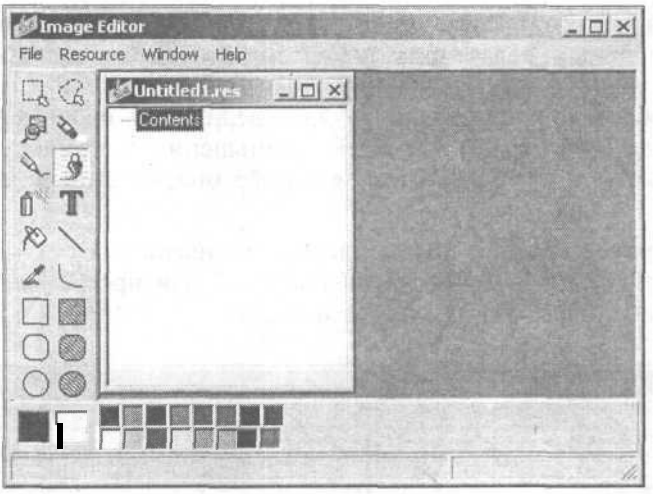


Рис. 3.19. Файл ресурсов создан. Теперь в него надо поместить необходимые ресурсы

Для того чтобы в файл ресурсов добавить новый ресурс, надо в меню **Resource** выбрать команду New | **Bitmap** (Новый битовый образ). В результате выполнения этой команды открывается диалоговое окно **Bitmap Properties** (Характеристики битового образа), в котором нужно установить размер (в пикселах) битового образа и выбрать палитру (рис. 3.20). В результате нажатия кнопки **OK** в списке **Contents** появится новый элемент **Bitmap1**, соответствующий новому ресурсу, добавленному в файл ресурсов (рис. 3.21).

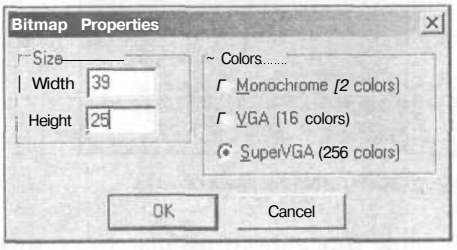


Рис. 3.20. В диалоговом окне **Bitmap Properties** надо задать характеристики создаваемого битового образа

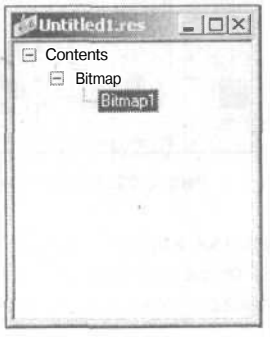


Рис. 3.21. Окно файла ресурсов после добавления ресурса **Bitmap**

Bitmap1 — это автоматически созданное имя ресурса, которое можно изменить, выбрав команду **Resource | Rename.** После этого можно приступить к редактированию (созданию) битового образа. Для этого надо в меню **Resource** выбрать команду **Edit.** В результате этих действий будет активизирован режим редактирования битового образа.

Графический редактор Image Editor предоставляет программисту стандартный для подобных редакторов набор инструментов, используя которые можно нарисовать нужную картинку. Если во время работы надо изменить масштаб отображения картинки, то для увеличения масштаба следует выбрать команду **View | Zoom In,** а для уменьшения — команду **View | Zoom Out.** Увидеть картинку в реальном масштабе можно, выбрав команду **View | Actual Size.**

В качестве примера на рис. 3.22 приведен вид диалогового окна **Image Editor,** в котором находится файл ресурсов images.res для программы Flight. Файл содержит два битовых образа FACTORY и APLANE.

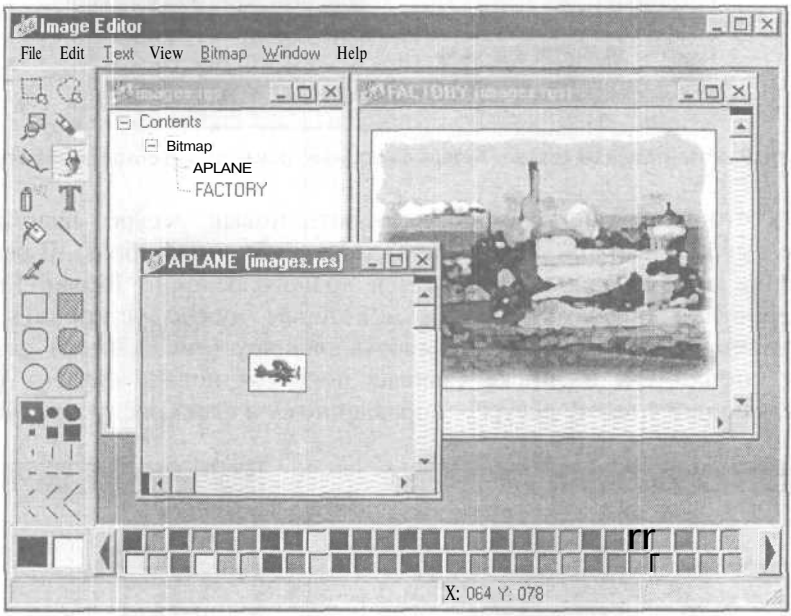


Рис. 3.22. Файл ресурсов images.res содержит два битовых образа

Если нужная картинка уже существует в виде отдельного файла, то ее можно через буфер обмена (Clipboard) поместить в битовый образ файла ресурсов. Делается это следующим образом.

1. Сначала надо запустить графический редактор, например Microsoft Paint, загрузить в него файл картинки и выделить всю картинку или ее часть. В процессе выделения следует обратить внимание на информацию о размере (в пикселах) выделенной области (Paint выводит размер выделяемой области в строке состояния). Затем, выбрав команду **Копировать** меню **Правка,** необходимо поместить копию выделенного фрагмента в буфер.
2. Далее нужно переключиться в Image Editor, выбрать ресурс, в который надо поместить находящуюся в буфере картинку, и установить значения

характеристик ресурса в соответствии с характеристиками картинки, находящейся в буфере. Значения характеристик ресурса вводятся в поля диалогового окна **Bitmap Properties,** которое открывается выбором команды **Image Properties** меню **Bitmap.** После установки характеристик ресурса можно вставить картинку в ресурс, выбрав команду **Past** меню **Edit.**

После добавления всех нужных ресурсов файл ресурса следует сохранить в том каталоге, где находится программа, для которой этот файл создается. Сохраняется файл ресурса обычным образом, т. е. выбором команды Save меню **File**. Image Editor присваивает файлу ресурсов расширение res.

Подключение файла ресурсов

Для того чтобы ресурсы, находящиеся в файле ресурсов, были доступны программе, в текст профаммы надо поместить инструкцию (директиву), которая сообщит компилятору, что в исполняемый файл следует добавить содержимое файла ресурсов.

В общем виде эта директива выглядит следующим образом:

```
ipragma resource ФайлРесурсов
```

где *ФайлРесурсов* — имя файла ресурсов.

Например, для профаммы flight_1 директива, обеспечивающая включение содержимого файла ресурсов в выполняемый файл, выглядит так:

```
#pragma resource "images.res"
```

Загрузить битовый образ из ресурса можно при помощи метода LoadFromResourceName, который имеет два параметра: идентификатор программы и имя ресурса. В качестве идентификатора профаммы используется глобальная переменная HInstance. Имя ресурса должно быть представлено в виде строковой константы.

Например, в профамме flight_1 инструкция зафузки фона из ресурса выглядит так:

```
back->LoadFromResourceName((int)HInstance,"FACTORY");
```

В качестве примера в листинге 3.8 приведен фрагмент профаммы flight_1 – функция TForm1::FormCreate, которая обеспечивает загрузку битовых образов из ресурсов.

Листинг 3.8. Загрузка битовых образов из ресурса

```
// подключить файл ресурсов, в котором находятся
// необходимые программе битовые образы
#pragma resource "images.res"
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    // загрузить фоновый рисунок из ресурса
    back = new Graphics::TBitmap();
    back->LoadFromResourceName((int)HInstance,"FACTORY");

    // установить размер клиентской (рабочей) области формы
    // в соответствии с размером фонового рисунка
    ClientWidth = back->Width;
    ClientHeight = back->Height;

    // загрузить изображение объекта из ресурса
    sprite = new Graphics::TBitmap();
    sprite->LoadFromResourceName((int)HInstance,"APLANE");
    sprite->Transparent = true;

    // исходное положение самолета
    x=-20; // чтобы самолет "вылетал" из-за левой границы окна
    y=20;
}
```

Преимущества загрузки картинок из ресурса программы очевидны: при распространении программы не надо заботиться о том, чтобы во время работы программы были доступны файлы иллюстраций, т. к. все необходимые программе картинки находятся в исполняемом файле.

Теперь рассмотрим, как можно реализовать в диалоговом окне программы вывод *баннера*, представляющего собой последовательность сменяющих друг друга картинок (кадров). Кадры баннера обычно находятся в одном файле или в одном ресурсе. В начале работы программы они загружаются в буфер (объект типа `TBitmap`). Вывод кадров баннера обычно выполняет функция обработки сигнала от таймера (события `OnTimer`), которая выделяет очередной кадр и выводит его на поверхность формы.

Вывести кадр баннера (фрагмент битового образа) на поверхность формы можно при помощи метода copyRect, который копирует прямоугольную область одной графической поверхности на другую.

Инструкция применения метода copyRect в общем виде выглядит так:

Canvas1->CopyRect(*Область1*, *Canvas2*, *Область2*)

где:

- *Canvas1* — поверхность, на которую выполняется копирование;
- *Canvas2* — поверхность, с которой выполняется копирование;
- *Область1* — структура типа TRect, которая задает положение и размер области, куда выполняется копирование (приемник);

- *Область2* — структура типа TRect, которая задает положение и размер области, откуда выполняется копирование (источник).

Определить прямоугольную область, заполнить поля структуры TRect можно при помощи функции Rect или Bounds. Функции Rect надо передать в качестве параметров координаты левого верхнего и правого нижнего углов области, функции Bounds — координаты левого верхнего угла и размер области. Например, если надо определить прямоугольную область, то это можно сделать так:

```
rct = Rect(x1,y1,x2,y2)
```

или так:

```
rct = Bounds(x1,y1,w,h)
```

где xl, yl — координаты левого верхнего угла области; x2, y2 — координаты правого нижнего угла области; w и h — ширина и высота области.

Следующая программа (ее текст приведен в листинге 3.9) выводит в диалоговое окно баннер — рекламное сообщение. На рис. 3.23 приведены кадры этого баннера (содержимое файла baner.bmp), а на рис. 3.24 — диалоговое окно. Форма программы содержит один-единственный компонент — таймер.

Рис. 3.23. Кадры баннера

Рис. 3.24. Воспроизведение баннера в окне программы

Листинг 3.9. Баннер (baner.h, baner_.cpp)

```
// baner.h
class TForm1 : public TForm
{
__published:
    TTimer *Timer1;
```

```
    void _ fastcall FormCreate (TObject *Sender) ;
    void _fastcall Timer1Timer (TObject * Sender);
private:
     Graphics::TBitmap *baner; // баннер
     TRect kadr;               // кадр баннера
     TRect scr;                // область воспроизведения баннера
     int w, h;                 // размер кадра
     int c;                    // номер воспроизводимого кадра

public:
    _ fastcall TForm1 (TComponent* Owner);
};

// baner_.cpp
#define FBANER "borland.bmp" // баннер
#define NKADR  4 // количество кадров в баннере

void _ fastcall TForm1 :: FormCreate (TObject *Sender)
{
    baner = new Graphics : :TBitmap();
    baner->LoadFromFile (FBANER);     // загрузить баннер

    h = baner->Height;
    w = baner->Width / NKADR;

    scr = Rect (10,10,10+w,10+h) ; // положение и размер области
                                   // воспроизведения баннера
    kadr = Rect (0,0,w,h) ;     // положение и размер первого кадра
                                // в баннере
}

// обработка события OnTimer
void _ fastcall TForm1: : Timer1Timer (TObject *Sender)
{
    // вывести кадр баннера
    Canvas->CopyRect(scr,baner->Canvas,kadr);

    // подготовиться к воспроизведению следующего кадра
    if (c < NKADR)
    (
        // воспроизводимый в данный момент
        // кадр — не последний
        c++;
        kadr.Left += w;
        kadr.Right += w;
    }
```

```
    else
    {
        C = 0;
        kadr.Left = 0;
        kadr.Right = w;
    }
}
```

Программа состоит из двух функций. Функция `TForm1::FormCreate` создает объект `TBitmap` и загружает в него баннер — BMP-файл, в котором находятся кадры баннера. Затем, используя информацию о размере загруженного битового образа, функция устанавливает значения характеристик кадра: высоту и ширину.

Основную работу в программе выполняет функция обработки события OnTimer, которая выделяет из битового образа `baner` очередной кадр и выводит его на поверхность формы. Выделение кадра и его отрисовку путем копирования фрагмента картинки с одной поверхности на другую выполняет метод copyRect (рис. 3.25), которому в качестве параметров передаются координаты области, куда нужно копировать, поверхность и положение области, откуда нужно копировать. Положение фрагмента в фильме, т. е. координата x левого верхнего угла, определяется умножением ширины кадра на номер текущего кадра.

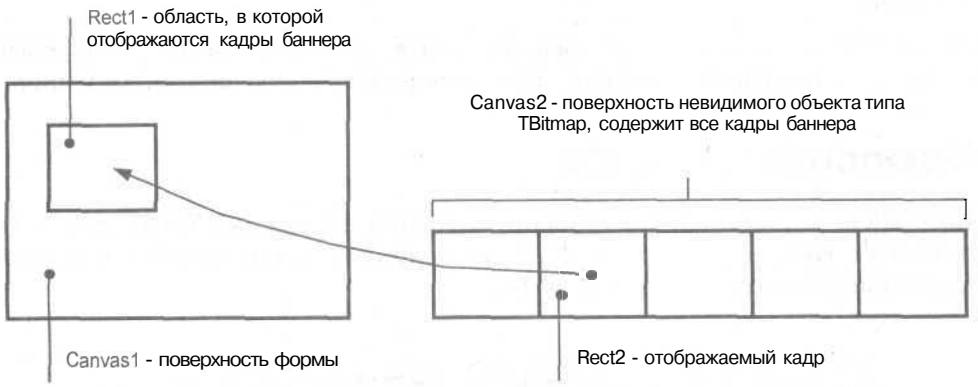


Рис. 3.25. Метод CopyRect копирует в область Rectl поверхности Canvasl область Rect2 с поверхности Canvas2

ГЛАВА 4

Мультимедиа

Большинство современных программ, работающих в среде Windows, являются мультимедийными. Такие программы обеспечивают просмотр видеороликов и мультипликации, воспроизведение музыки, речи, звуковых эффектов. Типичные примеры мультимедийных программ — игры и обучающие программы.

C++ Builder предоставляет в распоряжение программиста два компонента, которые позволяют разрабатывать мультимедийные программы:

- Animate — обеспечивает вывод простой, не сопровождаемой звуком анимации;
- `MediaPlayer` - позволяет решать более сложные задачи, например воспроизводить видеоролики, звук и сопровождаемую звуком анимацию.

Компонент *Animate*

Компонент Animate, значок которого находится на вкладке **Win32** (рис. 4.1), позволяет воспроизводить простую, *не сопровождаемую звуком* анимацию, кадры которой находятся в AVI-файле.

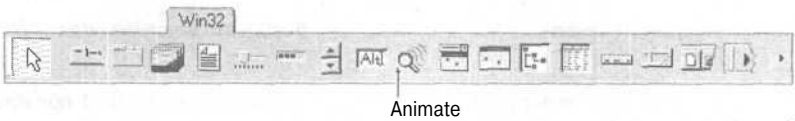


Рис. 4.1. Значок компонента Animate

Компонент Animate добавляется к форме обычным образом. После того как компонент будет добавлен к форме, следует выполнить его настройку – установить значения свойств. Свойства компонента Animate перечислены в табл. 4.1.

Таблица 4.1. *Свойства компонента Animate*

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента и для управления его поведением
FileName	Имя AVI-файла, в котором находится анимация, отображаемая при помощи компонента
FrameWidth	Ширина кадров анимации
FrameHeight	Высота кадров анимации
FrameCount	Количество кадров анимации
AutoSize	Признак автоматического изменения размера компонента в соответствии с размером кадров анимации
Center	Признак центрирования кадров анимации в поле компонента. Если значение свойства равно true и размер компонента больше размера кадров (AutoSize = false), кадры анимации располагаются в центре поля компонента
StartFrame	Номер кадра, с которого начинается отображение анимации
StopFrame	Номер кадра, на котором заканчивается отображение анимации
Active	Признак активизации процесса отображения анимации
Color	Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация
Transparent	Режим использования "прозрачного" цвета при отображении анимации
Repetitions	Количество повторов отображения анимации
CommonAVI	Определяет стандартную анимацию Windows (см. табл. 4.2)

Компонент Animate позволяет программисту использовать в своих программах стандартные анимации Windows. Вид анимации определяется значением свойства CommonAVI. Значение свойства задается при помощи именованной константы. В табл. 4.2 приведены некоторые константы, вид анимации и описание процессов, для иллюстрации которых используются эти анимации.

Таблица 4.2. *Значение свойства CommonAvi определяет анимацию*

Значение	Анимация	Процесс
aviCopyFiles		Копирование файлов

Таблица 4.2 (окончание)

Значение	Анимация	Процесс
aviDeleteFile		Удаление файла
aviRecycleFile		Удаление файла в корзину

Следует еще раз обратить внимание, что компонент Animate предназначен для воспроизведения AVI-файлов, которые содержат *только* анимацию. При попытке записать в свойство `FileName` имя файла, в котором находится сопровождаемая звуком анимация, возникает исключение и C++ Builder выводит сообщение об ошибке **Cannot open AVI.**

Следующая программа (вид ее диалогового окна приведен на рис. 4.2, а текст — в листинге 4.1) демонстрирует использование компонента Animate для просмотра анимации.

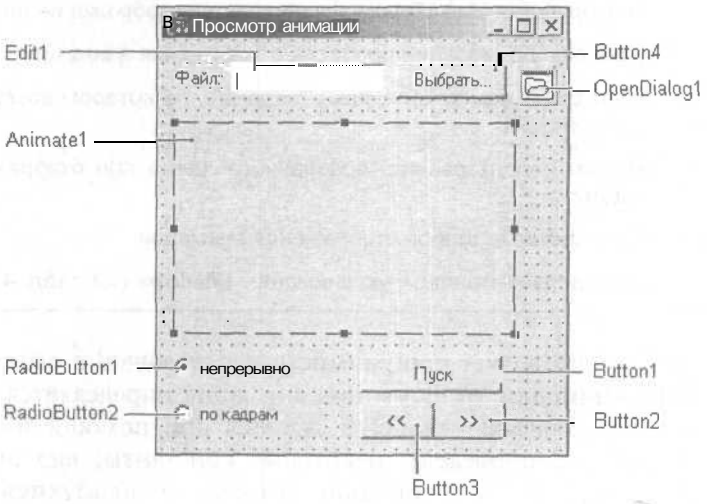


Рис. 4.2. Форма программы "Просмотр анимации"

После запуска программы в форме будет выведен первый кадр анимации, которая находится в каталоге проекта. Если ни одного файла с расширением avi в каталоге проекта нет, то поле компонента Animate останется пустым.

Имя файла, в котором находится анимация, можно ввести в поле `Edit1` или выбрать в стандартном диалоговом окне **Открыть файл,** которое становится

доступным в результате щелчка на кнопке **Выбрать.** Доступ к стандартному диалоговому окну **Открыть файл** обеспечивает компонент OpenDialog1. Значок компонента OpenDialog находится на вкладке **Dialogs.**

Программа "Просмотр анимации" обеспечивает два режима просмотра: непрерывный и по кадрам. Кнопка Button1 используется как для инициализации процесса воспроизведения анимации, так и для его приостановки. Процесс непрерывного воспроизведения анимации инициирует процедура обработки события Onclick на кнопке **Пуск,** которая присваивает значение true свойству Active. Эта же процедура заменяет текст на кнопке Button1 с "Пуск" на "Стоп". Режим воспроизведения анимации выбирается при помощи переключателей RadioButton1 и RadioButton2. Процедуры обработки события onclick на этих переключателях изменением значения свойства Enabled блокируют или, наоборот, делают доступными кнопки управления: активизации воспроизведения анимации (Button1), перехода к следующему (Button2) и предыдущему (Button3) кадру. Во время непрерывного воспроизведения анимации процедура обработки события onckick на кнопке **Стоп** (Buttoni) присваивает значение false свойству Active и тем самым останавливает процесс воспроизведения анимации.

Листинг 4.1. Использование компонента Animate

```
// обработка события OnCreate
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    TSearchRec sr;  // содержит информацию
                    // о файле, найденном функцией FindFirst

    //  найдем AVI-файл в текущем каталоге
    if (FindFirst("*.avi",faAnyFile,sr) == 0)
    {
        Edit1->Text = sr.Name;

        /* если анимация содержит звук, то при
           выполнении следующего оператора произойдет
           ошибка, т. к. компонент Animate обеспечивает
           воспроизведение только простой,
           не сопровождаемой звуком анимации
        */

        try
        {
            Animate1->FileName = sr.Name;
        }
```

```
        catch (Exception &e)
        {
            return;
        }
        RadioButton1->Enabled = true;
        RadioButton2->Enabled = true;
        Button1->Enabled = true;
    }
}

// щелчок на кнопке Выбрать
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    OpenDialog1->InitialDir = "";       // открыть каталог, из которого
                                        // запущена программа
    OpenDialog1->FileName = "*.avi";  // вывести список AVI-файлов

    if ( OpenDialog1->Execute())
    {
        // пользователь выбрал файл и нажал кнопку Открыть

        // Компонент Animate может отображать только простую,
        // не сопровождаемую звуком анимацию. Поэтому
        // при выполнении следующей инструкции возможна ошибка
        try
        {
            Animate1->FileName = OpenDialog1->FileName;
        }
        catch (Exception &e)
        {
            Edit1->Text = "";
            // сделаем недоступными кнопки управления
            RadioButton1->Enabled = false;
            RadioButton2->Enabled = false;
            Button1->Enabled = false;
            Button2->Enabled = false;
            Button3->Enabled = false;

            // сообщение об ошибке
            AnsiString msg =
                "Ошибка открытия файла " +
                OpenDialog1->FileName +
                "\nВозможно анимация сопровождается звуком.";
```

```
            ShowMessage(msg);
            return;
        }
        Edit1->Text = OpenDialog1->FileName; // отобразить имя файла

        RadioButton1->Checked = true;// режим просмотра — непрерывно
        Button1->Enabled = true;      // кнопка Пуск доступна
        Button2->Enabled = false;     // кнопка Предыдущий кадр недоступна
        Button3->Enabled = false;     // кнопка Следующий кадр недоступна

        RadioButton1->Enabled = true;
        RadioButton2->Enabled = true;
    }
}

// щелчок на кнопке Пуск/Стоп
void _ fastcall TForm1: :Button1Click (TObject *Sender)
{
    if (Animate1->Active)
    {
        // анимация отображается, щелчок на кнопке Стоп
        Animate1->Active = false;
        Button1->Caption = "Пуск";
        RadioButton2->Enabled = true;
    }
    else  // щелчок на кнопке Пуск
    {
        // активизировать отображение анимации
        Animate1->StartFrame = 1;                    // с первого кадра
        Animate1->StopFrame = Animate1->FrameCount; // по последний кадр
        Animate1->Active = true;
        Button1->Caption = "Стоп";
        RadioButton2->Enabled = false;
    }
}

// выбор режима просмотра всей анимации
void _ fastcall TForm1 : :RadioButton1Click (TObject *Sender)
{
    Button1->Enabled = true;  // кнопка Пуск/Стоп доступна
    //  сделать недоступными кнопки режима просмотра по кадрам
    Button2->Enabled = false;
    Button3->Enabled = false;

    Animate1->Active = false;
}
```

```
// выбор режима просмотра по кадрам
void__fastcall TForm1::RadioButton2Click(TObject *Sender)
{
    Button1->Enabled = false; // кнопка Пуск/Стоп недоступна

    Button2->Enabled = true;   // кнопка Следующий кадр доступна
    Button3->Enabled = false;  // кнопка Предыдущий кадр недоступна
    // отобразить первый кадр
    Animate1->StartFrame =1;
    Animate1->StopFrame = 1;
    Animate1->Active = true;
    CFrame = 1; // запомним номер отображаемого кадра
}

// щелчок на кнопке Следующий кадр
void__fastcall TForm1::Button2Click(TObject *Sender)
{
    CFrame++;
    // отобразить кадр
    Animate1->StartFrame = CFrame;
    Animate1->StopFrame = CFrame;
    Animate1->Active = true;

    if (CFrame > 1)
        Button3->Enabled = true;

    if (CFrame == Animate1->FrameCount) // отобразили последний кадр
        Button2->Enabled = false;      // кнопка Следующий кадр недоступна
}

// щелчок на кнопке Предыдущий кадр
void__fastcall TForm1::Button3Click(TObject *Sender)
{
    if (CFrame == Animate1->FrameCount) // последний кадр
        Button2->Enabled = true;

    CFrame--;

    // отобразить кадр
    Animate1->StartFrame = CFrame;
    Animate1->StopFrame = CFrame;
    Animate1->Active = true;

    if (CFrame == 1)
        Button3->Enabled = false;   // кнопка Следующий кадр недоступна
}
```

Компонент *MediaPlayer*

Компонент MediaPlayer обеспечивает воспроизведение звуковых файлов различных форматов (WAV, MID, RMI, MP3), полноценной, сопровождаемой звуком анимации и видеороликов (AVI) и музыкальных компакт-дисков.

Значок компонента MediaPlayer находится на вкладке **System** (рис. 4.3).

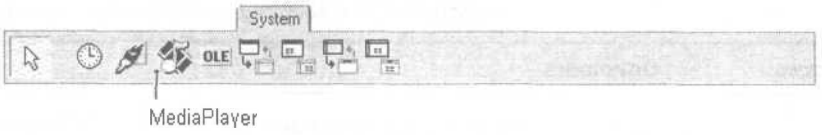


Рис. 4.3. Значок компонента MediaPlayer

Компонент MediaPlayer представляет собой группу кнопок (рис. 4.4), подобных тем, какие можно видеть на обычном аудио- или видеоплеере. Назначение этих кнопок пояснено в табл. 4.3. Свойства компонента MediaPlayer, доступные во время разработки формы, приведены в табл. 4.4.

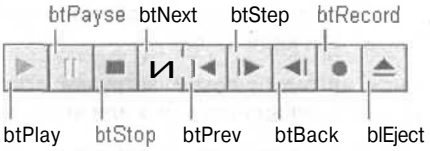


Рис. 4.4. Компонент MediaPlayer

Таблица 4.3. Кнопки компонента *MediaPlayer*

Кнопка	Обозначение	Действие
Воспроизведение	btPlay	Воспроизведение звука или видео
Пауза	btPause	Приостановка воспроизведения
Стоп	btstop	Остановка воспроизведения
Следующий	btNext	Переход к следующему кадру
Предыдущий	btPrev	Переход к предыдущему кадру
Шаг	btStep	Переход к следующему звуковому фрагменту, например, к следующей песне на CD
Назад	btBack	Переход к предыдущему звуковому фрагменту, например, к предыдущей песне на CD

Таблица 4.3 *(окончание)*

Кнопка	Обозначение	Действие
Запись	btRecord	Активизирует процесс записи
Открыть	btEject	Открывает CD-дисковод компьютера

Таблица 4.4. *Свойства компонента* `MediaPlayer`

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента и для управления работой плеера
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент MediaPlayer. Тип устройства задается именованной константой: dtAutoSelect — тип устройства определяется автоматически по расширению файла; dtVaweAudio — проигрыватель звука; `dtAVIVideo` — видеопроигрыватель; `dtCDAudio` — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
AutoOpen	Признак автоматической загрузки сразу после запуска программы, файла видеоролика или звукового фрагмента
Display	Определяет компонент, поверхность которого используется в качестве экрана для воспроизведения видеоролика (обычно в качестве экрана для отображения видео используют компонент Panel)
VisibleButtons	Составное свойство. Определяет видимые кнопки компонента. Позволяет сделать невидимыми некоторые кнопки

Помимо свойств, доступных в процессе разработки формы, компонент MediaPlayer предоставляет свойства, доступные во время работы программы (табл. 4.5), которые позволяют получить информацию о состоянии медиаплеера, воспроизводимом файле или треке Audio CD. Следует обратить внимание, что значения свойств, содержащих информацию о длительности, могут быть представлены в различных форматах. Наиболее универсальным форматом является формат `tfMilliseconds`, в котором длительность выражается в миллисекундах. Некоторые устройства поддерживают несколько форматов. Например, если MediaPlayer используется для воспроизведения Audio **CD,** то информация о воспроизводимом треке может быть представлена в формате tfTMSF (Track, Minute, Second, Frame — трек, минута, секунда, кадр). Для преобразования миллисекунд в минуты и секунды надо вос-

пользоваться известными соотношениями. Если значение свойства представлено в формате tfTMSF, то для преобразования можно воспользоваться макросами MCI_TMSF_TRACK, MCI_TMSF_SECOND и MCI_TMSF_MINUTE. Объявление этих и других полезных макросов можно найти в файле mmsystem.h.

Таблица 4.5. *Свойства компонента MediaPlayer, доступные во время работы программы*

Свойство	Описание
Length	Длина (время, необходимое для воспроизведения) открытого файла (например, WAV или AVI) или всех треков Audio CD
Tracks	Количество треков на открытом устройстве (количество композиций на Audio CD)
TrackLength	Длина (длительность) треков. Свойство представляет собой массив
Position	Позиция (время от начала) в процессе воспроизведения трека
TimeFormat	Формат представления значений свойств Length, TrackLength и Position. Наиболее универсальным является формат tfMilliseconds. Если медиаплеер представляет собой проигрыватель звуковых CD, то удобно использовать формат tfTMSF
Mode	Состояние устройства воспроизведения. Устройство может находиться в состоянии воспроизведения (mpPlaying). Процесс воспроизведения может быть остановлен (mpStopped) или приостановлен (mpPaused). Устройство может быть не готово к работе (mpNotReady) или в устройстве (CD-дисководе) может отсутствовать носитель (mpOpen)
Display	Экран — поверхность, на которой осуществляется отображение клипа. Если значение свойства не задано, то отображение осуществляется в отдельном, создаваемом во время работы программы окне
DisplayRect	Размер и положение области отображения клипа на поверхности экрана

Компонент MediaPlayer предоставляет методы (табл. 4.6), используя которые можно управлять работой медиаплеера из программы так, как будто это делает пользователь.

Таблица 4.6. *Методы компонента MediaPlayer*

Метод	Действие
Play ()	Активизирует процесс воспроизведения. Действие метода аналогично щелчку на кнопке **Play**

Таблица 4.6 *(окончание)*

Метод	Действие
Stop ()	Останавливает процесс воспроизведения
Pause ()	Приостанавливает процесс воспроизведения
Next ()	Переход к следующему треку, например к следующей композиции на Audio CD
Previous ()	Переход к предыдущему треку, например к следующей композиции на Audio CD
step ()	Переход к следующему кадру
Back ()	Переход к предыдущему кадру

Воспроизведение звука

В качестве примера использования компонента MediaPlayer для воспроизведения звука рассмотрим программу, используя которую, можно прослушать звуковые фрагменты, сопровождающие события Windows — такие, как начало и завершение работы, появление диалогового окна и др. Форма и диалоговое окно программы "Звуки Windows" представлены на рис. 4.5, текст — в листинге 4.2, а значения свойств компонента MediaPlayer1 — в табл. 4.7.

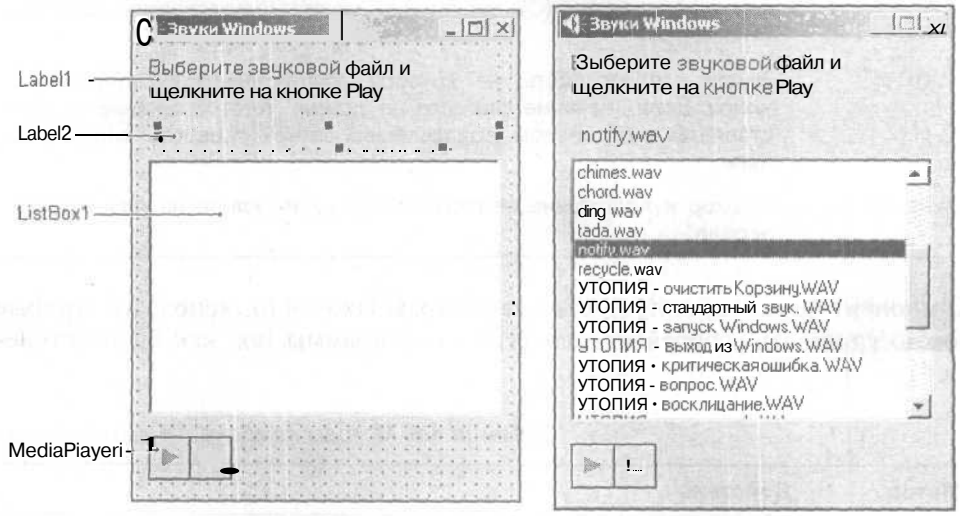


Рис. 4.5. Форма и диалоговое окно программы "Звуки Windows"

Таблица 4.7. *Значения свойств компонента Mediaplayer1*

Компонент	Значение
DeviceType	dtAutoSelect
VisibleButtons.btNext	false
VisibleButtons.BtPrev	false
VisibleButtons.btStep	false
VisibleButtons.btBack	false
VisibleButtons.btRecord	false
VisibleButtons.btEject	false

Помимо компонента MediaPlayer на форме находится компонент ListBox, который используется для выбора звукового файла, и два компонента Label, первый из которых используется для вывода информационного сообщения, второй — для отображения имени файла, выбранного пользователем.

Листинг 4.2. Использование компонента MediaPiayer для воспроизведения звука

```
void__fastcall TForm1::FormCreate(TObject *Sender)
{
    char *wd; // каталог Windows

    wd = (char*)AllocMem(MAX_PATH);
    GetWindowsDirectory(wd,MAX_PATH);
    SoundPath = wd;

    // звуковые файлы находятся в подкаталоге Media
    SoundPath = SoundPath + "\\Media\\";

    // сформируем список звуковых файлов
    TSearchRec sr;
    if (FindFirst( SoundPath + "*.wav", faAnyFile, sr) == 0)
    {
        // найден файл с расширением wav
        ListBox1->Items->Add(sr.Name); // добавим имя файла в список
        // еще есть файлы с расширением wav?
        while (FindNext(sr) == 0)
            ListBox1->Items->Add(sr.Name);
    }
```

```
    if (FindFirst( SoundPath + "*.mid", faAnyFile, sr) == 0)
    {
        // найден файл с расширением mid
        ListBox1->Items->Add(sr.Name); // добавим имя файла в список
        // еще есть файлы с расширением mid?
        while (FindNext(sr) == 0)
            ListBox1->Items->Add(sr.Name);
    }
    if (FindFirst( SoundPath + "*.rmi", faAnyFile, sr) == 0)
    {
        // найден файл с расширением rmi
        ListBox1->Items->Add(sr.Name); // добавим имя файла в список
        // еще есть файлы с расширением rmi?
        while (FindNext(sr) == 0)
            ListBox1->Items->Add(sr.Name);
    }
    // воспроизвести первый файл
    if ( ListBox1->Items->Count != 0)
    {
        Label2->Caption = ListBox1->Items->Strings[1];
        MediaPlayer1->FileName = SoundPath + ListBox1->Items->Strings[1];
        MediaPlayer1->Open();
        MediaPlayer1->Play();
    }
}

// щелчок на элементе списка
void __fastcall TForm1::ListBox1Click(TObject *Sender)
{
    Label2->Caption = ListBox1->Items->Strings[ListBox1->ItemIndex];
    MediaPlayer1->FileName = SoundPath + Label2->Caption;
    MediaPlayer1->Open();
    MediaPlayer1->Play();
}
```

Работает программа следующим образом. Сразу после запуска функция обработки события OnCreate формирует список звуковых файлов (WAV, MID и RMI), которые находятся в подкаталоге **Media** главного каталога Windows. Так как на разных компьютерах каталог, в который установлена операционная система, может называться по-разному, то для получения его имени используется API функция `GetWindowsDirectory`, значением которой является полное имя каталога Windows. Список звуковых файлов формируется

При помощи функций FindFirst и FindNext. Функция FindFirst обеспечивает поиск файла, удовлетворяющего критерию поиска, указанному при вызове функции. Функция FindNext продолжает процесс поиска. Обеим функциям в качестве параметра передается структура типа TSearchRec, поле Name которой (в случае успеха) содержит имя файла, удовлетворяющего критерию поиска. После того как список звуковых файлов сформирован, применением метода play активизируется процесс воспроизведения первого файла.

Щелчок на элементе списка обрабатывается функцией TForm1:: ListBoxlclick, которая присваивает значение свойству FileName компонента MediaPlayer1, при помощи метода open открывает выбранный файл и применением метода Play активизирует процесс воспроизведения.

Следующий пример показывает, как на основе компонента MediaPlayer можно создать вполне приличный проигрыватель компакт-дисков. Вид формы и диалогового окна программы приведен на рис. 4.6. Помимо компонентов, показанных на рисунке, в форме есть компонент MediaPiayer. Так как кнопки компонента MediaPiayer во время работы программы не используются (для управления плеером служат кнопки Button1, Button2 и Button3), свойству visible присвоено значение false, а сам компонент находится за границей формы.

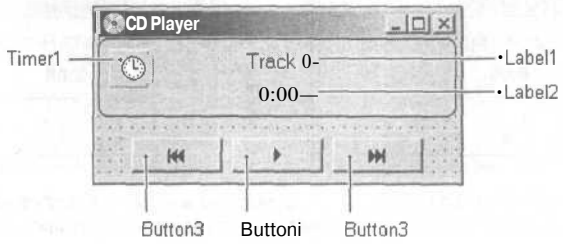


Рис. 4.6. Форма программы CD Player

Значки на кнопках управления - это текст, изображенный шрифтом Webdings. При использовании этого шрифта, например, цифре 4 соответствует значок Play. Соответствие значков Webdings и обычных символов отражает табл. 4.8.

Таблица 4.8. *Изображение символов шрифта Webdings*

Символ Webdings	Обычный символ — например, шрифт Arial	Код символа (шестнадцатеричный)
⏮	9	39
▶	4	34

Таблица 4.8 (окончание)

Символ Webdings	Обычный символ — например, шрифт Arial	Код символа (шестнадцатеричный)
⏭	:	3A
■	<	3C

Компонент Timer используется для организации цикла опроса состояния медиаплеера. Во время воспроизведения CD функция обработки события `OnTimer` выводит на индикатор (в поле метки `Label1`) номер трека и время воспроизведения.

Вид окна программы сразу после ее запуска в случае, если в CD-дисководе находится Audio CD, приведен на рис. 4.7. В случае, если в CD-дисководе диска нет или диск не звуковой, вместо информации о времени воспроизведения будет выведено сообщение "Вставьте Audio CD". Щелчок на кнопке **Play** (`Button1`) активизирует процесс воспроизведения. Во время воспроизведения на индикаторе отражается номер и длительность воспроизводимого трека, а также время от начала воспроизведения (рис. 4.8).

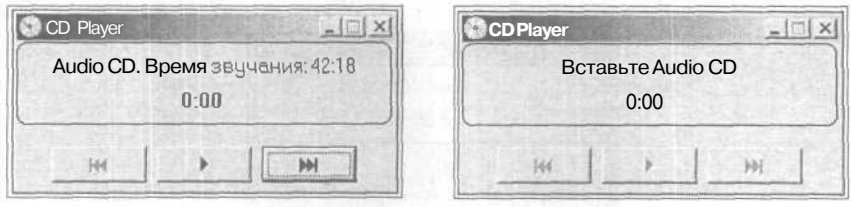


Рис. 4.7. В начале работы на индикаторе выводится информация о времени воспроизведения CD или сообщение о необходимости вставить в дисковод Audio CD

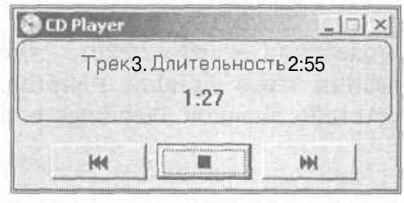


Рис. 4.8. Во время воспроизведения на индикаторе отображается информация о воспроизводимом треке

Текст программы приведен в листинге 4.3. Следует обратить внимание на событие Notify, которое может генерировать `MediaPlayer`. Событие Notify возникает в момент изменения состояния плеера при условии, что значение свойства Notify равно true. В рассматриваемой программе событие Notify

используется для обнаружения факта открытия CD-дисковода пользователем.

Листинг 4.3. Проигрыватель компакт дисков

```
tdefine Webdings // на кнопках плеера стандартные символы,
                 // изображение которых взято из шрифта Webdings

#ifdef Webdings
// "текст" на кнопках при использовании
// шрифта Webdings
#define PLAY      "4"
tdefine STOP      "<"
Idefine PREVIOUS  "9"
#define NEXT      ":"

#else
// текст на кнопках при использовании
// обычного шрифта, например, Arial
#define PLAY      "Play"
ttdefine STOP     "Stop"
#define PREVIOUS  "Previous"
#define NEXT      "Next"
#endif

// эти макросы обеспечивают перевод интервала времени,
// выраженного в миллисекундах в минуты и секунды
#define MINUTE(ms)  ((ms/1000)/60)
ttdefine SECOND(ms) ((ms/1000)%60)

// выводит в поле Label1 информацию о текущем треке
void__fastcall TForm1::TrackInfo()
{
    int ms; // время звучания трека, мсек
    AnsiString st;

    Track = MCI_TMSF_TRACK(MediaPlayer->Position);

    MediaPlayer->TimeFormat = tfMilliseconds;
    ms = MediaPlayer->TrackLength[Track];
    MediaPlayer->TimeFormat = tfTMSF;

    st = "Трек "+ IntToStr(Track);
    st = st + ". Длительность "+ IntToStr(MINUTE(ms));
    st = st + ":" + IntToStr(SECOND(ms));
```

```
    Label1->Caption = st;
}

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    Button1->Caption = PLAY;
    Button2->Caption = PREVIOUS;
    Button3->Caption = NEXT;
    MediaPlayer->Notify = true; // разрешить событие Notify
}

// изменение состояния плеера
void __fastcall TForm1::MediaPlayerNotify(TObject *Sender)
{
  switch ( MediaPlayer->Mode)
   {
    case mpOpen: // пользователь открыл дисковод
     {
        Button1->Enabled  =  false;
        Button1->Caption  =  PLAY;
        Button1->Tag  =  0;
        Button2->Enabled  =  false;
        Button3->Enabled  =  false;
        Label2->Caption  =  "00:00";

        /* по сигналу от таймера будем проверять
           состояние дисковода */
        Timer->Enabled  =  True;
     }
   }
  MediaPlayer->Notify  =  true;
}

// щелчок на кнопке Play/Stop
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if ( Button1->Tag == 0) {
    // щелчок на кнопке Play
    MediaPlayer->Play();
    Button1->Caption  =  STOP;
    Button1->Hint = "Стоп";
    Button1->Tag  =  1;
    Button3->Enabled  =  true; // доступна кнопка "следующий трек"
    MediaPlayer->Notify  =  true;
```

```
    Timer->Enabled = true;
    TrackInfo();
  }
  else {
    // щелчок на кнопке Stop
    Button1->Caption = PLAY;
    Button1->Hint = "Воспроизведение";
    Button1->Tag = 0;
    MediaPlayer->Notify = true;
    MediaPlayer->Stop();
    Timer->Enabled = false;
  }
}

// сигнал от таймера: вывести номер трека
// и время воспроизведения
void _fastcall TForm1::TimerTimer(TObject *Sender)
{
  int trk;          // трек
  int min, sec;     // время
  AnsiString st;

  if ( MediaPlayer->Mode == mpPlaying) // режим воспроизведения
  {
    // получить номер воспроизводимого трека
    trk = MCI_TMSF_TRACK(MediaPlayer->Position);

    if ( trk != Track) // произошла смена трека
    {
      TrackInfo();
      Track = trk;
      if ( Track == 2)
          Button2->Enabled = true;  // доступна кнопка "пред.трек"
      if ( Track == MediaPlayer->Tracks)
          Button3->Enabled = false; // кнопка "след.трек" недоступна
    }

    // вывод информации о воспроизводимом треке
    min = MCI_TMSF_MINUTE(MediaPlayer->Position);
    sec = MCI_TMSF_SECOND(MediaPlayer->Position);
    st.printf("%d:%.2d",min,sec);
    Label2->Caption = st;
    return;
  }
```

```
  /* Если дисковод открыт ими в нем нет
    Audio CD, то Mode == mpOpen.
    Ждем диск, т. е. до тех пор, пока не будет
    Mode == mpStopped + кол-во треков > 1 */
  if ( (MediaPlayer->Mode == mpStopped) &&
     (MediaPlayer->Tracks > 1) )
  {
    // диск вставлен
    Timer->Enabled = false;
    Button1->Caption = PLAY;
    Button1->Enabled = true;
    Button1->Tag = 0;
    MediaPlayer->Notify = true;

    // получить информацию о времени звучания CD
    MediaPlayer->TimeFormat = tfMilliseconds;

    int ms = MediaPlayer->Length;
    AnsiString st = "Audio CD. Время звучания: ";

    st = st + IntToStr(MINUTE(ms));
    st = st + ":" + IntToStr(SECOND(ms));
    Label1->Caption = st;

    MediaPlayer->TimeFormat = tfTMSF;
    Label1->Visible ,= true;
    Track = 0;
    return;
  }

  // дисковод открыт или в дисководе не Audio CD
  if ( ( MediaPlayer->Mode == mpOpen) | |
      (MediaPlayer->Mode == mpStopped) && (MediaPlayer->Tracks == 1))
  {
    Label1->Caption = "Вставьте Audio CD";
    if ( Label1->Visible)
        Label1->Visible = false;
    else Label1->Visible = true;
  }
}

// щелчок на кнопке "следующий трек"
void _ fastcall TForm1: :Button3Click (TObject *Sender)

<
  MediaPlayer->Next();
```

```
  // если перешли к последнему треку, то кнопку Next
  // сделать недоступной
  if ( MCI_TMSF_TRACK(MediaPlayer->Position) == MediaPlayer->Tracks)
    Button3->Enabled = false;
  if ( ! Button2->Enabled) Button2->Enabled = true;
  TrackInfo();
  Label2->Caption = "0:00";
}

// щелчок на кнопке "предыдущий трек"
void _ fastcall TForm1: :Button2Click (TObject *Sender)
{
    MediaPlayer->Previous(); // в начало текущего трека
    MediaPlayer->Previous(); // в начало предыдущего трека
    if ( MCI_TMSF_TRACK(MediaPlayer->Position) == 1)
      Button2->Enabled = false;
    if ( ! Button3->Enabled)
      Button3->Enabled = true;
    TrackInfo();
    Label2->Caption = "0:00";
}

// пользователь закрыл окно программы
void _ fastcall TForm1: : FormClose (TObject *Sender, TCloseAction &Action)
{
    MediaPlayer->Stop();
    MediaPlayer->Close();
}
```

Просмотр видеороликов

Как было сказано выше, компонент MediaPlayer позволяет просматривать видеоролики и сопровождаемую звуком анимацию. В качестве примера использования компонента рассмотрим программу Video Player, при помощи которой можно просмотреть небольшой ролик или анимацию. Вид формы программы Video Player приведен на рис. 4.9.

Компонент OpenDialog1 обеспечивает отображение стандартного диалогового окна **Открыть файл** для выбора файла. Окно **Открыть файл** становится доступным во время работы программы в результате щелчка на кнопке **Eject** (SpeedButton1). Следует обратить внимание, что для управления процессом воспроизведения кнопки компонента MediaPlayer1 не используются, поэтому **свойству** Visible **компонента** MediaPiayeri присвоено **Значение** false.

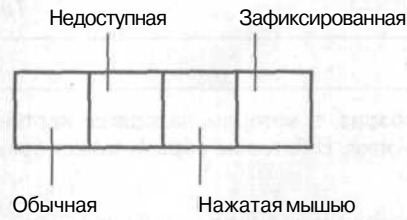


Рис. 4.11. Структура битового образа Glyph: картинки, соответствующие состоянию кнопки

Подготовить битовый образ для кнопки SpeedButton можно при помощи любого графического редактора, в том числе и Image Editor. После запуска Image Editor надо выбрать команду **File** | New | **Bitmap File** и в появившемся окне **Bitmap Properties** задать размер битового образа (например, 19x76 – если размер картинки на кнопке должен быть 19×19 пикселов). В качестве примера на рис. 4.12 приведен битовый образ для кнопки **Play** (SpeedButton2). Границы кнопок показаны условно. Обратите внимание, что изображение нажатой и зафиксированной кнопок смещены вверх и вправо относительно изображений нажатой и недоступных кнопок. Сделано это для того, чтобы во время работы программы у пользователя не создавалось впечатление, что кнопка "ушла".

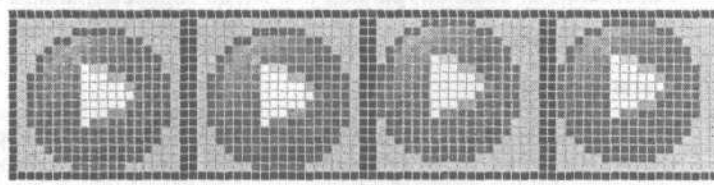


Рис. 4.12. Битовый образ для кнопки **Play**

В табл. 4.10 приведены значения свойств компонентов SpeedButton1 и speedButton2 разрабатываемой программы. Текст программы Video Player приведен в листинге 4.4.

Таблица 4.10. *Значение свойств компонентов SpeedButtoni И SpeedButton2*

Свойство	SpeedButtoni	SpeedButton2
Gliph	ejbtn.bmp	plbtn.bmp
NumGliphs	4	4
Flat	true	true

Таблица 4.10 (окончание)

Свойство	SpeedButton1	SpeedButton2
AllowUp	false	true
GroupIndex	0	1
Enabled	true	false
Width	25	25
Height	25	25
ShowHint	true	true
Hint	Eject	Play

Листинг 4.4. Видеоплеер

```
// обработка события Create
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    MediaPlayer1->Display = Form1; // отображение ролика на поверхности
                                   // формы
}

// возвращает размер кадра AVI-файла
void __fastcall GetFrameSize(AnsiString f, int *w, int *h)
{
    // в заголовке AVI-файла есть информация о размере кадра*
    struct {
        char      RIFF[4]; // строка RIFF
        long int  nu_1[5]; // не используется
        char      AVIH[4]; // строка AVIH
        long int  nu_2[9]; // неиспользуется
        long int  w;       // ширина кадра
        long int  h;       // высота кадра
    } header;

    TFileStream *fs;    // поток (для чтения заголовка файла)

    /* операторы объявления потока и его создания
    можно объединить: TFileStream *fs = new TFileStream(f,fmOpenRead); */

    fs = new TFileStream(f,fmOpenRead);    // открыть поток для чтения
    fs->Read(&header, sizeof(header));     // прочитать заголовок файла
```

```
    *w = header.w;
    *h = header.h;
    delete fs;
}

// щелчок на кнопке Eject (выбор видеоклипа)
void _ fastcall TForm1: : SpeedButton1Click (TObject *Sender)
{
    OpenDialog1->Title = "Выбор клипа";
    OpenDialog1->InitialDir = "";
    OpenDialog1->FileName = "*.avi";
    if ( ! OpenDialog1->Execute())
        return; // пользователь нажал кнопку Отмена

     /* При попытке открыть файл клипа, который уже открыт,
       возникает ошибка. */
    if ( MediaPlayer1->FileName == OpenDialog1->FileName)
        return;

    /* Пользователь выбрал клип. Зададим размер и положение "экрана",
       на который будет выведен клип. Для этого надо знать размер
       кадров клипа. */

    int fw, fh;    // размер кадра клипа
    int top, left; // левый верхний угол экрана
    int sw, sh;    // размер экрана (ширина, высота)

    int mw, mh;    // максимально возможный размер экрана
                   // (определяется текущим размером формы)

    float kw, kh; // коэф-ты масштабирования кадра по ширине и высоте
    float k;      // коэфф-т масштабирования кадра

    GetFrameSize(OpenDialog1->FileName, &fw, &fh);// получить размер кадра

    // вычислим максимально возможный размер кадра
    mw = Form1->ClientWidth;
    mh = Form1->SpeedButton1->Top-10;

    if ( fw < mw)
       kw = 1; // кадр по ширине меньше размера экрана
    else kw = (float) mw / fw;

    if ( fh < mh)
       kh = 1; // кадр по высоте меньше размера экрана
    else kh = (float) mh / fh;
```

```
    // масштабирование должно быть пропорциональным
    if ( kw < kh)
        k = kw;
    else k = kh;

    // здесь масштаб определен
    sw = fw * k; // ширина экрана
    sh = fh * k; // высота экрана

    left = (Form1->ClientWidth- sw) / 2;
    top = (SpeedButton1->Top - sh) / 2;

    MediaPlayer1->FileName = OpenDialog1->FileName;
    MediaPlayer1->Open();
    MediaPlayer1->DisplayRect = Rect(left,top,sw,sh);

    SpeedButton2->Enabled = True; // кнопка Play теперь доступна

    /* если размер кадра выбранного клипа меньше размера
       кадра предыдущего клипа, то экран (область формы)
       надо очистить */

    Form1->Canvas->FillRect(Rect(0,0,ClientWidth,SpeedButton1->Top));

    // активизируем процесс воспроизведения
    MediaPlayer1->Play();
    SpeedButton2->Down = True;
    SpeedButton2->Hint = "Stop";
    SpeedButton1->Enabled = False;
}

// щелчок на кнопке Play/Stop (воспроизведение/стоп)
void __fastcall TForm1::SpeedButton2Click(TObject *Sender)
{
    if (SpeedButton2->Down)
    {
        // нажата кнопка Play
        MediaPlayer1->Play();
        SpeedButton2->Hint = "Stop";
        SpeedButton1->Enabled = False; // кнопка Eject недоступна
    }
    else
    {
        // нажата кнопка Stop
        MediaPlayer1->Stop();
        SpeedButton2->Hint = "Play";
```

```
        SpeedButton1->Enabled = True; // кнопка Eject доступна
    }
}

// сигнал от плеера
void __fastcall TForm1::MediaPlayer1Notify(TObject *Sender)
{
    if ( ( MediaPlayer1->Mode == mpStopped) && ( SpeedButton2->Down) )
    {
        SpeedButton2->Down = False;    // "отжать" кнопку Play
        SpeedButton2->Hint = "Play";
        SpeedButton1->Enabled = True; // сделать доступной кнопку Eject
    }
}
```

Следует обратить внимание на следующее. В качестве экрана, на котором осуществляется воспроизведение видеороликов, используется поверхность формы. Поэтому установить значение свойства Display компонента MediaPlayer1 во время разработки формы нельзя. Кроме того, размер экрана должен быть равен или пропорционален размеру кадров ролика. Значение свойства Display устанавливает функция обработки события Create для формы, а размер и положение экрана на форме — функция обработки события Click на кнопке **Eject** (SpeedButton1). Размер экрана устанавливается максимально возможным и таким, чтобы ролик воспроизводился без искажения (высота и ширина экрана пропорциональны высоте и ширине кадров). Размер кадров ролика возвращает функция GetFrameSize, которая извлекает нужную информацию из заголовка файла.

Создание анимации

Процесс создания файла анимации (AVI-файла) рассмотрим на примере. Пусть надо создать анимацию, которая воспроизводит процесс рисования эскиза Дельфийского храма (окончательный вид рисунка представлен на рис. 4.13, несколько кадров анимации — на рис. 4.14).

Для решения поставленной задачи можно воспользоваться популярной программой Macromedia Flash.

В Macromedia Flash анимация, которую так же довольно часто называют "роликом" (Movie), состоит из *слоев*. В простейшем случае ролик представляет собой один-единственный слой (Layer). *Слой* — это последовательность кадров (Frame), которые в процессе воспроизведения анимации выводятся последовательно, один за другим. Если ролик состоит из нескольких слоев, то кадры анимации формируются путем наложения кадров одного слоя на кадры другого. Например, один слой может содержать изображение фона,

на котором разворачивается действие, а другой — изображение персонажей. Возможность формирования изображения путем наложения слоев существенно облегчает процесс создания анимации. Таким образом, чтобы создать анимацию, нужно распределить изображение по слоям и для каждого слоя создать кадры.

Рис. 4.13. Эскиз Дельфийского храма

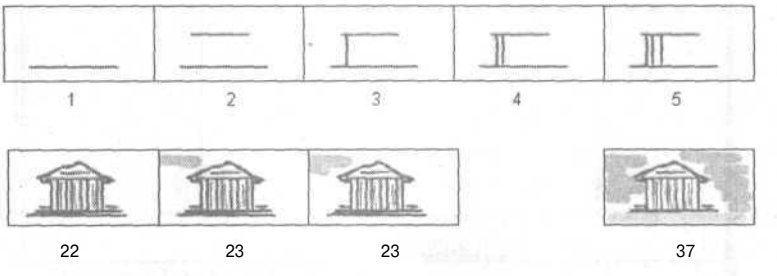


Рис. 4.14. Кадры анимации процесса рисования Дельфийского храма

После запуска Macromedia Flash на фоне главного окна программы появляется окно **Moviel** (рис. 4.15), которое используется для создания анимации. В верхней части окна, которая называется Timeline, отражена структура анимации, в нижней части, которая называется рабочей областью, находится изображение текущего кадра выбранного слоя. После запуска Macromedia Flash анимация состоит из одного слоя (Layer 1), который, в свою очередь, представляет собой один пустой (чистый) кадр.

Перед тем как приступить непосредственно к созданию кадров анимации, нужно задать общие характеристики анимации (ролика) — размер кадров и скорость их воспроизведения. Характеристики вводятся в поля диалогового окна **Movie Properties** (рис. 4.16), которое появляется в результате выбора из меню **Modify** команды **Movie.** В поле **Frame Rate** нужно ввести скорость воспроизведения ролика, измеряемую в "кадрах в секунду" (fps — frame per second, кадров в секунду), в поля **Width и Height** — ширину и высоту кадров. В этом же окне можно выбрать фон кадров (список **Background Color).**

После того как установлены характеристики ролика, можно приступить к созданию кадров анимации.

Первый кадр нужно просто нарисовать. Технология создания изображений в Macromedia Flash обычная: используется стандартный набор инструментов: кисть, карандаш, пульверизатор, резинка и другие инструменты.

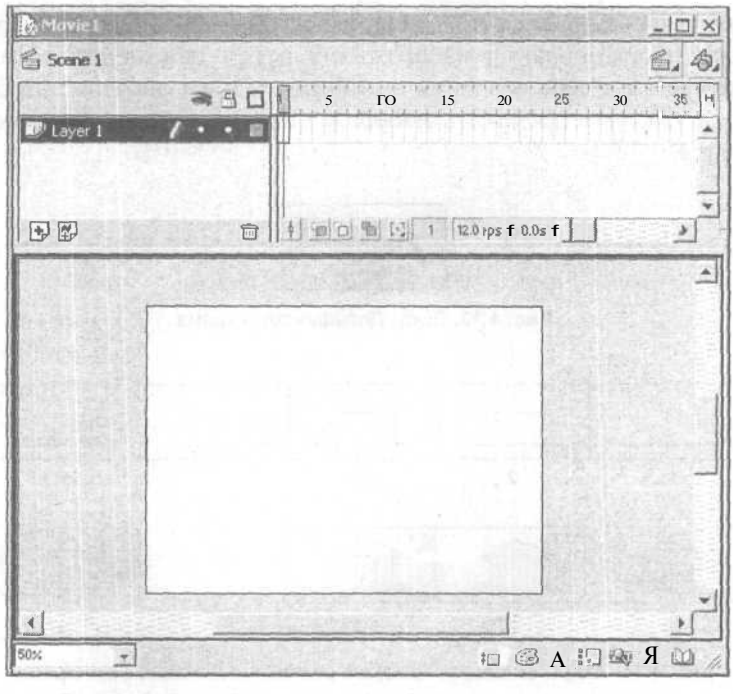


Рис. 4.15. Окно **Movie** *в* начале работы над новой анимацией

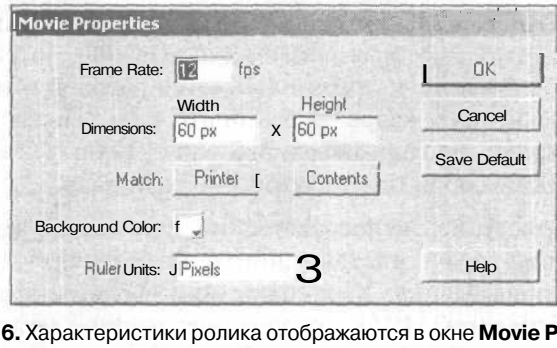


Рис. 4.16. Характеристики ролика отображаются в окне **Movie Properties**

Чтобы создать следующий кадр, нужно из меню **Insert** выбрать команду **Keyframe.** В результате в текущий слой будет добавлен кадр, в который будет скопировано содержимое предыдущего кадра (в большинстве случаев следующий кадр создается путем изменения предыдущего). Теперь можно нарисовать второй кадр. Аналогичным образом создаются остальные кадры анимации.

Иногда не нужно, чтобы новый кадр содержал изображение предыдущего. В этом случае вместо команды **Keyframe** надо воспользоваться командой **Blank Keyframe.**

Если некоторое изображение должно оставаться статичным в течение времени, кратного выводу нескольких кадров, то вместо того чтобы вставлять в слой несколько одинаковых кадров (Keyframe), можно сделать кадр статичным. Если кадр, изображение которого должно быть статичным, является последним кадром ролика, то в окне Timeline нужно выделить кадр, до которого изображение должно оставаться статичным, и из меню **Insert** выбрать команду **Frame.** Если кадр, изображение которого должно быть статичным, не является последним, то нужно выделить этот кадр и несколько раз из меню **Insert** выбрать команду **Frame.**

Можно значительно облегчить работу по созданию анимации, если разделить изображение на основное и фоновое, поместив каждое в отдельный слой (именно так поступают при создании мультфильмов). Сначала надо создать кадры слоя фона так, как было описано выше. Затем, выбрав из меню **Insert** команду **Layer,** нужно добавить слой основного действия.

Следует обратить внимание, что все действия по редактированию изображения направлены на текущий кадр выбранного слоя. В списке слоев выбранный слой выделен цветом, номер текущего кадра помечен маркером – красным квадратиком.

Чтобы выводимая анимация сопровождалась звуком, нужно сначала сделать доступным соответствующий звуковой файл. Для этого надо из меню **File** выбрать команду **Import** и добавить в проект звуковой файл (рис. 4.17).

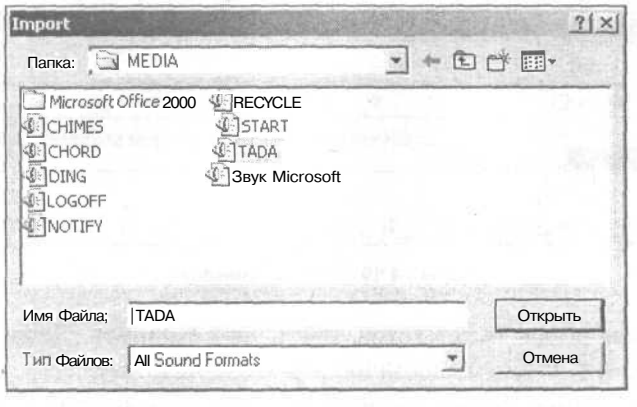


Рис. 4.17. Импорт звукового файла

Затем в окне Timeline нужно выделить кадр, при отображении которого должно начаться воспроизведение звукового фрагмента, используя диалоговое окно **Sound** (рис. 4.18), выбрать звуковой фрагмент и задать, если надо, параметры его воспроизведения. Количество повторов нужно ввести в поле **Loops,** эффект, используемый при воспроизведении, можно выбрать из списка **Effect.**

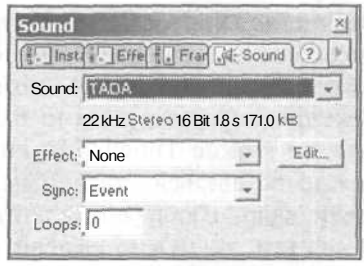


Рис. 4.18. Диалоговое окно **Sound**

В качестве примера на рис. 4.19 приведен вид окна Timeline в конце работы над анимацией. Анимация состоит из двух слоев. Слой Layer 2 содержит фон. Детали фона появляются постепенно, в течение 9 кадров. После этого фон не меняется, поэтому 9 кадр является статичным. Слой Layer 1 — слой основного действия, которое начинается после того, как будет выведен фон. Вывод анимации заканчивается стандартным звуком TADA (его длительность равна одной секунде). Начало воспроизведения звука совпадает с выводом последнего (49-го, если считать от начала ролика) кадра основного действия, поэтому этот кадр сделан статичным в течение вывода следующих 12 кадров (скорость вывода анимации — 12 кадров в секунду). Сделано это для того, чтобы процесс вывода анимации завершился одновременно с окончанием звукового сигнала.

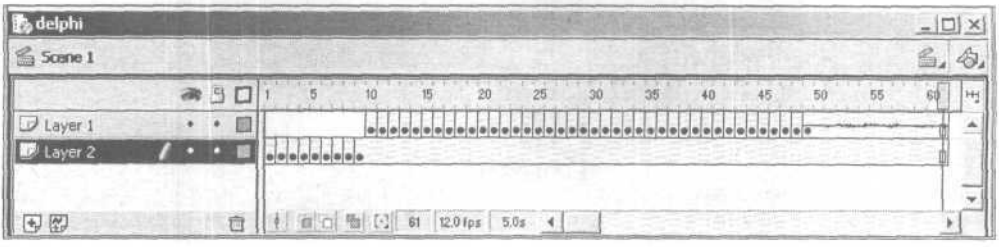


Рис. 4.19. Пример анимации

После того как ролик будет готов, его надо сохранить. Делается это обычным образом, т. *е.* выбором из меню **File** команды Save.

Для преобразования файла из формата Macromedia Flash в AVI-формат нужно из меню **File** выбрать команду **Export Movie** и задать имя файла. Затем в появившемся диалоговом окне **Export Windows AVI** (рис. 4.20) нужно задать размер кадра (поля width и Height), из списка **Video Format** выбрать формат, в котором будет записана видеочасть ролика, а из поля **Sound Format** — формат звука.

Если установлен переключатель **Compress video,** то после, щелчка на кнопке **OK** появится диалоговое окно, в котором можно будет выбрать один из

стандартных методов сжатия видео. При выборе видео и звукового формата нужно учитывать, что чем более высокие требования будут предъявлены к качеству записи звука и изображения, тем больше места на диске займет AVI-файл. Здесь следует иметь в виду, что завышенные требования не всегда оправданы.

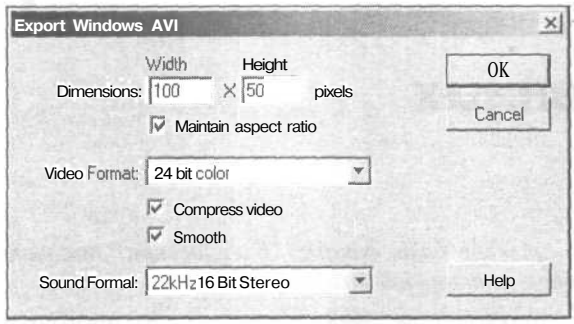
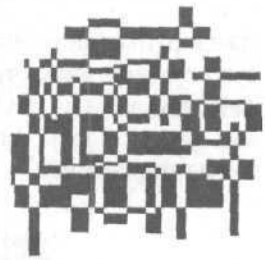


Рис. 4.20. Диалоговое окно **Export Windows AVI**



ГЛАВА 5

Базы данных

В этой главе на примере базы данных "Ежедневник" показан процесс создания приложения работы с локальной базой данных.

С точки зрения пользователя, *база данных* — это программа, которая обеспечивает работу с информацией. При запуске такой программы на экране, как правило, появляется таблица, просматривая которую можно найти нужные сведения. Если система позволяет, то пользователь может внести изменения в базу данных, например, добавить новую информацию или удалить ненужную.

С точки зрения программиста, *база данных* — это набор файлов, в которых находится информация. Разрабатывая базу данных для пользователя, программист создает программу, которая обеспечивает работу с файлами данных.

В состав C++ Builder включены компоненты, используя которые программист может создавать программы работы с файлами данных в форматах dBase, Microsoft Access, Infomix и Oracle и др.

База данных и СУБД

База данных — это набор, совокупность файлов, в которых находится информация. Программная система *(приложение),* обеспечивающая работу с базой данных (файлами данных) называется *системой управления базой данных* (СУБД). Следует обратить внимание, что вместо термина СУБД часто используется термин *база данных,* при этом файлы данных и СУБД рассматриваются как единое целое.

Локальные и удаленные базы данных

В зависимости от расположения программы, которая использует данные, и самих данных, а также от способа разделения данных между несколькими пользователями различают *локальные* и *удаленные* базы данных.

Данные локальной базы данных (файлы данных) локализованы, т. е. находятся на одном устройстве, в качестве которого может выступать диск компьютера или сетевой диск (диск другого компьютера, работающего в сети). Локальные базы данных не обеспечивают одновременный доступ к информации нескольким пользователям. Для обеспечения *разделения данных* (доступа к данным) между несколькими пользователями (программами, работающими на одном или разных компьютерах) в локальных базах данных используется метод, получивший название "блокировка файлов". Суть этого метода заключается в том, что пока данные используются одним пользователем, другой пользователь не может работать с этими данными, т. е. данные для него закрыты, заблокированы. Несомненным достоинством локальной базы данных является высокая скорость доступа к информации. Приложения работы с локальной базой данных и саму базу данных часто размещают на одном компьютере. dBase, Paradox, FoxPro и Microsoft Access — это локальные базы данных.

Удаленные базы данных строятся по технологии "клиент-сервер". Программа работы с удаленной базой данных состоит из двух частей: клиентской и серверной. Клиентская часть программы работает на компьютере пользователя и обеспечивает взаимодействие с серверной программой посредством запросов, передаваемых на удаленный компьютер (сервер), обеспечивая тем самым доступ к данным. Серверная часть программы, работающая на удаленном компьютере, принимает запросы, выполняет их и пересылает данные клиентской программе. Профамма, работающая на удаленном сервере, проектируется так, чтобы обеспечить одновременный доступ к базе данных нескольким пользователям. При этом для обеспечения доступа к данным вместо механизма блокировки файлов используют механизм *транзакций,* Транзакция — это последовательность действий, которая должна быть обязательно выполнена над данными перед тем, как они будут переданы. В случае обнаружения ошибки во время выполнения любого из действий вся последовательность действий, составляющая транзакцию, повторяется снова. Таким образом, механизм транзакций обеспечивает защиту от аппаратных сбоев. Он также обеспечивает возможность многопользовательского доступа к данным. Oracle, Infomix, Microsoft SQL Server и InterBase — это удаленные базы данных.

Структура базы данных

База данных — это набор однородной и, как правило, упорядоченной по некоторому критерию информации. База данных может быть представлена в "бумажном" или в "компьютерном" виде.

Типичным примером "бумажной" базы данных является каталог библиотеки — набор бумажных карточек, содержащих информацию о книгах. Информация в этой базе однородная (содержит сведения только о книгах) и

упорядоченная (карточки расставлены, например, в алфавитном порядке фамилий авторов). Другими примерами "бумажной" базы данных являются телефонный справочник и расписание движения поездов.

Компьютерная база данных представляет собой файл (или набор связанных файлов), содержащий информацию, который часто называют *файлом данных.* Файл данных состоит из *записей.* Каждая запись содержит информацию об одном экземпляре. Например, каждая запись базы данных "Ежедневник" содержит информацию только об одном экземпляре — запланированном мероприятии или задаче.

Записи состоят из полей. Каждое поле содержит информацию об одной характеристике экземпляра. Например, запись базы данных "Ежедневник" может состоять из полей: "Задача", "Дата" и "Примечание ". "Задача", "Дата" и "Примечание " - это имена полей. Содержимое полей характеризует конкретную задачу.

Следует обратить внимание, что каждая запись состоит из одинаковых полей. Некоторые поля могут быть не заполнены, однако они все равно присутствуют в записи.

На бумаге базу данных удобно представить в виде таблицы. Каждая строка таблицы соответствует записи, а ячейка таблицы — полю. При этом заголовок столбца таблицы — это имя поля, а номер строки таблицы — номер записи.

Информацию компьютерных баз данных обычно выводят на экран в виде таблиц. Поэтому часто вместо словосочетания "файл данных" используют словосочетание "таблица данных" или просто "таблица".

Псевдоним

Разрабатывая программу работы с базой данных, программист не знает, на каком диске и в каком каталоге будут находиться файлы базы данных во время ее использования. Например, пользователь может поместить базу данных в один из каталогов диска C:, D: или на сетевой диск. Поэтому возникает проблема передачи в программу информации о месте нахождения файлов базы данных.

В C++ Builder проблема передачи в программу информации о месте нахождения файлов базы данных решается путем использования *псевдонима* базы данных. Псевдоним (Alias) — это имя, поставленное в соответствие реальному, полному имени каталога базы данных. Например, псевдонимом каталога C:\data\Petersburg может быть имя Peterburg. Программа работы с базой данных для доступа к данным использует не реальное имя каталога, а псевдоним. Псевдоним базы данных можно создать при помощи утилиты BDE Administrator. Информация о всех зарегистрированных в системе псевдонимах хранится в специальном файле.

Компоненты доступа и манипулирования данными

Обычно для доступа и манипулирования данными используется соответствующая СУБД. Однако часто возникает необходимость получить доступ к информации, которая находится в базе данных, из прикладной программы. Решить эту задачу можно при помощи *компонентов доступа к данным*.

C++ Builder предоставляет в распоряжение программиста компоненты, используя которые можно построить приложение, обеспечивающее работу практически с любой базой данных.

Компоненты доступа к данным находятся во вкладках **BDE, Data Access, ADO и InterBase.** Компоненты вкладок **BDE** и **Data Access** для доступа к данным используют *процессор баз данных* Borland Database Engine (BDE), реализованный в виде набора динамических библиотек и драйверов. Компоненты вкладки ADO для доступа к данным используют разработанную Microsoft технологию ADO (ActiveX Data Object — ADO). Компоненты вкладки **InterBase** обеспечивают непосредственный доступ к данным InterBase.

Наиболее универсальным механизмом доступа к базам данных является механизм, реализованный на основе BDE. Драйверы, входящие в состав BDE, обеспечивают доступ как к локальным базам данных (Paradox, Access, dBASE), так и к удаленным серверам баз данных (Microsoft SQL Server, Oracle, Infomix). Набор драйверов, включенных в BDE, определяется вариантом C++ Builder.

Создание базы данных

Процесс создания базы данных рассмотрим на примере. Создадим локальную базу данных "Ежедневник", которая представляет собой одну-единственную таблицу в формате Paradox. Для этого воспользуемся поставляемой вместе с C++ Builder утилитой Database Desktop.

Запустить Database Desktop можно из C++ Builder, выбрав в меню **Tools** команду **Database Desktop,** или из Windows (команда **Пуск | Программы | C++Builder \ Database Desktop).**

Процесс создания базы данных состоит из двух шагов: сначала надо создать *псевдоним* базы данных, затем — таблицу (в общем случае — несколько таблиц). Псевдоним (Alias) определяет расположение таблиц базы данных и используется для доступа к ним.

Для того чтобы создать псевдоним, надо:

1. В меню **Tools** выбрать команду **Alias Manager.**
2. В появившемся диалоговом окне **Alias Manager** щелкнуть на кнопке New.

3. Ввести в поле **Database alias** псевдоним создаваемой базы данных — например, organizer,
4. Ввести в поле **Path** путь к файлам таблиц базы данных (таблицы будут созданы на следующем шаге).
5. Щелкнуть на кнопке **Keep** New (рис. 5.1).

Теперь можно приступить к созданию таблицы.

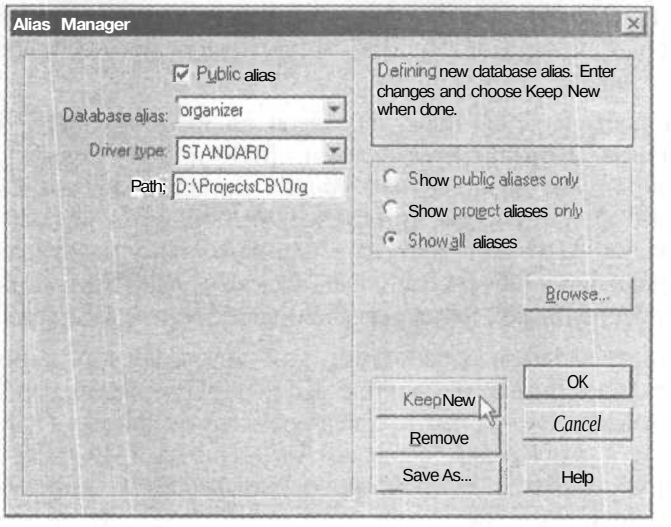


Рис. 5.1. Создание псевдонима базы данных

Чтобы создать таблицу, надо в меню **File** выбрать команду New | **Table** (рис. 5.2), затем в появившемся диалоговом окне **Create Table** — тип *таблицы* (рис. 5.3).

В результате выполнения перечисленных выше действий открывается окно **Create Table,** в котором надо определить структуру таблицы — задать имена полей базы данных и указать их тип и размер (рис. 5.4).

Записи базы данных "Ежедневник" состоят из двух полей: Task_F и Date_F. Поле Task_F (символьного типа) содержит название задачи (мероприятия), поле Date_F (типа Date) — дату, не позднее которой *задача* должна быть выполнена (дату проведения мероприятия).

Имена полей вводят в столбец **Field Name**, тип — в столбец **Type.** При записи имени поля можно использовать латинские буквы и цифры. При этом следует учитывать, что имя поля не должно совпадать ни с одним из ключевых слов языка SQL (таких, например, как WHEN или SELECT). Тип поля определяет тип данных, которые могут быть помещены *в* поле. Задается тип поля при помощи одной из приведенных в табл. 5.1 констант. Константа,

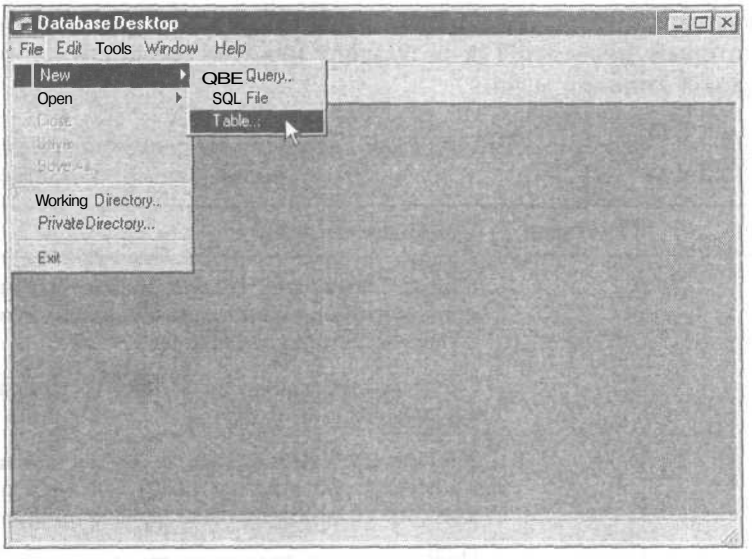


Рис. 5.2. Начало работы над новой таблицей

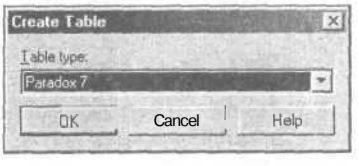


Рис. 5.3. В списке **Table type** надо выбрать тип создаваемой таблицы (файла данных)

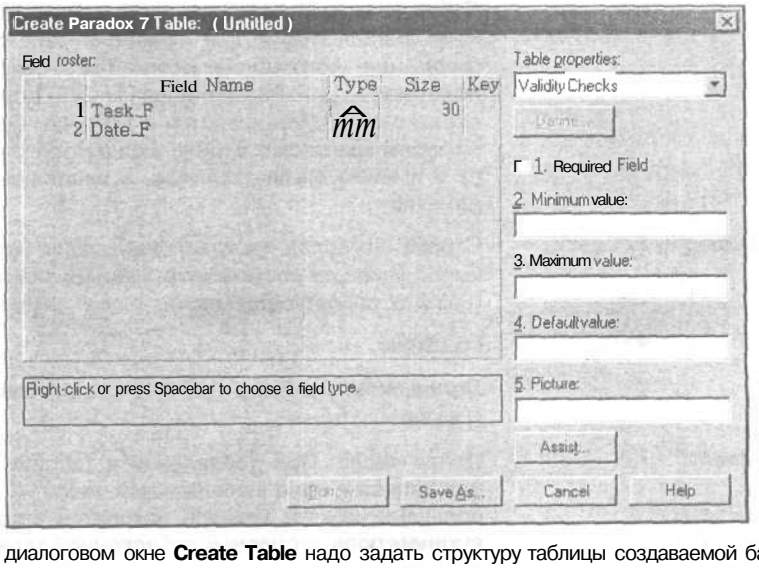


Рис. 5.4. В диалоговом окне **Create Table** надо задать структуру таблицы создаваемой базы данных

определяющая тип поля, может быть введена с клавиатуры или выбором в списке, который появляется в результате нажатия клавиши "пробел" или щелчка правой кнопкой мыши.

Таблица 5.1. *Тип поля определяет тип информации, которая может в нем находиться*

Тип поля	Константа	Содержимое поля
Alpha	A	Строка символов. Максимальная длина строки определяется характеристикой **Size,** значения которой находятся в диапазоне 1—255
Number	N	Число из диапазона 10^{-307}—10^{308} с 15-ю значащими цифрами
Money	$	Число в денежном формате. Цифры числа делятся на группы при помощи разделителя групп разрядов. Так же выводится знак денежной единицы
Short	S	Целое число из диапазона от –32767 до 32767
Long Integer	I	Целое число из диапазона от –2 147 483 648 до 2 147 483 647
Date	D	Дата
Time	T	Время, отсчитываемое от полуночи, выраженное в миллисекундах
Timestamp	@	Время и дата
Memo	M	Строка символов произвольной длины. Поле типа Memo используется для хранения текстовой информации, которая не может быть сохранена в поле типа Alpha. Размер поля (1—240) определяет, сколько символов хранится в таблице. Остальные символы хранятся в файле, имя которого совпадает с именем файла таблицы, а расширение файла — mb
Formated Memo	F	Строка символов произвольной длины (как у типа Memo). Имеется возможность указать тип и размер шрифта, способ оформления и цвет символов
Graphic	G	Графика
Logical	L	Логическое значение "истина" (true) или "ложь" (false)
Autoincrement	+	Целое число. При добавлении в таблицу очередной записи в поле записывается число на единицу большее, чем то, которое находится в соответствующем поле последней добавленной записи

***Таблица 5.1** (окончание)*

Тип поля	Константа	Содержимое поля
Bytes	Y	Двоичные данные. Поле этого типа используется для хранения данных, которые не могут быть интерпретированы Database Desktop
Binary	B	Двоичные данные. Поле этого типа используется для хранения данных, которые не могут быть интерпретированы Database Desktop. Как и данные типа Memo, эти данные не находятся в файле таблицы. Поля типа Binary, как правило, содержат аудиоданные

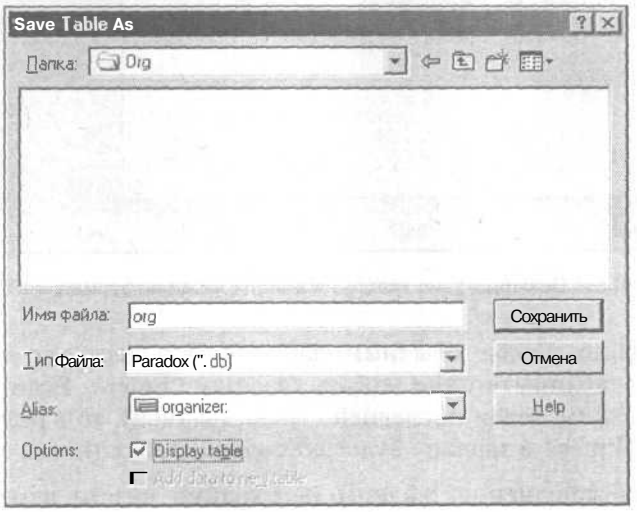


Рис. 5.5. Сохранение таблицы базы данных

После того как будут определены все поля, надо щелкнуть на кнопке **Save** As. На экране появится диалоговое окно Save **Table** As (рис. 5.5). В нем нужно выбрать (в списке **Alias)** псевдоним базы данных, элементом которой является сохраняемая таблица, в поле **Имя файла** ввести имя файла таблицы, установить переключатель **Display table** и щелкнуть на кнопке **Сохранить.** В результате в указанном каталоге (псевдоним связан с конкретным каталогом локального или сетевого диска) будет создан файл таблицы и на экране появится диалоговое окно **Table** (рис. 5.6), в котором можно ввести данные в только что созданную таблицу (базу данных). Следует обратить внимание, что по умолчанию Database Desktop открывает таблицы в *режиме просмотра,* и для того чтобы внести изменения в таблицу (добавить, удалить или

изменить запись), необходимо, выбрав в меню **Table** команду **Edit** (или нажав клавишу <F8>), активизировать режим редактирования таблицы.

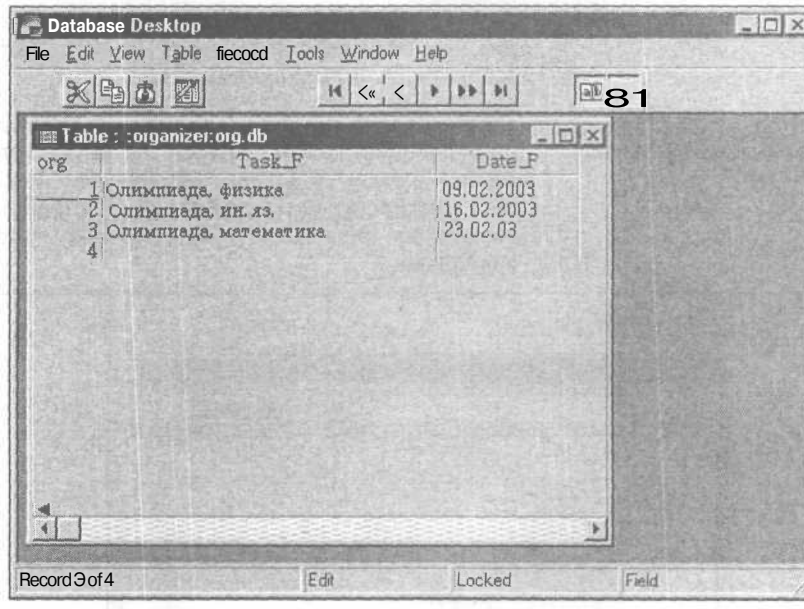


Рис. 5.6. Database Desktop можно использовать для ввода информации в базу данных

Данные в таблицу вводят обычным образом. Для перехода к следующему полю (столбцу таблицы) нужно нажать клавишу <Enter>. Если текущее поле является последним полем последней строки (записи), то в результате нажатия клавиши <Enter> в таблицу будет добавлена строка (новая запись).

Если во время заполнения таблицы необходимо внести изменения в уже заполненное поле, то надо, используя клавиши перемещения курсора, выбрать это поле и нажать клавишу <F2>.

Если при вводе данных в таблицу буквы русского алфавита отображаются неверно, то надо изменить шрифт, который используется для отображения данных. Для этого нужно в меню **Edit** выбрать команду **Preferences,** затем, в появившемся диалоговом окне во вкладке **General** щелкнуть на кнопке **Change.** В результате этих действий откроется диалоговое окно **Change Font** (рис. 5.7), в котором надо выбрать русифицированный шрифт TrueType. Следует обратить внимание, что в Microsoft Windows 2000 (Microsoft Windows XP) используются шрифты типа Open Type, в то время как программа Database Desktop ориентирована на работу со шрифтами TrueType. Поэтому в списке шрифтов нужно выбрать русифицированный шрифт именно TrueType. После выбора шрифта необходимо завершить работу с Database

Desktop, т. к. внесенные в конфигурацию изменения будут действительны только после перезапуска утилиты.

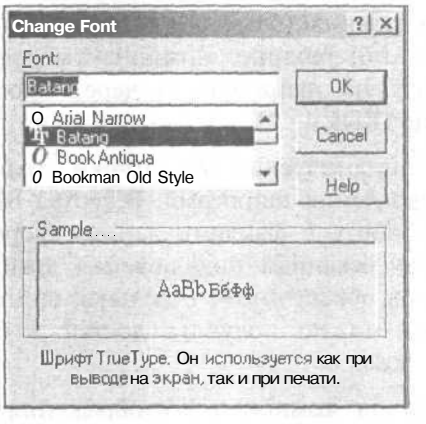


Рис. 5.7. Для правильного отображения данных в Database Desktop нужно выбрать русифицированный шрифт TrueType

Доступ к базе данных

Доступ к базе данных обеспечивают компоненты Database, Table, Query и DataSource. Значки этих компонентов находятся на вкладках **Data Access** и **BDE** (рис. 5.8).

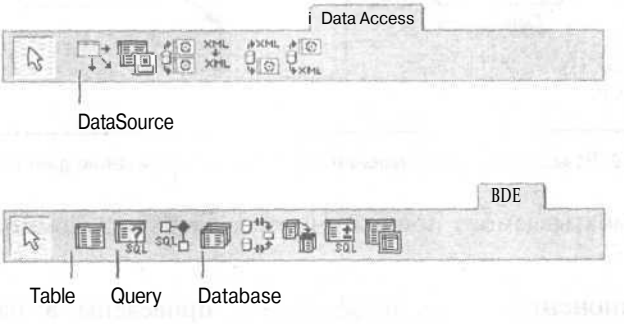


Рис. 5.8. Компоненты вкладок **Data Access** и **BDE** обеспечивают доступ к данным

Компонент Database представляет базу данных как единое целое, т. е. как совокупность таблиц, а компонент Table — как одну из таблиц базы данных. Компонент DataSource (источник данных) обеспечивает связь между компонентом отображения-редактирования данных (например, компонент DBGrid) и источником данных, в качестве которого может выступать таблица

(компонент Table) или результат выполнения SQL-запроса *к* таблице (компонент Query). Компонент DataSource позволяет оперативно выбирать источники данных, использовать один и тот же компонент (например, DBGrid) для отображения всей таблицы (базы данных) или только результата выполнения SQL-запроса к этой таблице. Компоненты доступа к данным обращаются к базе данных не напрямую, а через процессор баз данных - Borland Database Engine (BDE).

Ядро BDE образуют динамические библиотеки, реализующие механизмы обмена данными и управления запросами. В состав BDE включены драйверы, обеспечивающие работу с файлами данных форматов Paradox, dBase, FoxPro. Имеется также механизм подключения драйверов ODBC. Доступ к данным SQL серверов обеспечивает отдельная система драйверов — SQL Links. С их помощью можно получить доступ к базам данных Oracle, Infomix, Sysbase и Interbase.

Механизм взаимодействия компонента отображения-редактирования данных (DBGrid) С Данными (Table ИЛИ Query) Через компонент DataSource ПОКазан на рис. 5.9.

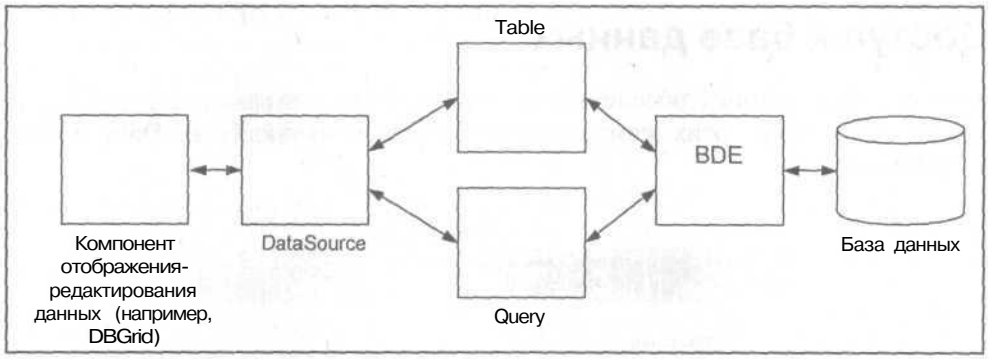


Рис. 5.9. Взаимодействие компонентов доступа-отображения данных и BDE

В форму разрабатываемого приложения надо добавить компоненты Table и DataSource.

Свойства компонентов `Table` и DataSource приведены в табл. 5.2 и 5.3. Свойства перечислены в том порядке, *в* котором рекомендуется устанавливать их значения.

Значения свойств `DatabaseName` и `TableName` задаются путем выбора из списков. В списке DatabaseName перечислены все *зарегистрированные на данном компьютере* псевдонимы, а в списке TabieName — имена файлов таблиц, которые находятся в соответствующем псевдониму каталоге.

Таблица 5.2. *Свойства компонента* `Table`

Свойство	Определяет
`DatabaseName`	Имя базы данных, частью которой является таблица (файл данных), для доступа к которой используется компонент. В качестве значения свойства следует использовать псевдоним базы данных
`TableName`	Имя файла данных (таблицы данных), для доступа к которому используется компонент
`TableType`	Тип таблицы. Таблица может быть набором данных в формате Paradox (ttParadox), dBase (ttDBase), FoxPro (ttFoxPro) или другого типа. По умолчанию значение свойства равно ttDefault — это означает, что тип таблицы будет определен на основе информации, которая находится в файле таблицы
Active	Признак активизации файла данных (таблицы). В результате присваивания свойству значения true файл таблицы будет открыт

Таблица 5.3. *Свойства компонента DataSource*

Свойство	Определяет
`Name`	Имя компонента. Используется для доступа к свойствам компонента
DataSet	Компонент, представляющий входные данные (таблица или запрос)

Свойство DataSet компонента DataSource обеспечивает возможность выбора источника данных, а также связь между компонентом, представляющим данные (таблица или запрос), и компонентом отображения данных. Например, большая база данных может быть организована как набор таблиц одинаковой структуры. В этом случае в приложении работы с базой данных каждой таблице будет соответствовать свой компонент Table, а выбор конкретной таблицы можно осуществить установкой значения свойства DataSet.

Компоненты доступа к базе данных являются невизуальными и во время работы программы на форме не видны. Поэтому их можно поместить в любую точку формы (рис. 5.10).

Значения свойств компонентов `Table1` и DataSource1 приложения "Ежедневник" приведены в табл. 5.4 и 5.5.

Таблица 5.4. *Значения свойств компонента* `Table1`

Свойство	Значение
Name	Table1
DatabaseName	organizer
TableName	org.db
Active	false

Таблица 5.5. *Значения свойств компонента* `DataSource1`

Свойство	Значение
Name	DataSource1
DataSet	Table1

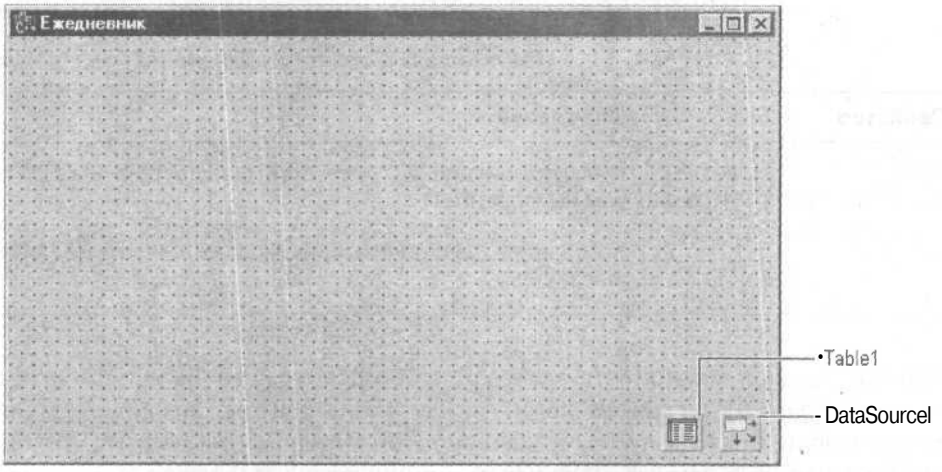


Рис. 5.10. Форма после добавления компонентов Table и DataSource

Отображение данных

Пользователь может просматривать базу данных в *режиме формы* или *в режиме таблицы*. В режиме формы можно видеть только одну запись, а в режиме таблицы — несколько записей одновременно. Часто эти два режима комбинируют. Краткая информация (содержимое некоторых ключевых полей) выводится в табличной форме, а при необходимости увидеть содержимое всех полей выполняется переключение в режим формы.

Компоненты, обеспечивающие отображение и редактирование полей записей базы данных, находятся на вкладке **Data Controls** (рис. 5.11).

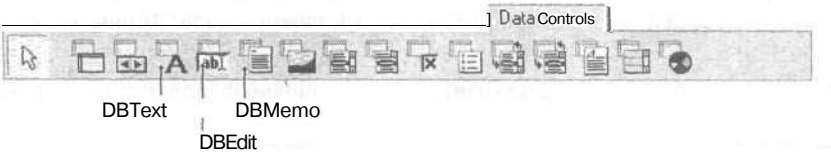


Рис. 5.11. Компоненты отображения и редактирования полей

Компонент DBText обеспечивает отображение содержимого отдельного поля, а компоненты DBEdit и DBMemo — отображение и редактирование. В табл. 5.6 перечислены некоторые свойства этих компонентов. Свойства перечислены в том порядке, в котором следует устанавливать их значения.

Таблица 5.6. *Свойства компонентов DBText, DBEdit и* DBMemo

Свойство	Определяет
DataSource	Источник данных (компонент Table или Query)
DataField	Поле записи, для отображения или редактирования которого используется компонент

Для обеспечения просмотра базы данных в режиме таблицы используется компонент DBGrid. Свойства компонента DBGridl определяют вид таблицы и действия, которые могут быть выполнены над данными во время работы программы. В табл. 5.7 перечислены некоторые свойства компонента DBGrid.

Таблица 5.7. *Свойства компонента DBGrid*

Свойство	Определяет
DataSource	**Источник данных (компонент** Table **или** Query)
Columns	**Отображаемая информация (поля записей)**
Options.dgTitles	Разрешает вывод строки заголовка столбцов
Options.dgIndicator	Разрешает вывод колонки индикатора. Во время работы с базой данных текущая запись помечается в колонке индикатора треугольником, новая запись — звездочкой, редактируемая — специальным значком
Options.dgColumnResize	Разрешает менять во время работы программы ширину колонок таблицы

Таблица 5,7 *(окончание)*

Свойство	Определяет
Options.dgColLines	Разрешает выводить линии, разделяющие колонки таблицы
Options.dgRowLines	Разрешает выводить линии, разделяющие строки таблицы

В диалоговом окне программы "Ежедневник" данные отображаются в режиме таблицы. Поэтому в форму надо добавить компонент DBGridi и установить значения его свойств в соответствии с табл. 5.8.

Таблица 5.8. *Значения свойств компонента DBGridi*

Свойство	Значение
DataSource	DataSource1

Как было сказано ранее, свойство Colunns компонента DBGrid определяет поля, содержимое которых будет отображено в таблице DBGrid. Свойство columns является сложным свойством и представляет собой массив элементов типа TColumn. Свойства элементов массива определяют поля, содержимое которых будет в таблице, а так же вид колонок (табл. 5.9).

Таблица 5.9. *Свойства объекта TColumn*

Свойство	Определяет
FieldName	Поле, содержимое которого отображается в колонке
width	Ширину колонки в пикселах
Font	Шрифт, используемый для вывода текста в ячейках колонки
Color	Цвет фона колонки
Alignment	Способ выравнивания текста в ячейках колонки. Текст может быть выровнен по левому краю (taLeftJustify), по центру (taCenter) или по правому краю (taRightJustify)
Title.Caption	Заголовок колонки. Значением по умолчанию является имя поля записи
Title.Alignment	Способ выравнивания заголовка колонки. Заголовок может быть выровнен по левому краю (taLeftJustify), по центру (taCenter) или по правому краю (taRightJustify)
Title.Color	Цвет фона заголовка колонки
Title.Font	Шрифт заголовка колонки

По умолчанию компонент DBGrid содержит одну колонку. Чтобы добавить в компонент DBGrid еще одну колонку, надо в окне **Object Inspector** выбрать свойство `Colums` компонента DBGrid, щелкнуть на кнопке с тремя точками, а затем в появившемся окне **Editing** — на кнопке Add New (рис. 5.12). После этого, используя Object Inspector, надо установить значения свойств элементов массива columns.

Выбрать настраиваемую колонку (ее свойства отражаются в окне **Object Inspector)** можно в окне **Editing** или в окне **Object TreeView**.

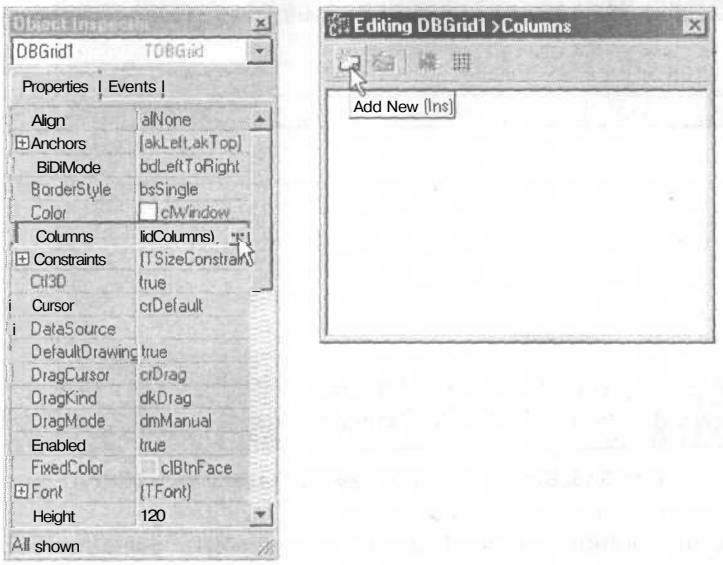


Рис. 5.12. Чтобы добавить колонку в компонент DBGrid, щелкните в строке **Coiums** на кнопке с тремя точками, затем — на кнопке **Add New**

В простейшем случае для каждой колонки достаточно установить значение свойства FieldName, которое определяет поле, содержимое которого отображается в колонке, а также значение свойства `Title.Caption`, определяющее заголовок колонки. В табл. 5.10 приведены значения свойств компонента `DBGrid1`, а на рис. 5.13 — вид формы после настройки компонента.

Таблица 5.10. *Значения свойств компонента* `DBGrid1`

Свойство	Значение
`Columns[0].FieldName`	`Date_F`
`Columns[0].TitleCaption`	**Когда**
`Columns[0].Title.Font.Style.Italic`	`true`

Таблица 5.10 (окончание)

Свойство	Значение
Columns[1].FieldName	Task_F
Columns[1].TitleCaption	Что
Columns[1].Title.Font.Style.Italic	true

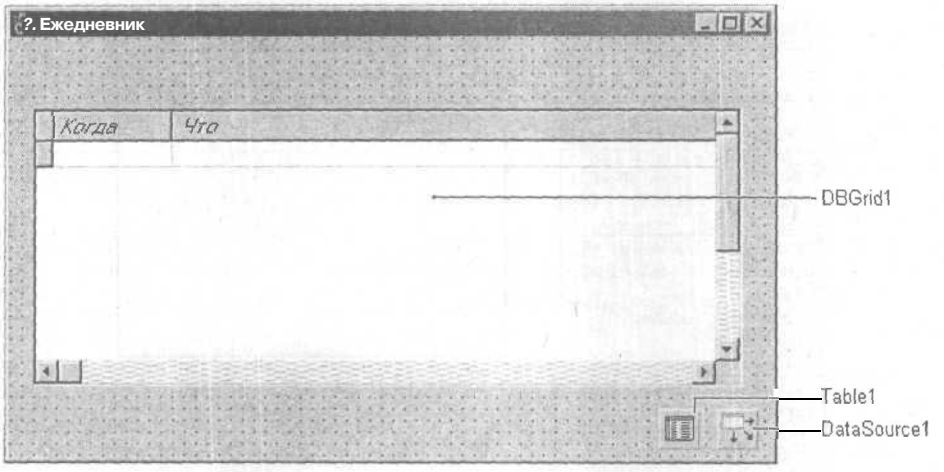


Рис. 5.13. Вид формы после настройки компонента DBGrid

Если после настройки компонента DBGrid присвоить значение true свойству Active компонента Table1, то в поле компонента DBGrid будет выведено содержимое базы данных.

Манипулирование данными

Для того чтобы пользователь мог не только просматривать базу данных (решение этой задачи в рассматриваемой программе обеспечивает компонент DBGrid), но и редактировать ее, в форму приложения надо добавить компонент DBNavigator, значок которого находится на вкладке **Data Controls** (рис. 5.14). Компонент DBNavigator (рис. 5.15) представляет собой набор командных кнопок, обеспечивающих перемещение указателя текущей записи к следующей, предыдущей, первой или последней записи базы данных, а также добавление в базу данных новой записи и удаление текущей записи.

Табл. 5.11 содержит описания действий, которые выполняются в результате щелчка на соответствующей кнопке компонента DBNavigator. Свойства компонента DBNavigator перечислены в табл. 5.12.

Рис. 5.14. Значок компонента DBNavigator находится на вкладке **Data Controls**

Рис. 5.15. Компонент DBNavigator

Таблица 5.11. 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	Активизирует 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 5.12. Свойства компонента *DBNavigator*

Свойство	Определяет
DataSource	Компонент, являющийся источником данных. В качестве источника данных может выступать база данных (компонент Database), таблица (компонент Table) или результат выполнения запроса (компонент Query)
VisibleButtons	Видимые командные кнопки

Следует обратить внимание на свойство VisibleButtons. Оно позволяет скрыть некоторые кнопки компонента DBNavigator и тем самым запретить выполнение соответствующих операций над файлом данных. Например, присвоив значение false свойству VisibleButtons.nbDelete, можно скрыть кнопку nbDeiete и тем самым запретить удаление записей.

На рис. 5.16 приведен вид формы приложения "Ежедневник" после добавления Компонента DBNavigator. СВОЙСТВУ DataSource Компонента DBNavigatorl следует ПрИСВОИТЬ Значение Tablel.

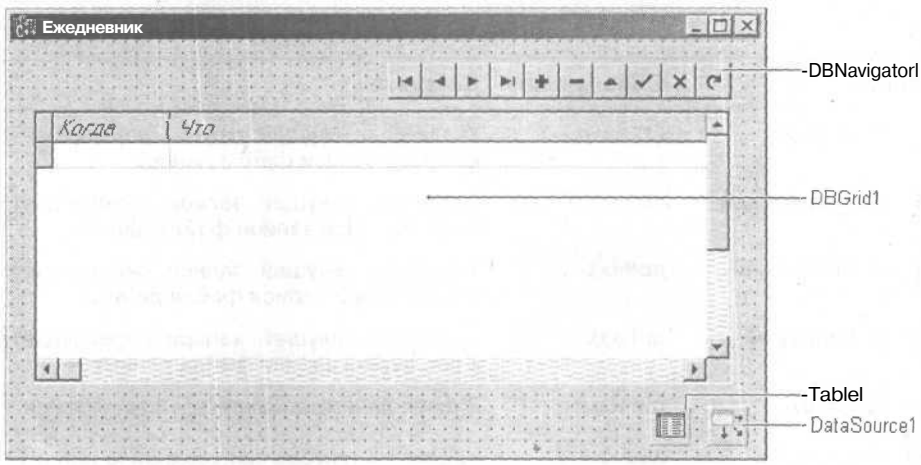


Рис. 5.16. Форма приложения после добавления компонента DBNavigator

После этого программу можно откомпилировать и запустить. Следует обратить внимание, что для того чтобы после запуска программы в окне появилась информация или, если база данных пустая, можно было вводить новую информацию, свойство Active таблицы-источника данных должно иметь значение true.

Работа с базой данных, представленной в виде таблицы, во многом похожа на работу с электронной таблицей Microsoft Excel. Используя клавиши перемещения курсора вверх и вниз, а также клавиши листания текста страницами (<Page Up> и <Page Down>), можно, перемещаясь от строки к строке, просматривать записи базы данных. Нажав клавишу <Ins>, можно добавить запись, а нажав клавишу <Del> — удалить. Для того чтобы внести изменения в поле записи, нужно, используя клавиши перемещения курсора влево и вправо, выбрать необходимое поле и нажать клавишу <F2>.

Выбор информации из базы данных

При работе с базой данных пользователя, как правило, интересует не все ее содержимое, а некоторая конкретная информация. Найти нужные сведения можно последовательным просмотром записей. Однако такой способ поиска неудобен и малоэффективен.

Большинство систем управления базами данных позволяют выполнять *выборку* нужной информации путем выполнения *запросов*. Пользователь формулирует запрос, указывая критерий, которому должна удовлетворять интересующая его информация, а система выводит записи, удовлетворяющие запросу.

Для выборки из базы данных записей, удовлетворяющих некоторому критерию, предназначен компонент Query (рис. 5.17).

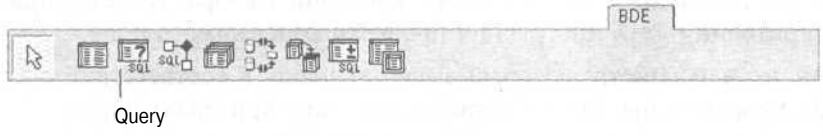


Рис. 5.17. Компонент Query

Компонент Query, как и компонент Table, представляет собой записи базы данных, но в отличие от последнего он представляет не всю базу данных (все записи), а только ее часть — записи, удовлетворяющие критерию запроса.

В табл. 5.13 перечислены некоторые свойства компонента Query.

***Таблица 5.13.** Свойства компонента Query*

Свойство	Определяет
Name	Имя компонента. Используется компонентом DataSource для связи результата выполнения запроса (набора записей) с компонентом, обеспечивающим просмотр записей, например `DBGrid`
SQL	Записанный на языке SQL запрос к базе данных (к таблице)
Active	При присвоении свойству значения `true` активизируется процесс выполнения запроса
`RecordCount`	Количество записей, удовлетворяющих критерию запроса

Для того чтобы во время разработки программы задать, какая информация должна быть выделена из базы данных, в свойство SQL надо записать запрос — команду на языке SQL (Structured Query Language, язык структурированных запросов).

В общем виде SQL-запрос на выборку данных из базы данных (таблицы) выглядит так:

```
SELECT СписокПолей
FROM Таблица
WHERE
(Критерий)
ORDER BY СписокПолей
```

где:

- SELECT — команда "выбрать из таблицы записи и вывести содержимое полей, имена которых указаны в списке";
- FROM — параметр команды, который определяет имя таблицы, из которой нужно сделать выборку;
- WHERE — параметр, который задает критерий выбора. В простейшем случае критерий — это инструкция проверки содержимого поля;
- ORDER BY — параметр, который задает условие, в соответствии с которым будут упорядочены записи, удовлетворяющие критерию запроса.

Например, запрос

```
SELECT Date_F, Task_F
FROM ':organizer:org.db'
WHERE ( Date_F = '09.02.2003')
ORDER BY Date_F
```

обеспечивает выборку записей из базы данных organizer (из таблицы org.db), у которых в поле Date_F находится текст 09.02.2003, т. *е.* формирует список мероприятий, назначенных на 9 февраля 2003 года.

Другой пример. Запрос

```
SELECT Date_F, Task_F
FROM ':organizer:org.db'
WHERE
( Date_F >= '10.02.2003') AND ( Date_F <='16.02.2003')
ORDER BY Date_F
```

формирует список дел, назначенных на неделю (с 10 по 16 февраля 2003 года).

Запрос может быть сформирован и записан в свойство SQL компонента Query во время разработки формы или во время работы программы.

Для записи запроса в свойство SQL во время разработки формы используется редактор списка строк (рис. 5.18), окно которого открывается в результате

щелчка на кнопке с тремя точками в строке свойства SQL (в окне **Object Inspector).**

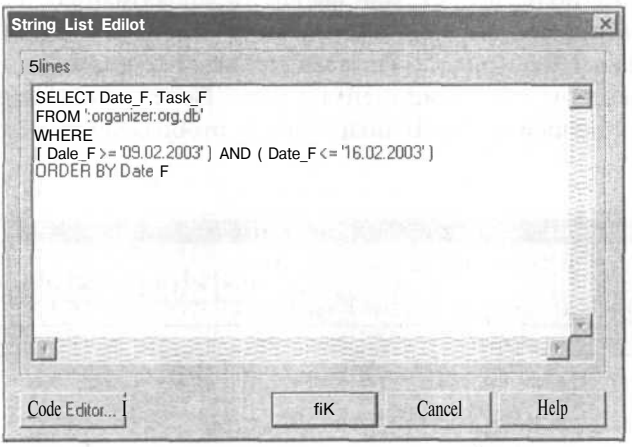


Рис. 5.18. Ввод SQL-запроса во время разработки формы приложения

Сформировать запрос во время работы профаммы можно при помощи метода Add, применив его к свойству SQL компонента Query.

Ниже приведен фрагмент кода, который формирует запрос (т. е. записывает текст запроса в свойство SQL компонента Query) на выбор информации из таблицы org базы данных organizer. Предполагается, что строковая переменная today (ТИП `AnsiString`) содержит дату В формате `dd/mm/yyyy`.

```
Form1->Query1->SQL->Add("SELECT Date_F, Task_F");
Form1->Query1->SQL->Add("FROM ':organizer:org.db'");
Form1->Query1->SQL->Add("WHERE (Date_F = '" + today + "')");
Form1->Query1->SQL->Add("ORDER BY Date_F");
```

Если запрос записан в свойство SQL компонента Query во время разработки формы приложения, то во время работы профаммы критерий запроса можно изменить простой заменой соответствующей строки текста запроса.

Например, для запроса:

```
SELECT Date_F, Task_F
FROM ':organizer:org.db'
WHERE
 ( Date_F= '09.02.2003')
ORDER BY Date_F
```

инструкция замены критерия выглядит так:

```
Query1->SQL->Strings[3] = "(Date_F = '" + tomorrow + "')";
```

Следует обратить внимание на то, что свойство SQL является структурой типа TStrings, в которой строки нумеруются с нуля.

Для того чтобы пользователь мог выбирать информацию из базы данных, в форму разрабатываемого приложения надо добавить кнопки **Сегодня, Завтра, Эта неделя** и **Все** (рис. 5.19). Назначение этих кнопок очевидно. Также в форму добавлены два компонента Label. Поле `Label1` используется для отображения текущей даты. В поле `Label2` отображается режим просмотра базы данных.

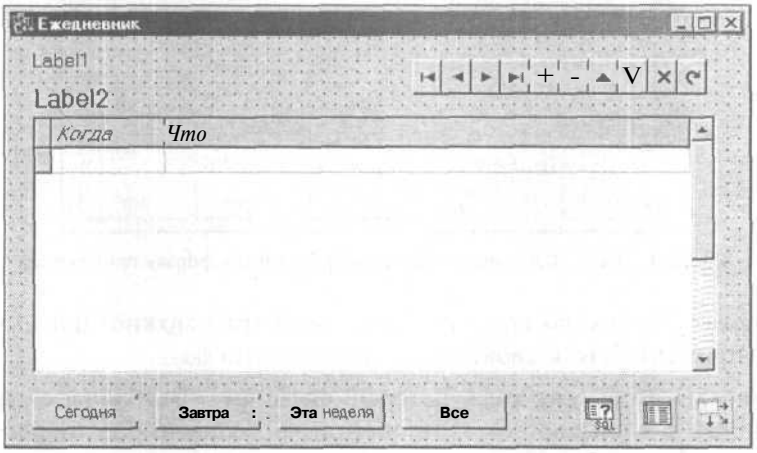


Рис. 5.19. Окончательный вид формы

Функции обработки события click на кнопках **Сегодня, Завтра и Эта неделя** приведены в листинге 5.1. Каждая из этих функций изменяет соответствующим образом сформированный во время разработки формы SQL-запрос. Для получения текущей даты функции обращаются к стандартной функции NOW, которая возвращает текущую дату и время. Преобразование даты в строку символов выполняет стандартная функция FormatDateTime.

Листинг 5.1. Обработка события Click на кнопках Сегодня, Завтра и Эта неделя

```
// Щелчок на кнопке Сегодня
void__fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString today = FormatDateTime("dd/mm/yyyy",Now());

    Form1->Label2->Caption = "Сегодня";

    // изменить критерий запроса
    Query1->SQL->Strings[3] = "(Date_F = '" + today + "')";
```

```
    // выполнить запрос
    Form1->Query1->Open();
    Form1->DataSource1->DataSet = Form1->Query1;

    if ( ! Form1->Query1->RecordCount)
    {
        ShowMessage("На сегодня никаких дел не запланировано!");
    }
}

// щелчок на кнопке Завтра
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    AnsiString tomorrow = FormatDateTime("dd/mm/yyyy", Now() +1);

    Form1->Label2->Caption = "Завтра";

    // изменить критерий запроса
    Query1->SQL->Strings[3] = " (Date_F= '" + tomorrow + "')";

    // выполнить запрос
    Form1->Query1->Open();

    Form1->DataSource1->DataSet = Form1->Query1;

    if ( ! Form1->Query1->RecordCount)
    {
        ShowMessage("На завтра никаких дел не запланировано!");
    }
}

// щелчок на кнопке Эта неделя
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // от текущего дня до конца недели (до воскресенья)
  TDateTime Present,
            EndOfWeek;

  Label2->Caption = "На этой неделе";
  Present= Now(); // Wow — возвращает текущую дату

  // ***************************
  //  для доступа к StartOfWeek, EndOfWeek, YearOf и WeekOf
  //  надо подключить DateUtils.hpp (см. директивы #include)
  // ***************************
  EndOfWeek = StartOfAWeek(YearOf(Present),WeekOf(Present)+1);
```

```
  Query1->SQL->Strings[3] =
        "(Date_F >= '"+ FormatDateTime("dd/mm/yyyy",Present)+"') AND " +
        "(Date_F <  '"+ FormatDateTime("dd/mm/yyyy",EndOfWeek)+"')";
  Query1->Open();
  if ( Query1->RecordCount)
  {
       DataSource1->DataSet = Form1->Query1;
  }
  else
       ShowMessage("На эту неделю никаких дел не запланировано.");
}
```

В результате щелчка на кнопке Все в диалоговом окне программы должно быть выведено все содержимое базы данных. Базу данных представляет компонент Table1. Поэтому функция обработки события click на кнопке Все просто "переключает" источник данных на таблицу (листинг 5.2).

Листинг 5.2. Обработка события на кнопке Все

```
// Щелчок на кнопке Все
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    // установить: источник данных — таблица
    // таким образом, отображается вся БД
    Form1->DataSource1->DataSet = Form1->Table1;
    Label2->Caption = "Все, что намечено сделать";
}
```

Программа "Ежедневник" спроектирована таким образом, что при каждом ее запуске в диалоговом окне выводится текущая дата и список дел, запланированных на этот и ближайшие дни. Вывод даты и названия дня недели в поле Label выполняет функция обработки события OnActivate (ее текст приведен в листинге 5.3). Эта же функция формирует критерий запроса к базе данных, обеспечивающий вывод списка задач, решение которых запланировано на сегодня (в день запуска программы) и на завтра. Если программа запускается в пятницу, субботу или воскресенье, то завтрашним днем считается понедельник. Такой подход позволяет сделать упреждающее напоминание, ведь, возможно, что пользователь не включит компьютер в выходные дни.

Листинг 5.3. Функция обработки события OnActivate

```
AnsiString stDay[7] = {"воскресенье","понедельник","вторник", "среда",
                       "четверг","пятница","суббота"};
```

```
AnsiString stMonth[12] = {"января","февраля","марта",
                          "апреля","мая","июня","июля",
                          "августа","сентября","октября",
                          "ноября","декабря"};

// активизация формы
void__fastcall TForm1::FormActivate(TObject *Sender)
{
    TDateTime Today,     // сегодня
              NextDay;   // следующий день (не обязательно завтра)

    Word Year, Month, Day; // год, месяц, день

    Today = Now ();

    DecodeDate(Today, Year, Month, Day);

    Label1->Caption = "Сегодня " + IntToStr(Day) + " " +
                      stMonth[Month-1] + " " +
                      IntToStr(Year) + " года, " +
                      stDay[DayOfWeek(Today) -1] ;

    Label2->Caption = "Сегодня и ближайшие дни";

    // вычислим следующий день
    // если сегодня пятница, то, чтобы не забыть,
    // что запланировано на понедельник, считаем, что следующий
    // день - понедельник
    switch ( DayOfWeek(Today)) {.
        case 6  : NextDay = Today + 3; break; // сегодня пятница
        case 7  : NextDay = Today + 2; break; // сегодня суббота
        default : NextDay = Today + 1; break;
    }

    // запрос к базе данных; есть ли дела, запланированные
    // на сегодня и на следующий день

    Query1->SQL->Strings[3] =
         "(Date_F >= '"+FormatDateTime("dd/mm/yyyy",Today)+"') AND " +
         "(Date_F <= '"+FormatDateTime("dd/mm/yyyy",NextDay)+"')";

    Query1->Open();
    DataSource1->DataSet = Form1->Query1;
    if ( ! Query1->RecordCount)
     {
        ShowMessage("На сегодня и ближайшие дни никаких дел
                     не запланировано.");
    }
}
```

Использование псевдонима для доступа к базе данных обеспечивает независимость программы от размещения данных в системе, позволяет размещать программу работы с данными и базу данных на разных дисках компьютера, в том числе и на сетевом. Вместе *с* тем для локальных баз данных типичным решением является размещение базы данных в отдельном подкаталоге того каталога, в котором находится программа работы с базой данных. Таким образом, программа работы с базой данных "знает", где находятся данные. При таком подходе можно отказаться от создания псевдонима при помощи Database Desktop и возложить задачу создания псевдонима на программу работы с базой данных. Очевидно, что такой подход облегчает администрирование базы данных.

В качестве иллюстрации сказанного в листинге 5.4 приведен вариант реализации функции `OnActivate`, которая создает псевдоним для базы данных organizer. Предполагается, что база данных находится в подкаталоге DATA того каталога, в котором находится выполняемый файл программы. Непосредственное создание псевдонима выполняет функция `AddStandardAlias`, которой в качестве параметра передается псевдоним и соответствующий ему каталог. Так как во время разработки программы нельзя знать, в каком каталоге будет размещена программа работы с базой данных и, следовательно, подкаталог базы данных, имя каталога определяется во время работы программы путем обращения к функциям `ParamStr(0)` и `ExtractFilePatch`. Значение первой — полное имя выполняемого файла программы, второй - путь к этому файлу. Таким образом, процедуре AddStandardAlias передается полное имя каталога базы данных.

Листинг 5.4. Создание псевдонима во время работы программы

```
void__fastcall TForm1::FormActivate(TObject *Sender)
{
    TDateTime Today,    // сегодня
              NextDay;  // следующий день (не обязательно завтра)

    Word Year, Month, Day; // год, месяц, день

    Today = Now ();

    DecodeDate(Today, Year, Month, Day);

    Label1->Caption = "Сегодня " + IntToStr(Day) + " " +
                      stMonth[Month-1] + " " +
                      IntToStr(Year) + " года, " +
                      stDay[DayOfWeek(Today) -1] ;
```

```
    Label2->Caption = "Сегодня и ближайшие дни";

    // вычислим следующий день
    // если сегодня пятница, то, чтобы не забыть,
    // что запланировано на понедельник, считаем, что следующий
    // день — понедельник
    switch ( DayOfWeek(Today)) {
        case 6  : NextDay = Today + 3; break; // сегодня пятница
        case 7  : NextDay = Today + 2; break; // сегодня суббота
        default : NextDay = Today + 1; break;
    }

#define DIN_ALIAS // псевдоним доступа к БД создается динамически
                  // если псевдоним создан при помощи Database Desktop
                  // или BDE Administrator, директиву #define DIN_ALIAS
                  // надо удалить ("закомментировать")

#ifdef DIN_ALIAS // псевдоним создается динамически

    // создадим псевдоним для доступа к БД
    Session->ConfigMode = cmSession;
    Session->AddStandardAlias("organizer",
                    ExtractFilePath(ParamStr(0))+"DATA\\",
                    "PARADOX"); // база данных "Ежедневник" -
                                // в формате Paradox
#endif

    Form1->Table1->Active = true; // открыть таблицу

    // запрос к базе данных: есть ли дела, запланированные
    // на сегодня и завтра

    Query1->SQL->Strings[3] =
         "(Date_F >= '"+ FormatDateTime("dd/mm/yyyy",Today)+"') AND " +
         "(Date_F <= '"+ FormatDateTime("dd/mm/yyyy",NextDay)+"')";

    Query1->Open();
    DataSource1->DataSet = Form1->Query1;
    if ( ! Query1->RecordCount)
    {
        ShowMessage("На сегодня и ближайшие дни никаких дел
                     не запланировано.");
    }
}
```

Перенос программы управления базой данных на другой компьютер

Часто возникает необходимость перенести базу данных на другой компьютер. В отличие от процесса переноса обычной программы, когда, как правило, достаточно скопировать только выполняемый файл (exe-файл), при переносе программы управления базой данных необходимо выполнить перенос BDE.

Здесь следует вспомнить, что BDE представляет собой совокупность программ, библиотек и драйверов, обеспечивающих работу прикладной программы с базой данных. Выполнить перенос BDE на другой компьютер "вручную" довольно трудно. Поэтому для переноса (распространения) программы, работающей с базами данных, Borland рекомендует создать установочную программу, которая выполнит копирование всех необходимых файлов, в том числе и компонентов BDE. В качестве средства создания установочной программы Borland настоятельно советует использовать утилиту InstallShield Express, которая входит в состав всех наборов C++ Builder. Поставляемая с C++ Builder версия этой утилиты специально адаптирована к задаче переноса и настройки BDE.

ГЛАВА 6

Компонент программиста

C++ Builder позволяет программисту создать свой собственный компонент, поместить его на одну из вкладок палитры компонентов и использовать при разработке приложений точно так же, как и другие компоненты C++ Builder.

Наиболее просто создать компонент программиста можно на базе существующего (базового) компонента путем расширения или ограничения возможностей базового компонента. Например, компонент, обеспечивающий ввод и редактирование числа, логично создать на основе компонента, обеспечивающего ввод строки символов.

Процесс создания компонента может быть представлен как последовательность следующих этапов:

1. Выбор базового класса.
2. Создание модуля компонента.
3. Тестирование компонента.
4. Добавление компонента в пакет компонентов.

Рассмотрим процесс создания компонента программиста на примере. Создадим компонент `NkEdit`, который обеспечивает ввод и редактирование целого или дробного числа. Компонент также должен контролировать вводимое значение на принадлежность заданному диапазону. Тип числа и границы диапазона должны задаваться во время разработки формы приложения, использующего компонент.

Выбор базового класса

Приступая к разработке нового компонента, следует четко сформулировать назначение компонента. Затем необходимо определить, какой из компонентов C++ Builder наиболее близок по своему назначению, виду и функцио-

нальным возможностям к компоненту, который разрабатывается. Именн< этот компонент следует выбрать в качестве базового.

Создание модуля компонента

Перед началом работы по созданию нового компонента нужно создать отдельный каталог для модуля и других файлов компонента. После этого можно приступить к созданию компонента.

Чтобы начать работу над новым компонентом, надо активизировать процесс создания нового приложения (команда **File** | New | **Application),** в меню **Component** выбрать команду New **Component** и в поля диалогового окна New **Component** (рис. 6.1) ввести информацию о создаваемом компоненте.

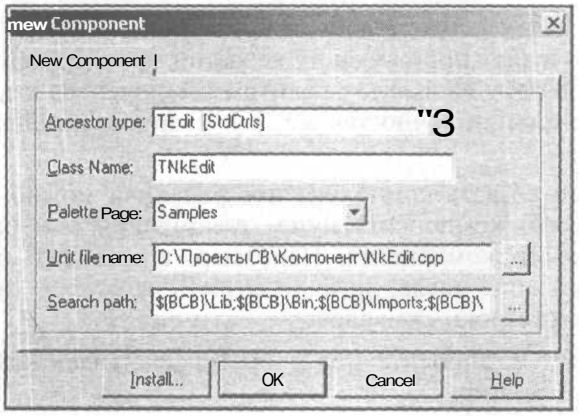


Рис. 6.1. Начало работы над новым компонентом

В поле **Ancestor type** надо ввести базовый тип создаваемого компонента. Для разрабатываемого компонента базовым компонентом является стандартный компонент Edit (поле ввода-редактирования). Поэтому базовым типом разрабатываемого компонента *является* тип `TEdit`.

В поле **Class Name** необходимо ввести имя класса разрабатываемого компонента (например, `TNkEdit`). Вспомните, что в C++ Builder принято соглашение, согласно которому имена типов должны начинаться буквой т.

В поле **Palette Page** нужно ввести имя вкладки палитры компонентов, на которую будет помещен значок компонента. Название вкладки палитры компонентов можно выбрать из раскрывающегося списка. Если в поле **Palette Page** ввести имя еще не существующей вкладки палитры компонентов, то непосредственно перед добавлением компонента вкладка с указанным именем будет создана.

В поле **Unit file name** находится автоматически сформированное имя файла модуля создаваемого компонента. C++ Builder присваивает модулю компо-

нента имя, которое совпадает с именем типа компонента, но без буквы т. Щелкнув на кнопке с тремя точками, можно выбрать каталог, в котором должен быть сохранен модуль компонента.

В результате щелчка на кнопке OK будет сформирован модуль компонента, состоящий из двух файлов: файла заголовка (листинг 6.1) и файла реализации (листинг 6.2).

Листинг 6.1. Файл NkEdit.h

```
ttifndefNkEditH
tfdefineNkEditH

#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>

class PACKAGE TNkEdit : public TEdit
{
private:
protected:
public:
    __fastcall TNkEdit(TComponent* Owner);
__published:
};
tendif
```

Листинг 6.2. Файл NkEdit.cpp

```
#include <vcl.h>

ttpragmahdrstop

#include "NkEdit.h"
#pragma package(smart_init)

static inline void ValidCtrCheck(TNkEdit *)
{
    new TNkEdit(NULL);

>
__fastcall TNkEdit::TNkEdit(TComponent* Owner)
    : TEdit(Owner)
{
}
```

```
namespace NkEdit
{
    void _ fastcall PACKAGE Register ()
    {
        TComponentClass classes [1]= { _classid(TNkEdit) } ;
        RegisterComponents ("Samples", classes, 0) ;
    }
}
```

В файле заголовка NkEdit.h находится объявление нового класса. В файл реализации NkEdit.cpp помещена функция Register, которая обеспечивает регистрацию, установку значка компонента на указанную вкладку палитры компонентов.

В сформированный C++ Builder шаблон компонента нужно внести дополнения: объявить поля данных, функции доступа к полям данных, свойства и методы. Если на некоторые события компонент должен реагировать не так, как базовый, то в объявление класса нужно поместить объявления соответствующих функций обработки событий.

В листингах 6.3 и 6.4 приведены файлы заголовка и реализации компонента NkEdit после внесения всех необходимых изменений.

Листинг 6,3. nkedit.h

```
#ifndef NkEditH
#define NkEditH

#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>

class PACKAGE TNkEdit : public TEdit
{
private:
   bool FEnableFloat; // разрешен ввод дробного числа
   // диапазон
   float FMin; // нижняя граница
   float FMax; // верхняя граница

    /* функция SetNumb используется для изменения содержимого
       поля редактирования */
    void _ fastcall SetNumb (float n) ;

    /* Функция GetNumb используется для доступа к полю редактирования */
    float __fastcall GetNumb (void) ;
```

```
    /* эти функции обеспечивают изменение границ диапазона
       допустимых значений */
    bool __fastcall SetMin(float min);
    bool __fastcall SetMax(float max);

protected:

public:
    __fastcall TNkEdit(TComponent* Owner);    // конструктор

    /* Свойство Numb должно быть доступно только во время
       работы программы. Поэтому оно объявлено в секции public.
       Если надо, чтобы свойство было доступно во время разработки формы
       и его значение можно было задать в окне Object Inspector, то
       его объявление нужно поместить в секцию published
    V
    __property float Numb = {read = GetNumb }; //, write = SetNumb};

    // Функция обработки события KeyPress
    DYNAMIC void __fastcall KeyPress(char &key);

__published:
      // объявленные здесь свойства доступны в Object Inspector

    __property bool EnableFloat = { read    = FEnableFloat,
                                    write   = FEnableFloat };

    __property float Min = {read  = FMin,
                            write = SetMin };

   __property float Max = {read  = FMax,
                           write = SetMax } ;
};

#endif
```

Листинг 6.4. nkedit.cpp

```
ttinclude<vcl.h>

#pragma hdrstop

#include "NkEdit.h"
#pragma package(smart_init)
```

```
static inline void ValidCtrCheck(TNkEdit *)
{
    new TNkEdit(NULL);
}

// конструктор
_fastcall TNkEdit::TNkEdit(TComponent*Owner)
    : TEdit(Owner)
{
    // конструктор имеет прямой доступ к полям компонента
    Text = "0";
    FMin = 0;
    FMax = 100;
    FEnableFloat = true;
}

namespace Nkedit
{
    void _fastcall PACKAGE Register()
    {
        TComponentClass classes[1] = {__classid(TNkEdit)};
        RegisterComponents("Samples", classes, 0);
    }
}

void _fastcall TNkEdit::SetNumb(float n)
{
    Text = FloatToStr(n);
}

// возвращает значение, соответствующее строке,
// которая находится в поле редактирования
float _fastcall TNkEdit::GetNumb(void)
{
    if(Text.Length())
        return StrToFloat(Text);
    else return 0;
}

// функция обработки события KeyPress в поле компонента NkEdit
void _fastcall TNkEdit::KeyPress(char&Key)
{
    // Коды запрещенных клавиш заменим нулем, в результате чего
    // эти символы в поле редактирования не появятся
    switch (Key) {
        case '0' :
```

```
        case '1' :
        case '2' :
        case '3' :
        case '4' :
        case '5' :
        case '6' :
        case '7' :
        case '8' :
        case '9' : break;

        case '.' :
        case ',' :
              Key = DecimalSeparator;
              if (Text.Pos(DecimalSeparator) || (! FEnableFloat))
                   Key = 0;
              break;

        case '-' : // знак "минус"
                   if (Text.Length()||(FMin >= 0))
                      // минус уже введен или fMin >= 0
                     Key = 0;
                   break;

        case VK_BACK:    // клавиша <Backspace>
                   break;

        default :  // остальные символы запрещены
                   Key = 0;
    }

    if ((Key>='0') && (Key <= '9')) {
        /* Проверим, не приведет ли ввод очередной цифры
           к выходу числа за границы диапазона. Если да,
           то заменим введенное число на максимальное или минимальное*/

       AnsiString st = Text + Key;

       if (StrToFloat(st) < FMin) {
            Key = 0;
            Text = FloatToStr(FMin);
       }
       if (StrToFloat(st) > FMax) {
            Key = 0;

            Text = FloatToStr(FMax);
       }
    }
```

```
    // вызвать функцию обработки события KeyPress базового класса
    TEdit::KeyPress(Key);
}

// устанавливает значение поля FMin
bool _fastcall TNkEdit::SetMin(float min)
{
    if (min > FMax) return false;

    FMin = min;
    return true;
}

// устанавливает значение поля FMin
bool _fastcall TNkEdit::SetMax(float max)
{
    if ( max < FMin) return false;

    FMax = max;
    return true;
}
```

В объявлении класса TNkEdit добавлены объявления полей FEnabledFloat, FMin и FMax. Имена полей, согласно принятому в C++ Builder соглашению, начинаются с буквы F (от Field, поле). Поле FEnabiedFioat хранит признак возможности ввода в поле редактирования дробного числа. Поля FMin и FMax хранят границы диапазона. Доступ к полям обеспечивают соответствующие свойства: EnabledFloat, Min и мах. Так как объявления этих свойств находятся в секции published, то они будут доступны в окне **Object Inspector.** Свойство Numb, представляющее собой число, которое находится в поле редактирования, объявлено в секции public, поэтому оно доступно только во время работы программы. Здесь следует обратить внимание на то, что у свойства Numb нет соответствующего поля. Значение этого свойства вычисляется во время работы программы путем преобразования в число значения свойства Text базового компонента. Свойства Min и мах получают доступ к полям данных для чтения напрямую, для записи — посредством функций SetMin и SetMax. Свойство EnabiedFioat получает доступ к полю FEnabiedFioat для чтения и записи напрямую. Так как компонент NkEdit должен обеспечить фильтрацию символов (в поле редактирования должны отображаться только цифры и, в случае, если значение свойства EnabiedFioat равно true, десятичный разделитель), то в объявление класса добавлено объявление функции KeyPress, которая предназначена для обработки соответствующего события.

Реакцию компонента NkEdit на нажатие клавиши клавиатуры определяет функция TNkEdit::KeyPress. В качестве параметра эта функция получает код

нажатой клавиши. Перед вызовом функции `TEdit:`:KeyPress, которая обеспечивает обработку события Keypress базовым компонентом, код нажатой клавиши проверяется на допустимость. Если нажата недопустимая клавиша, то код символа заменяется на ноль. Допустимыми для компонента `NkEdit`, в зависимости от его настройки, являются цифровые клавиши, разделитель целой и дробной частей числа (в зависимости от настройки Windows, точка или запятая), "минус" и клавиша <Backspase>.

Здесь следует вспомнить, что в тексте программы дробная часть числовой константы отделяется от целой части точкой. Во время работы программы при вводе исходных данных пользователь должен использовать тот символ, который задан в настройке Windows. В качестве разделителя обычно используют запятую (стандартная для России настройка) или точку. Приведенная процедура обработки события onKeypress учитывает, что настройка Windows может меняться, и поэтому введенный пользователем символ сравнивается не с константой, а со значением глобальной переменной `DecimalSeparator`, которая содержит символ-разделитель, используемый в Windows в данный момент.

Тестирование компонента

Перед тем как добавить новый компонент в палитру компонентов, необходимо всесторонне его проверить. Для этого надо создать приложение, использующее компонент, и убедиться, что компонент работает так, как надо.

Во время создания формы приложения нельзя добавить в форму компонент, значка которого нет в палитре компонентов. Однако такой компонент может быть добавлен в форму динамически, т. е. во время работы приложения.

Наиболее просто выполнить тестирование компонента можно следующим образом. Сначала надо активизировать процесс создания нового приложения, а затем создать форму (добавить и настроить необходимые компоненты) и сохранить приложение в том каталоге, в котором находятся файлы тестируемого компонента.

Вид формы приложения тестирования компонента `NkEdit` приведен на рис. 6.2.

Форма содержит две метки. Первая метка обеспечивает вывод общей информации о компоненте; вторая метка (на рисунке она выделена) используется для вывода информации о настройке компонента. Самого компонента NkEdit в форме нет. Он будет создан во время работы программы, в момент ее запуска.

После того как форма тестового приложения будет создана, в модуль приложения надо внести дополнения, приведенные ниже.

1. В текст файла реализации (cpp-файл) включить следующую директиву `#include "nkedit.cpp"`.
2. Объявить компонент `NkEdit` (оператор TNkEdit *NkEdit). Здесь следует вспомнить, что компонент является объектом (точнее ссылкой на объект), поэтому объявление компонента не обеспечивает создание компонента, а только создает указатель на компонент.
3. *Для* формы приложения создать процедуру обработки события `OnCreate`, которая путем вызова конструктора тестируемого компонента создаст компонент и выполнит его настройку (присвоит значения свойствам компонента).

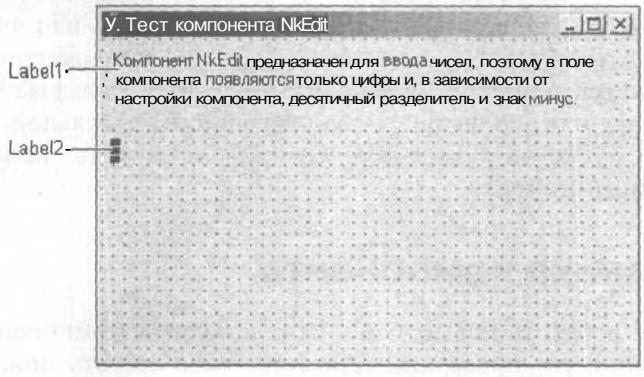


Рис. 6.2. Форма приложения "Тест компонента NkEdit"

В листинге 6.5 приведен файл реализации приложения тестирования компонента NkEdit.

Листинг 6.5. Тест компонента `NkEdit`

```
#include <vcl.h>
#pragma hdrstop

#include "tk_.h"

#include "nkedit.cpp"

#pragma package (smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;      // форма
TNkEdit *NkEdit;    // компонент программиста

// конструктор формы
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
```

```
{
    // создадим и инициализируем компонент NkEdit
    NkEdit = new TNkEdit(Form1);
    NkEdit->Parent = Form1;
    NkEdit->Text = "0";
    NkEdit->Left = 10;
    NkEdit->Top = 100;

    // настроим компонент
    // зададим границы диапазона
    NkEdit->Min = -100;
    NkEdit->Max = 100;
    NkEdit->EnableFloat = true;  // разрешен ввод дробных чисел
}

// обработка события FormCreate
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    AnsiString st = "Введите ";

    // информация о компоненте
    if (NkEdit->EnableFloat)
        st = st + "дробное";
    else st = st ,+ "целое";
    st = st + " число от ";
    st = st + FloatToStr(NkEdit->Min);
    st = st + " до ";
    st = st + FloatToStr(NkEdit->Max);

    Label2->Caption = st;
}
```

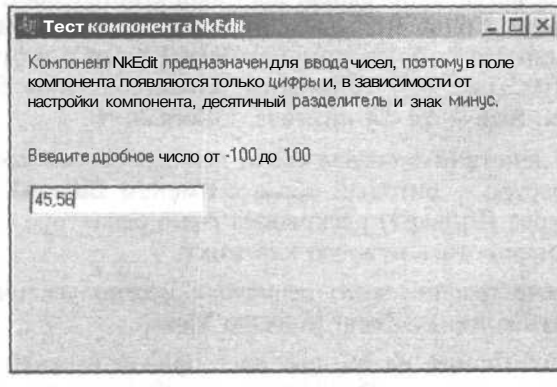


Рис. 6.3. Тестирование компонента: поле ввода — компонент NkEdit

Тестируемый компонент создает и настраивает конструктор формы. Следует обратить внимание, что свойству Parent созданного компонента обязательно надо присвоить значение. Если этого не сделать, то компонент на форме не появится. Информацию о настройках созданного компонента выводит функция `FormCreate`. На рис. 6.3 приведен вид окна программы "Тест компонента NkEdit" во время ее работы.

Установка компонента

Для того чтобы значок компонента появился в палитре компонентов, компонент должен быть добавлен в один из пакетов (Packages) компонентов C++ Builder. *Пакет компонентов* — это специальная библиотека (файл с расширением bpk). Например, для компонентов, созданных программистом, предназначен пакет dclusr.bpk.

В процессе добавления компонента в пакет C++ Builder использует *файл ресурсов компонента,* в котором должен находиться битовый образ значка компонента. Имя файла ресурсов компонента должно совпадать с именем файла модуля компонента. Файл ресурсов имеет расширение dcr (Dynamic Component Resource). Битовый образ, находящийся внутри файла ресурсов, должен иметь имя, совпадающее с именем класса компонента.

Ресурсы компонента

Файл ресурсов компонента можно создать при помощи утилиты Image Editor, которую можно запустить из C++ Builder (команда **Tools | Image Editor)** или из Windows (команда **Пуск | Программы | C++ Builder | Image Editor).**

Для того чтобы создать файл ресурсов компонента, нужно из меню **File** выбрать команду New и из появившегося списка выбрать тип создаваемого файла — **Component Resource File** (рис. 6.4).

В результате открывается окно файла ресурсов Untitled1.dcr, а в меню диалогового окна **Image Editor** появляется новый пункт — **Resource.** Теперь нужно из меню Resourse выбрать команду **New/Bitmap** и в открывшемся окне **Bitmap Properties** (рис. 6.5) установить характеристики битового образа значка компонента: **Size** — 24x24 пиксела, **Colors** — 16.

В результате этих действий в создаваемый файл ресурсов компонента будет добавлен новый ресурс — битовый образ с именем **Bitmap**1. Двойной щелчок на имени ресурса (**Bitmap1**) раскрывает окно редактора битового образа, в котором можно нарисовать нужную картинку.

Изображение в окне графического редактора можно увеличить. Для этого необходимо выбрать команду **Zoom In** меню **View.**

Следует обратить внимание на то, что цвет правой нижней точки рисунка определяет "прозрачный" цвет. Элементы значка компонента, закрашенные этим цветом, на палитре компонентов C++ Builder не видны.

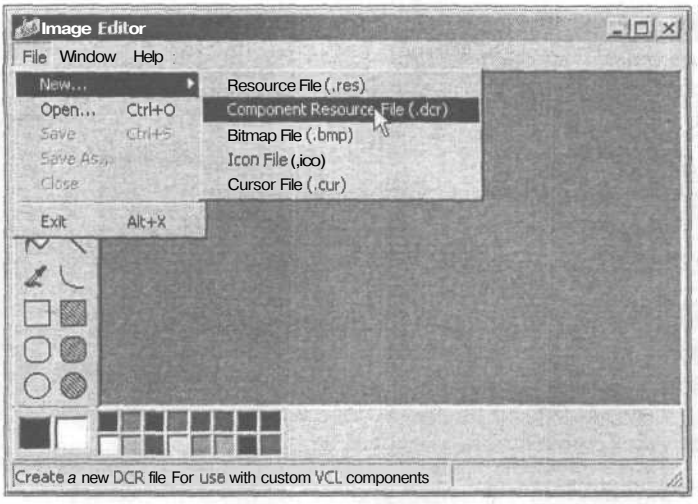


Рис. 6.4. Начало работы по созданию файла ресурсов компонента

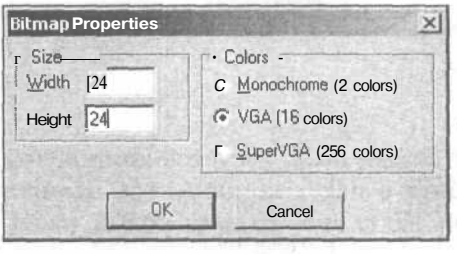


Рис. 6.5. Диалоговое окно **Bitmap Properties**

Перед тем как сохранить файл ресурсов компонента, битовому образу надо присвоить имя. Имя должно совпадать с именем класса компонента. Чтобы задать имя битового образа, необходимо щелкнуть правой кнопкой мыши на имени битового образа (**Bitmap1**), выбрать в появившемся контекстном меню команду **Rename** и ввести новое имя.

Созданный файл ресурсов компонента нужно сохранить в том каталоге, в котором находится файл модуля компонента. Для этого надо из меню File выбрать команду Save.

На рис. 6.6 приведен вид окна **Image Editor,** в левой части которого содержится файл ресурсов компонента `TNkEdit` (nkedit.dcr), а в правой части – окно редактора битового образа, в котором находится изображение значка компонента. Обратите внимание, что имя файла ресурсов компонента (NkEdit.dcr) должно совпадать с именем модуля компонента (NkEdit.cpp), а имя битового образа (TNKEDIT) — с именем класса компонента (TNkEdit).

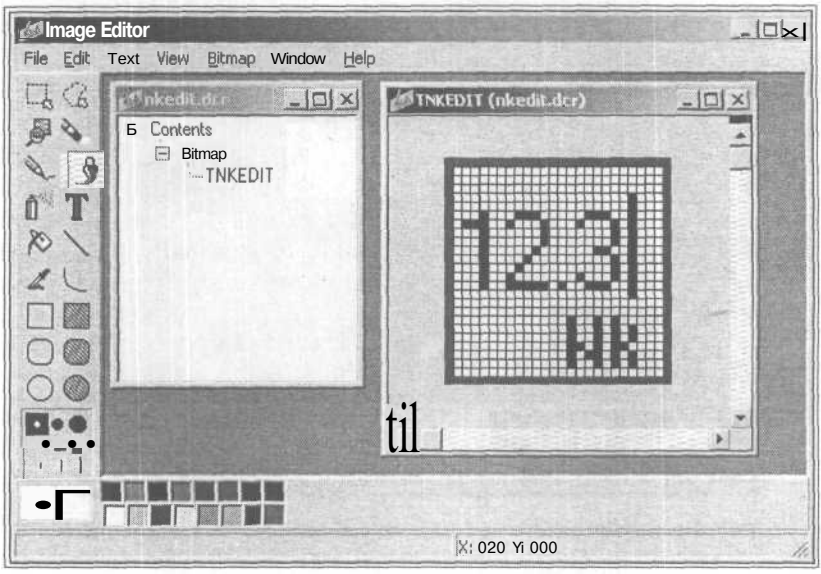


Рис. 6.6. Значок компонента NkEdit

Установка

После того как будет создан файл ресурсов компонента, можно приступить к установке компонента в пакет компонентов. Компонент можно установить в существующий пакет или создать новый пакет и затем установить в него компонент.

Чтобы установить компонент в существующий пакет, надо из меню **Component** выбрать команду **Install Component** и заполнить поля вкладки **Into existing package** диалогового окна **Install Component** (рис. *6.1)*.

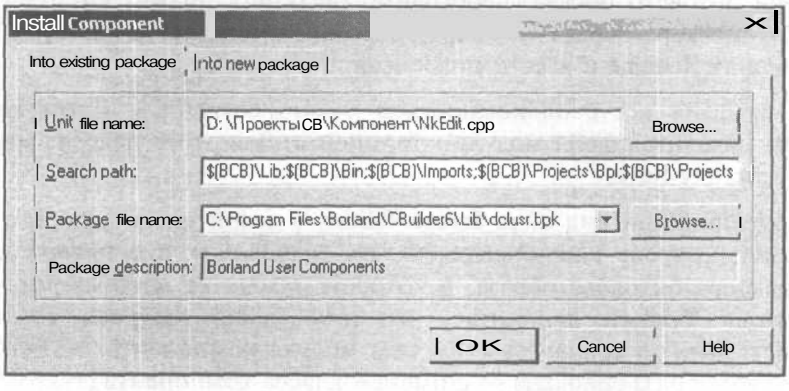


Рис. 6.7. Диалоговое окно **Install Component**

В поле **Unit file name** (Имя файла модуля) нужно ввести имя файла модуля. Для этого удобно воспользоваться кнопкой **Browse.**

Поле **Search path** (Путь поиска) должно содержать разделенные точкой с запятой имена каталогов, в которых C++ Builder во время установки компонента будет искать необходимые файлы, в частности файл ресурсов компонента. Если имя файла модуля было введено в поле **Unit file name** выбором файла из списка, полученного при помощи кнопки **Browse,** то C++ Builder автоматически добавляет в поле **Search path** имена необходимых каталогов. Следует обратить внимание на то, что файл ресурсов компонента должен находиться в одном из каталогов, перечисленных в поле **Search path.** Если его там нет, то компоненту будет назначен значок его родительского класса.

Поле **Package file name** должно содержать имя пакета, в который будет установлен компонент. По умолчанию компоненты, создаваемые программистом, добавляются в пакет dclusr.bpk.

Поле **Package description** содержит название пакета. Для пакета dclusr.bpk это текст: **Borland User Components**.

После того как поля будут заполнены, надо щелчком на кнопке **OK** активизировать процесс установки. Сначала на экране появляется окно **Confirm** (рис. 6.8), в котором C++ Builder просит подтвердить обновление пакета.

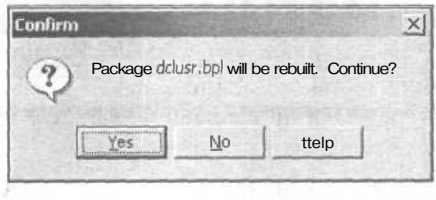


Рис. 6.8. Запрос подтверждения обновления пакета в процессе установки компонента

После щелчка на кнопке **Yes** процесс установки продолжается. Если он завершается успешно, то на экране появляется информационное сообщение (рис. 6.9) о том, что пакет обновлен, а компонент зарегистрирован.

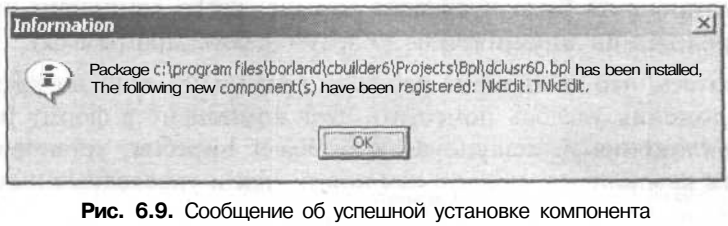


Рис. 6.9. Сообщение об успешной установке компонента

После установки компонента в пакет открывается диалоговое окно **Package** (Редактор пакета компонентов) (рис. 6.10), в котором перечислены компоненты, находящиеся в пакете.

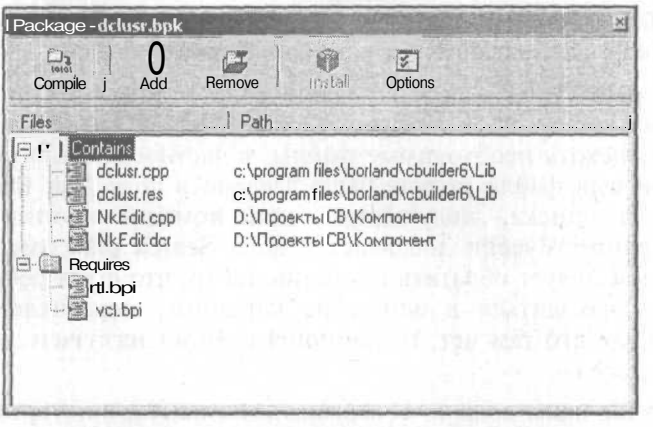


Рис. 6.10. Окно редактора пакета компонентов

На этом процесс установки компонента заканчивается. В результате на вкладке палитры компонентов, имя которой было задано при создании модуля компонента, появляется значок установленного компонента (рис. 6.11).

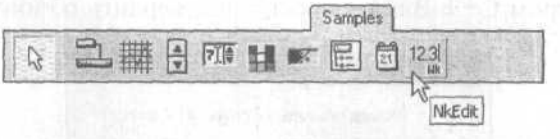


Рис. 6.11. Значок компонента `NkEdit` на вкладке **Samples**

Проверка компонента

После того как компонент будет добавлен в пакет и его значок появится в палитре компонентов, необходимо проверить поведение компонента во время разработки приложения, использующего этот компонент (работоспособность компонента была проверена раньше, когда компонент добавлялся в форму приложения динамически, во время работы программы).

Можно считать, что компонент работает правильно, если во время разработки приложения удалось поместить этот компонент в форму разрабатываемого приложения и, используя окно **Object Inspector,** установить значения свойств компонента, причем как новых, так и унаследованных от родительского класса.

Работоспособность компонента NkEdit можно проверить, использовав его, например, в приложении "Сила тока", вид формы которого приведен на рис.6.12.

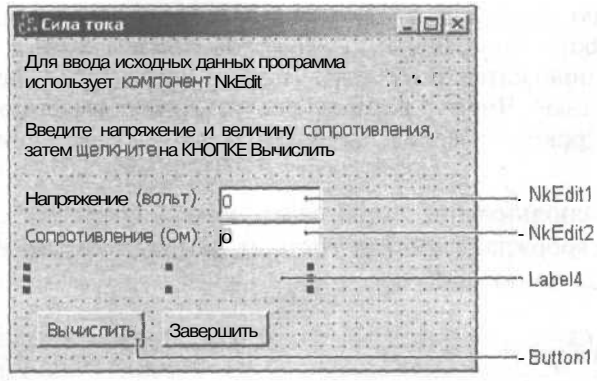


Рис. 6.12. Форма приложения "Сила тока" (поля ввода-редактирования компоненты NkEdit)

Внешне форма разрабатываемого приложения почти ничем не отличается от формы приложения "Сила тока", рассмотренного в *гл. 2.* Однако если выбрать поле ввода, то в окне **Object Inspector** будет указано, что текущим компонентом является компонент класса `TNkEdit`, а в списке свойств можно будет увидеть свойства, которых нет у стандартного компонента Edit (рис. 6.13).

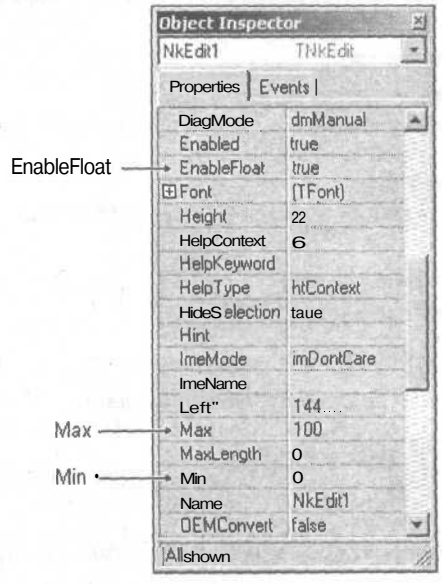


Рис. 6.13. Значения свойств EnableFloat, Max и Min компонента NkEdit можно задать в окне Object Inspector

В листинге 6.6 приведен модуль приложения "Сила тока". Здесь надо обратить внимание на следующее. Первое. В программе нет кода, обеспечиваю-

щего фильтрацию символов, вводимых в поле редактирования. Тем не менее во время работы программы пользователь может ввести в поле редактирования только положительное число. Второе. В программе не используется функция StrToFloat. Число, которое соответствует введенной в поле редактирования строки символов, получается путем обращения к свойству Numb.

Очевидно, что использование в программе компонента NkEdit вместо стандартного Edit освобождает программиста от рутины, сокращает размер кода и делает его более понятным.

Листинг 6.6. "Сила тока"

```
// нажатие клавиши в поле Напряжение
void _ fastcall TForm1 : :NkEdit1KeyPress (TObject *Sender, char &Key)
{
        if ( Key == VK_RETURN)
                NkEdit2->SetFocus();
}

// нажатие клавиши в поле Сопротивление
void _ fastcall TForm1 : :NkEdit2KeyPress (TObject *Sender, char &Key)
{
        if ( Key == VK_RETURN)
                Button1->SetFocus();
}

// нажатие кнопки Вычислить
void _ fastcall TForm1: :Button1Click (TObject *Sender)
{
    float u; // напряжение
    float r; // сопротивление
    float i; // ток

    // получить исходные манные из полей ввода
    u = NkEdit1->Numb;
    r = NkEdit2->Numb;

    if ( r == 0) {
        ShowMessage ("Сопротивление не должно быть равно нулю");
        return;
    }

    // вычислить ток
    i = u/r;
```

```
    // вывести результат
    Label4->Caption = "Ток : " +
        FloatToStrF(i,ffGeneral,7,3) + "A";
}
```

Настройка палитры компонентов

C++ Builder позволяет менять порядок следования вкладок в палитре компонентов, названия вкладок, а также порядок следования значков компонентов на вкладках. Настройка палитры компонентов выполняется в диалоговом окне **Palette Properties,** которое открывается выбором из меню **Component** команды **Configure Palette** (рис. 6.14).

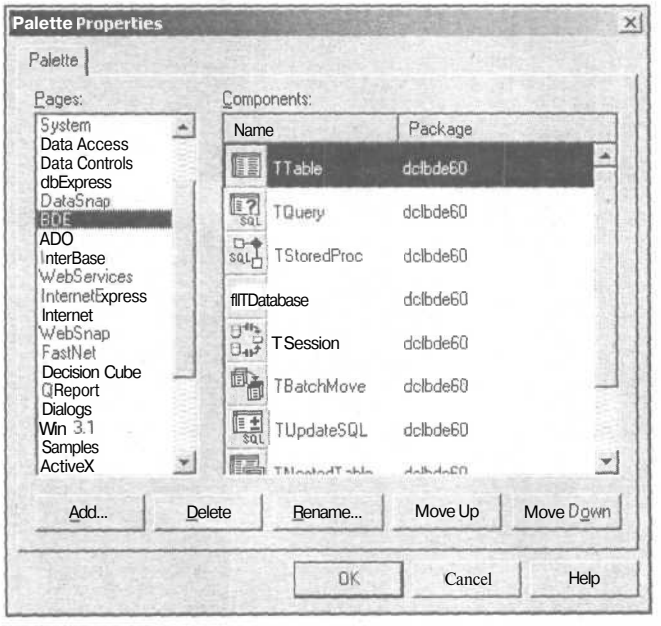


Рис. 6.14. Диалоговое окно **Palette Properties**

Сначала в списке **Pages** необходимо выделить нужную вкладку палитры компонентов. Затем, если надо изменить порядок следования вкладок палитры компонентов, следует воспользоваться кнопками **Move Up и Move Down** и путем перемещения выбранного имени по списку Pages добиться нужного порядка следования вкладок.

Если надо изменить порядок следования значков компонентов на вкладке, то в списке **Components** следует выбрать нужный значок компонента и кнопками **Move Up и Move Down** переместить значок на новое место.

При необходимости изменить имя вкладки палитры следует в списке **Pages** выбрать имя нужной вкладки, нажать кнопку **Rename** и в поле **Page name** открывшегося диалогового окна **Rename page** (рис. 6.15) ввести новое имя.

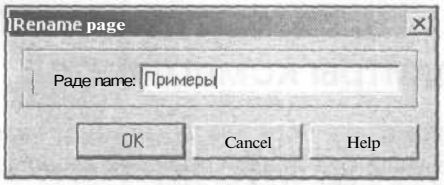


Рис. 6.15. Диалоговое окно **Rename page**

ГЛАВА 7

Консольное приложение

Хотя эта книга посвящена программированию в Windows, нельзя обойти вниманием так называемые *консольные приложения*. Консольное приложение — это приложение, которое для взаимодействия с пользователем не использует графический интерфейс. Устройством, обеспечивающим взаимодействие с пользователем, является *консоль* — клавиатура и монитор, работающий в режиме отображения символьной информации (буквы, цифры и специальные знаки).

В операционной системе консольное приложение работает в окне командной строки.

Консольные приложения удобны как иллюстрации при рассмотрении общих вопросов программирования, когда надо сосредоточиться на сути проблемы, а также как небольшие утилиты "для внутреннего потребления".

Ввод/вывод

Перед тем как приступить к созданию консольного приложения, рассмотрим функции, обеспечивающие вывод на экран и ввод с клавиатуры.

Наиболее универсальными функциями, обеспечивающими вывод и ввод информации в консольных приложениях, являются функции printf и scanf. Для того чтобы программа могла их использовать, в начало программы надо включить директиву `#include <stdio.h>`.

Функция *printf*

В общем виде инструкция вызова функции printf выглядит так:

```
printf(УправляющаяСтрока, СписокПеременных)
```

Параметр *УправляющаяСтрока* задает способ отображения (формат) значений переменных, имена которых задает параметр *СписокПеременных*. Помимо спецификаторов формата, параметр *УправляющаяСтрока* может содержать символы и управляющие последовательности.

Параметр *СписокПеременных* не является обязательным и представляет собой последовательность разделенных запятыми имен переменных, значения которых должны быть выведены.

Спецификатор формата задает вид вывода. Например, значение переменной типа float можно вывести как десятичное число с точкой (%f) или как число в формате с плавающей точкой (%e). В спецификаторе формата можно задать размер поля вывода (количество позиций экрана), а для формата f – размер поля для вывода целой и дробной частей числа. Если во время работы программы окажется, что выводимое значение не умещается в поле, указанном в спецификации, то для его вывода будет использовано столько позиций, сколько необходимо.

В табл. 7.1 приведены наиболее часто используемые спецификаторы формата. Необязательный параметр *n*, вместо которого надо подставить десятичное число, задает размер поля вывода; параметр *m* — размер поля для вывода цифр дробной части.

Таблица 7.1. Спецификаторы формата

Спецификатор	Тип переменной	Форма вывода
%nd	int	Десятичное со знаком
%n.mf	float или double	Дробное с десятичной точкой
%ne	float или double	Дробное в виде числа с плавающей точкой
%nc	char	Символ
%ns		Строка

При выводе одной инструкцией значений нескольких переменных значение первой переменной выводится в соответствии с первым по порядку спецификатором формата из управляющей строки, второй со вторым и т. д.

Следует обратить внимание на то, что компилятор не проверяет, соответствует ли количество переменных, значения которых должны быть выведены, количеству спецификаторов в управляющей строке, а также соответствие типа переменной — спецификатору. Например, если переменная x объявлена как float, то в инструкции printf("x=%i", x) компилятор не обнаружит ошибку.

Если надо вывести символ, который не может быть помещен в строку вывода обычным образом путем набора на клавиатуре, — например, символ новой строки или двойная кавычка, которая в языке C/C++ используется для ограничения в тексте программы строк, — то вместо этого символа применяется специальная последовательность символов. Специальная (управляющая) последовательность начинается символом обратной наклонной черты. Во время работы программы символы специальной последовательности на экран не выводятся, а выполняется действие, обозначаемое этой последовательностью. В табл. 7.2 приведены наиболее часто используемые управляющие последовательности.

Таблица 7.2. *Управляющие последовательности*

Последовательность	Действие
\n	Переводит курсор в начало следующей строки
\r	Переводит курсор на следующую строку текущей колонки
\t	Переводит курсор в следующую позицию табуляции
\"	Выводит двойную кавычку
\\	Выводит обратную наклонную черту
\0x*ШестнадцатеричноеЧисло*	Выводит символ, код которого указан

Для вывода на экран сообщений часто используют функцию puts, которая, в отличие от printf, после вывода автоматически переводит курсор в начало следующей строки. У функции puts один параметр — сообщение. В простейшем случае в качестве параметра функции puts используется строковая константа. Например, функция

```
puts("У лукоморья дуб зеленый,\nЗлатая цепь на дубе том.")
```

выводит две строчки стихотворения, каждую на отдельной строке, и переводит курсор в начало следующей строки.

Чтобы вывести цветной текст, надо использовать функции cprintf и cputs. Они ничем не отличаются от рассмотренных ранее printf и puts, за исключением того, что цвет символов, выводимых этими функциями, можно задать, вызвав функцию textcolor, а цвет фона — textbackground.

В общем виде инструкции вызова указанных выше функций выглядят так:

```
textcolor(Цвет);
textbackground(Цвет);
```

Параметр цвет — параметр целого типа, в качестве которого обычно используют одну из именованных констант (табл. 7.3).

Таблица 7.3. Константы, в качестве параметра Цвет

Цвет	Константа	Значение константы
Черный	BLACK	0
Синий	BLUE	1
Зеленый	GREEN	2
Бирюзовый	CYAN	3
Красный	RED	4
Сиреневый	MAGENTA	5
Коричневый	BROWN	6
Светло-серый	LIGHTGRAY	7
Серый	DARKGRAY	8
Голубой	LIGHTBLUE	9
Светло-зеленый	LIGHTGREEN	10
Светло-бирюзовый	LIGHTCYAN	11
Алый	LIGHTRED	12
Светло-сиреневый	LIGHTMAGENTA	13
Желтый	YELLOW	14
Белый (яркий)	WHITE	15

Следует обратить внимание на то, что в качестве параметра функции `textcolor` можно использовать символьные константы со значением от О до 15, а в качестве параметра функции `textbackground` — только от 0 до 7.

При выводе на экран весьма полезна функция `clrscr`, которая очищает экран, закрашивая его цветом фона, установленным функцией textbackground.

Функции textcoior, textbackground, cirscr и приведенные выше константы объявлены в файле conio.h, поэтому, чтобы они были доступны, в текст **Программы НУЖНО ВКЛЮЧИТЬ директиву** ttinclude <conio.h>.

Функция *scanf*

Наиболее универсальной функцией, которая позволяет ввести данные с клавиатуры, является функция scanf. В общем виде инструкция вызова функции scanf для ввода значения одной переменной выглядит так:

```
scanf(Формат, &Переменная);
```

где:

- Формат — это строка, которая содержит спецификатор формата, определяющий то, как должна интерпретироваться строка, введенная с клавиатуры. Наиболее часто используемыми спецификаторами являются: %i — для ввода целых, %f — для ввода дробных, %s — для ввода строк;
- *&Переменная* — это адрес переменной, значение которой вводится.

Например, инструкция

```
scanf("%i", &kol);
```

вводит целое число, а инструкция

```
scanf("%i%f", &kol, &cena);
```

вводит целое и дробное.

При вызове функции scanf происходит следующее. Программа приостанавливает работу и ждет, пока пользователь наберет на клавиатуре строку символов и нажмет клавишу <Enter>. До нажатия <Enter> можно редактировать вводимую строку (например, нажав клавишу <Back Space> можно удалить последний введенный символ). После нажатия клавиши <Enter> функция scanf преобразует введенную строку в данные и записывает их в переменную, адрес которой указан. Преобразование выполняется в соответствии со спецификатором формата. Например, в результате выполнения инструкции `scanf("%f", &cena)` и набора на клавиатуре строки `25.99` значение переменной сепа будет равно 25.99.

Следует обратить внимание, что при использовании функции scanf наиболее частой ошибкой, причем не обнаруживаемой компилятором, является отсутствие символа & перед именем переменной.

Если введенная пользователем строка не соответствует типу ожидаемых данных — например, программа ждет ввода целого числа, а пользователь ввел дробное, — то функция scanf обрабатывает только ту часть введенной строки, которая может быть преобразована в требуемые данные. Например, в программе для ввода данных о стоимости покупки используется инструкция `scanf("%i%f", &kol, &cena)`, которая предполагает, что пользователь введет в одной строке сначала количество предметов, а затем цену предмета. Если во время работы программы вместо строки 3 24.99 (три предмета по 24.99) ввести строку 24.99 3, то значение переменной `kol` будет равно 24, а переменной сепа — 99.

Создание консольного приложения

Консольное приложение создается следующим образом. Сначала нужно из меню **File** выбрать команду New | **Other Application** и на вкладке New появившегося диалогового окна New **Items** щелкнуть на значке **Console Wizard**

(рис. 7.1). В результате этих действий на экране появится окно **Console Wizard** (рис. 7.2). В этом окне можно выбрать язык программирования и указать, будет ли использоваться та или иная библиотека. После того как будут заданы параметры создаваемого консольного приложения, надо щелкнуть на кнопке OK. В результате C++ Builder создаст проект консольного приложения и на экране появится окно редактора кода, в котором находится шаблон консольного приложения — функция main (рис. 7.3).

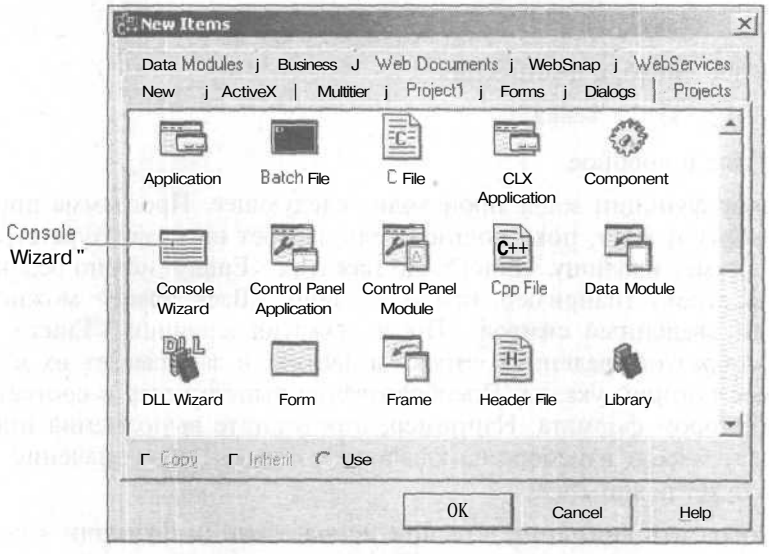


Рис. 7.1. Чтобы приступить к созданию консольного приложения, надо щелкнуть на значке **Console Wizard**

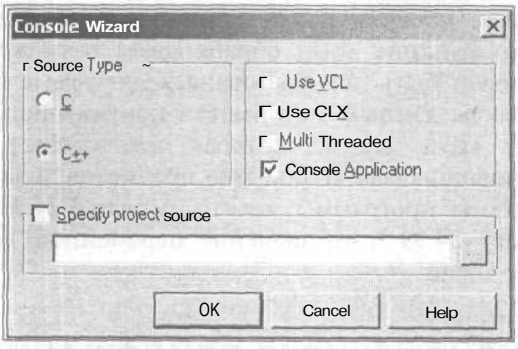


Рис. 7.2. В окне **Console Wizard** надо задать характеристики консольного приложение

Начинается консольное приложение директивой `#pragma hdrstop`, которая запрещает выполнение предварительной компиляции подключаемых фай-

лов. После этой директивы надо вставить директивы #include, обеспечивающие подключение необходимых библиотек (например, #include <stdio.h>). Директива #pragma argsused отключает предупреждение компилятора о том, что аргументы, указанные в заголовке функции, не используются.

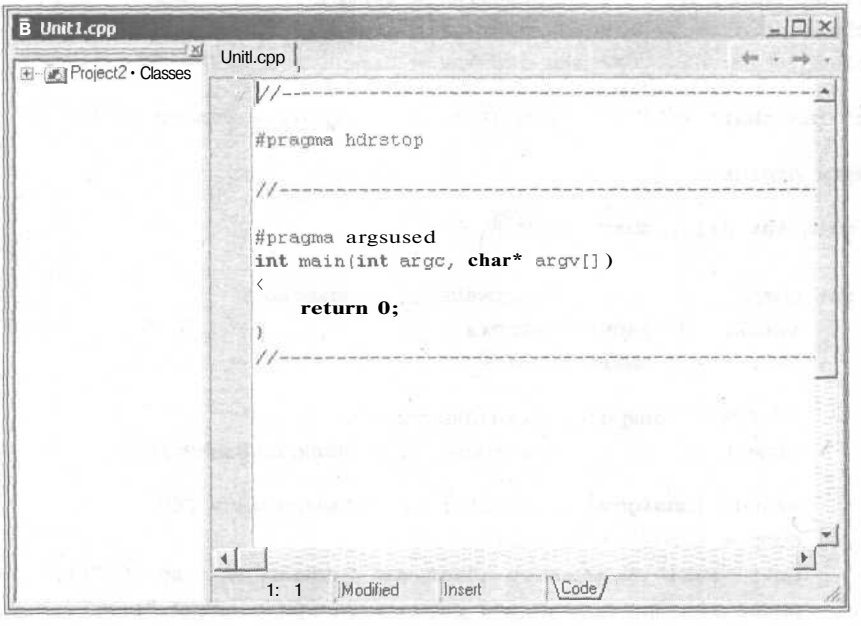


Рис. 7.3. Шаблон консольного приложения

Следует обратить внимание на то, что консольное приложение разрабатывается в Windows, а выполняется как программа DOS. В DOS и Windows буквы русского алфавита имеют разные коды (в DOS используется кодировка ASCII, а в Windows — ANSI). Это приводит к тому, что консольное приложение вместо сообщений на русском языке выводит "абракадабру".

Проблему вывода сообщений на русском языке консольными приложениями можно решить, разработав функцию перекодировки ANSI-строки в строку ASCII. Если эту функцию назвать rus, то инструкция вывода сообщения может выглядеть, например, так:

```
printf( rus("Скорость: %3.2f км/час"), v);
```

В качестве примера консольного приложения в листинге 7.1 приведена программа "Угадай число", которая для вывода сообщений использует функцию RUS. Значение функции rus — строка символов в кодировке ASCII, соответствующая строке, полученной в качестве параметра.

Листинг 7.1. Пример консольного приложения

```
#pragma hdrstop

#include <stdio.h>
#include <conio.h>    // для доступа к getch()
#include <stdlib.h>   // для доступа к srand(), rand()
#include <time.h>     // для доступа к time_t и time()

char* rus(char* st); // преобразует ANSI-строку в строку ASCII

#pragma argsused

int main(int argc, char* argv[])
{
    int comp,   // число, "задуманное" компьютером
        igrok,  // вариант игрока
        n=0;    // число попыток

        // ГСЧ — генератор случайных чисел
        time_t t;  // текущее время (для инициализации ГСЧ)

        srand( (unsigned)time(&t)) ; // инициализация ГСЧ
        comp = rand() % 10 + 1;
        puts ( rus ( "\nКомпьютер \"задумал\" число от 1 до 10."));
        puts ( rus ( "Вы должны его угадать за три попытки."));
        do
        {
                printf ("->");
                scanf("%i", &igrok);
                n++;
        }
        while ( igrok != comp && n < 3) ;

        if (igrok == comp)
                printf( rus("ВЫ ВЫИГРАЛИ!"));
        else {
                puts( rus("Вы проиграли."));
                printf( rus("Компьютер \"задумал\" число %d"), comp);
        }
        printf( rus("\nДля завершения нажмите любую клавишу..."));
        getch();
        return 0;
}
```

```
/*    Функция rus преобразует ANSI-строку в строку ASCII
      и может использоваться для вывода сообщений
      на русском языке в консольных программах.

      Пример использования:
            printf(rus("Скорость:  %3.2f км/час"), v);
          printf( rus("У лукоморья дуб зеленый\n"));
V
char* rus(char* st)
{
      unsigned char* p = st;
      /* при объявлении символов как char русские буквы
         кодируются отрицательными числами */
      while ( *p)
      {
          if (*p >= 192)  // здесь русская буква
                if (*p<= 239)  // А,Б, ...Я,а,б, ...п
                      *p -= 64;
                else // р ... я
                      *p -= 16;
          p++;
      }
      return st;
}
```

Компиляция консольного приложения выполняется обычным образом, т. е. выбором из меню **Project** команды **Compile.**

После успешной компиляции программа может быть запущена выбором из меню **Run** команды **Run.** При запуске консольного приложения на экране появляется стандартное окно командной строки. На рис. 7.4 приведен вид окна командной строки, в котором работает консольное приложение, созданное в C++ Builder.

Процесс сохранения проекта консольного приложения стандартный. В результате выбора из меню **File** команды **Save Project** на экране сначала появляется окно **Save Project,** в котором нужно ввести имя проекта, а затем – окно **Save Utit,** в котором надо задать имя модуля.

Получить доступ к модулю консольного приложения (тексту программы) для того, чтобы внести изменения в программу, несколько сложнее. Сначала, выбрав в меню **File** команду **Open Project,** нужно открыть файл проекта. Затем надо открыть окно **Project Manager** (команда **View | Project Manager),**

раскрыть список файлов, выбрать cpp-файл и из контекстного меню выбрать команду **Open** (или сделать двойной щелчок на имени cpp-файла) (рис. 7.5).

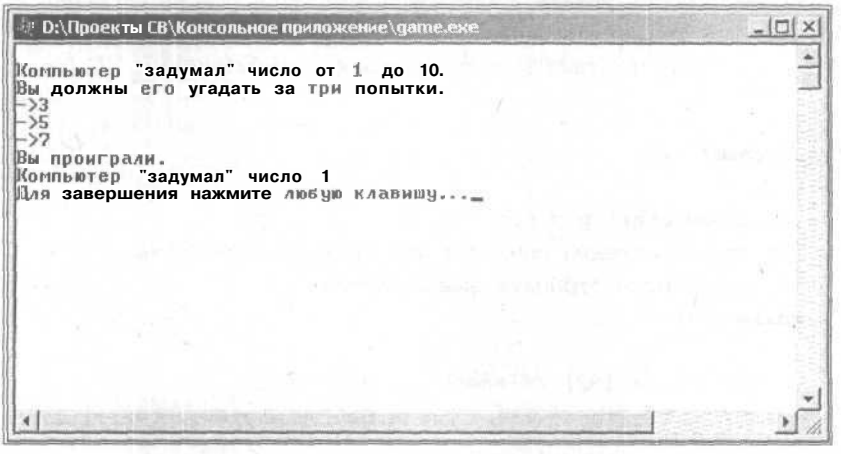


Рис. 7.4. Окно командной строки, в котором работает консольное приложение

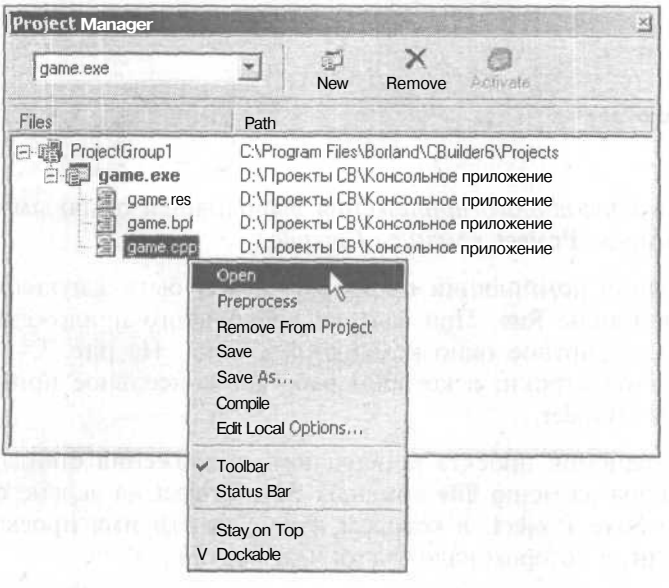


Рис. 7.5. Чтобы внести изменения в программу, надо в окне **Project Manager** выбрать cpp-файл и из контекстного меню выбрать команду **Open**

ГЛАВА 8

Справочная система

Каждая программа должна обеспечивать пользователя справочной информацией. Существует два способа отображения справочной информации: классический (рис. 8.1, левый) и современный, в "интернет-стиле" (рис. 8.1, правый). Классический способ отображения справочной информации применяется большинством приложений, в том числе и C++ Builder. Отображение справочной информации в интернет-стиле используется в программных продуктах Microsoft и, в последнее время, в продуктах других разработчиков программного обеспечения.

Классическая справочная система представляет собой набор файлов, используя которые программа Winhelp, являющаяся составной частью Windows, выводит справочную информацию. Основой такой справочной системы являются hip-файлы.

Основой современной справочной системы являются chm-файлы. Chm-файл представляет собой компилированный HTML-документ, полученный путем компиляции (объединения) файлов, составляющих HTML-документ, в том числе и файлов иллюстраций.

Создать hip-файл можно при помощи утилиты Microsoft Help Workshop, которая входит в комплект C++ Builder (файл утилиты hce.exe находится в каталоге \CBuilder\Help\Tools). Chm-файл можно создать при помощи утилиты Microsoft HTML Help Workshop, которая, к сожалению, в состав C++ Builder не включена.

Рассмотрим процесс создания справочной системы, сначала классической, а затем — современной.

Создание справочной системы при помощи Microsoft Help Workshop

Процесс создания справочной системы состоит из двух этапов. На первом этапе надо подготовить справочную информацию, на втором — преобразо-

вать справочную информацию в справочную систему. Задача первого этапа может быть решена при помощи редактора текста, второго — посредством утилиты Microsoft Help Workshop.

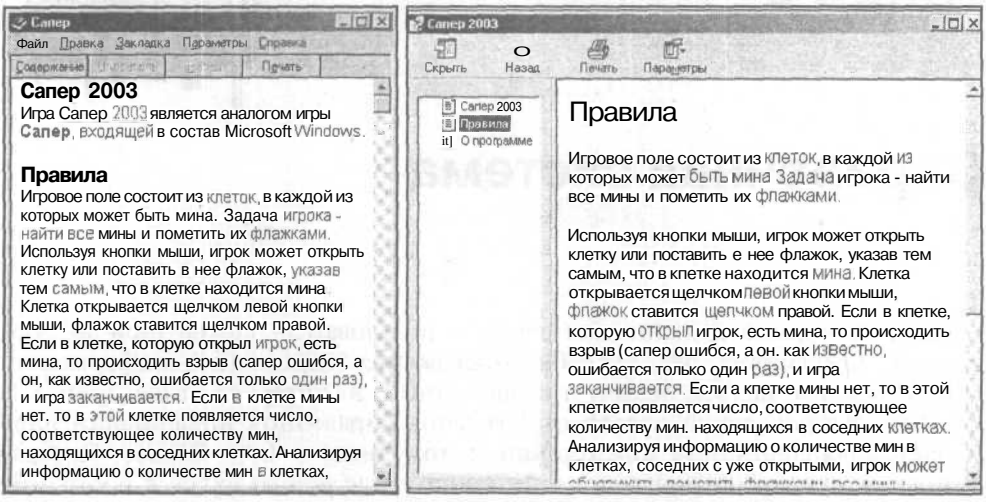


Рис. 8.1. Два способа представления справочной информации: классический и современный

Подготовка справочной информации

Исходным материалом для Microsoft Help Workshop является справочная информация, представленная в виде rtf-файла. Наиболее просто подготовить rtf-файл справочной информации можно при помощи Microsoft Word.

Сначала нужно набрать текст разделов справки. Заголовки разделов нужно оформить одним из стилей **Заголовок.** Каждый раздел должен заканчиваться символом "разрыв страницы".

После того как текст разделов будет набран, каждому разделу надо назначить *идентификатор.* Идентификатор назначается путем вставки перед заголовком раздела сноски #.

Для того чтобы назначить разделу идентификатор, нужно установить курсор перед первой буквой заголовка раздела, затем в меню **Вставка** выбрать команду **Сноска,** а появившемся диалоговом окне **Сноски** выбрать в группе **Нумерация** положение **другая,** в поле редактирования ввести символ "#" (рис. 8.2).

В результате щелчка на кнопке OK в документ будет вставлена сноска #, а в нижней части окна документа откроется окно ввода текста сноски. В этом окне рядом со значком сноски надо ввести идентификатор раздела (рис. 8.3). Рекомендуется, чтобы идентификатор раздела состоял из префикса `IDH_` и порядкового номера раздела, например, `IDH_1`, `IDH_2` и т. д.

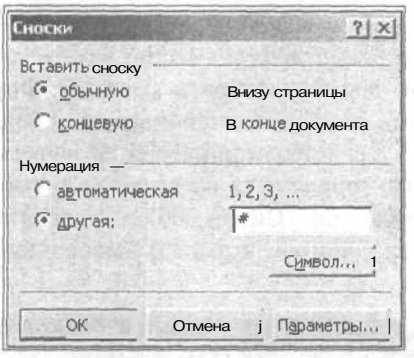


Рис. 8.2. Чтобы задать идентификатор раздела, надо перед заголовком раздела вставить сноску #

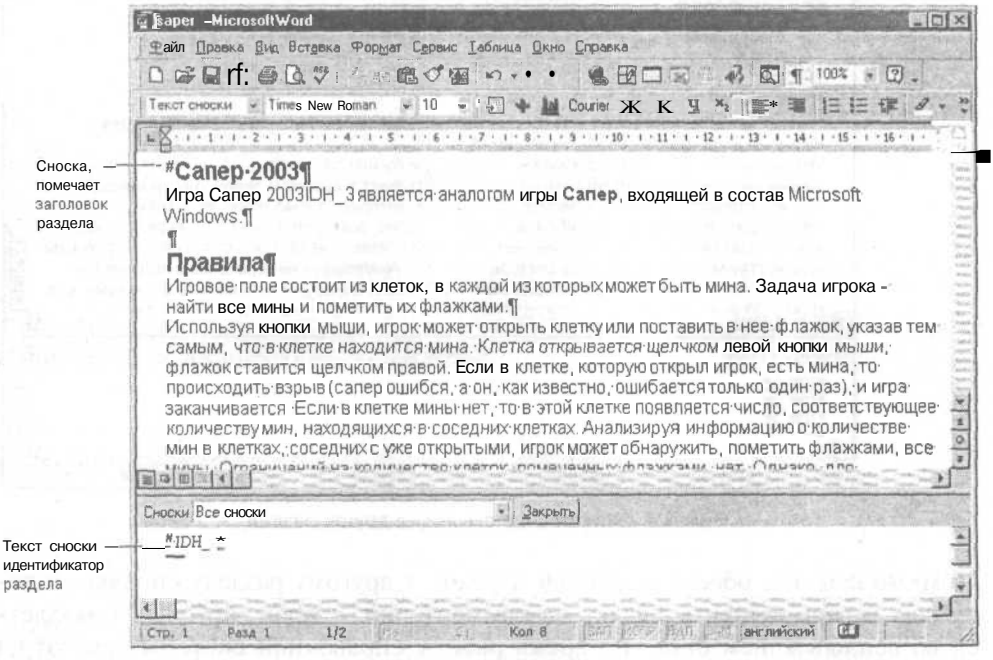


Рис. 8.3. Вставка в документ сноски, помечающей заголовок раздела справки

Обычно в тексте справочной информации есть *ссылки,* которые обеспечивают переход к другому разделу, связанному с тем, который в данный момент видит пользователь.

Во время подготовки текста справочной информации слово-ссылку следует подчеркнуть двойной линией и сразу за этим словом, без пробела, поместить идентификатор раздела справки, к которому должен быть выполнен

переход в результате выбора ссылки. Вставленный идентификатор необходимо оформить как *скрытый текст*. Чтобы задать двойное подчеркивание, нужно выделить слово-ссылку, выбрать команду **Формат | Шрифт** и в появившемся окне выбрать способ подчеркивания. Аналогичным образом надо задать "скрытый текст" для идентификатора. В качестве примера на рис. 8.4 приведен вид окна редактора текста во время подготовки файла справочной информации для программы "Сапер 2003". Название игры помечено как ссылка на другой раздел справки, который имеет идентификатор `IDH_3`.

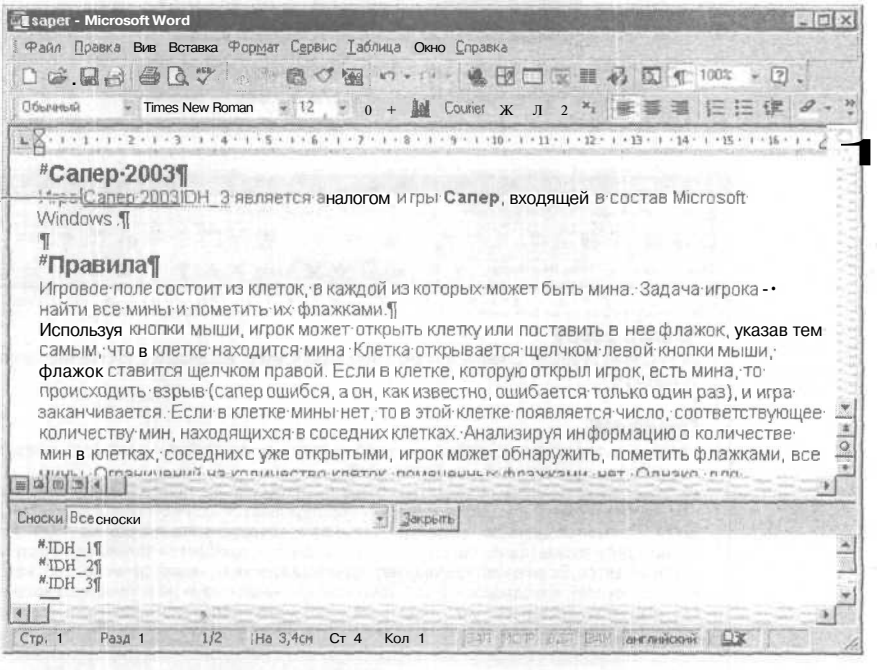


Рис. 8.4. Оформление ссылки на другой раздел

Помимо ссылки, обеспечивающей переход к другому разделу справки, в документ можно вставить ссылку на комментарий — текст, который появляется во всплывающем окне. Во время работы справочной системы ссылки на комментарии выделяются цветом и подчеркиваются пунктирной линией. При подготовке документа справочной системы комментарии, как и разделы справки, располагают на отдельной странице, однако текст комментария не должен иметь заголовка. Сноска # должна быть поставлена перед текстом комментария. Ссылка на комментарий оформляется следующим образом: сначала надо подчеркнуть одинарной линией слово, выбор которого должен вызвать появление комментария, затем сразу после этого слова вставить идентификатор комментария, оформив его как скрытый текст.

Следует отметить, что справочная информация может быть распределена по нескольким файлам.

Проект справочной системы

Преобразование файла справочной информации в файл справочной системы выполняет входящий в состав Microsoft Help Workshop компилятор. Исходными данными для компилятора является справочная информация, представленная в виде rtf-файлов, и файл проекта справочной системы.

Для того чтобы преобразовать справочную информацию, подготовленную в редакторе текста, в справочную систему, сначала надо создать файл проекта справочной системы. Для этого нужно запустить Microsoft Help Workshop и в меню **File** выбрать команду New | **Help Project** (рис. 8.5). На экране появится диалоговое окно **Project File Name.** В этом окне (рис. 8.6) надо открыть папку, в которой находится файл справочной информации (rtf-файл), задать имя проекта и щелкнуть на кнопке **Сохранить.** В результате этих действий будет создан файл проекта (hpj-файл) и станет доступным окно проекта справочной системы (рис. 8.7).

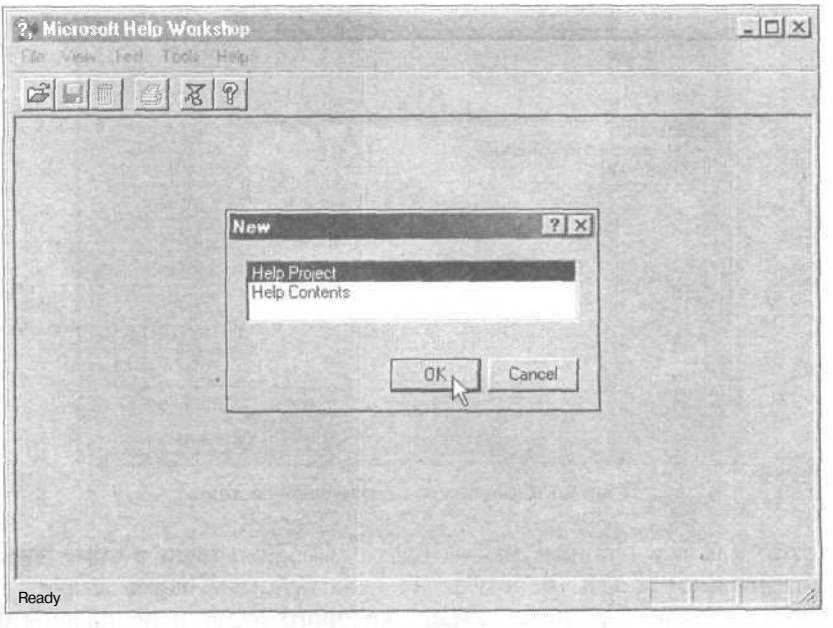


Рис. 8.5. Начало работы над новым проектом

Первое, что надо сделать, — это добавить в проект rtf-файл, в котором находится справочная информация. Для этого нужно сначала щелкнуть на кнопке **Files,** затем в открывшемся окне **Topic Files** — на кнопке Add (рис. 8.8).

На экране появится стандартное диалоговое окно **Открытие файла,** исполь зуя которое можно выбрать нужный rtf-файл. Если справочная информаци распределена по нескольким файлам, то операцию добавления файла нужн повторить.

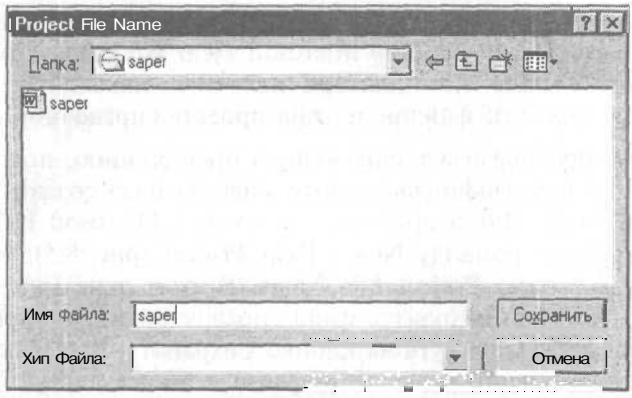


Рис. 8.6. В поле **Имя файла** надо ввести название проекта

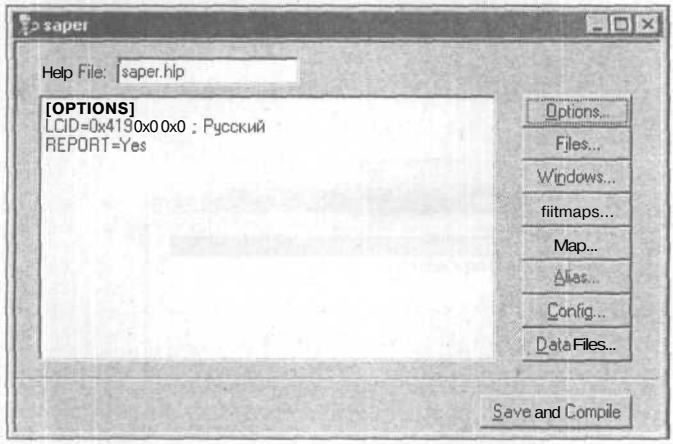


Рис. 8.7. Окно проекта справочной системы

После того как все нужные файлы будут выбраны, надо в окне **Topic Files** щелкнуть на кнопке OK. В результате этих действий вновь станет доступным окно проекта, в разделе [FILES] которого будут перечислены файлы, в которых находится справочная информация.

Следующее, что надо сделать, — это назначить числовые значения идентификаторам разделов справочной информации. Для этого в окне проекта надо сначала щелкнуть на кнопке **Map,** затем, в открывшемся окне Map, — на

кнопке Add (рис. 8.8). На экране появится окно **Add Map Entry** (рис. 8.9). В поле **Topic ID** этого окна надо ввести идентификатор раздела справки (идентификаторы были назначены разделам во время создания rtf-файла), в поле **Mapped numeric value** — число, идентифицирующее раздел. В поле **Comment** можно ввести комментарий — название раздела справки. После того как будут назначены числовые значения идентификаторам разделов справки, окно Map можно закрыть.

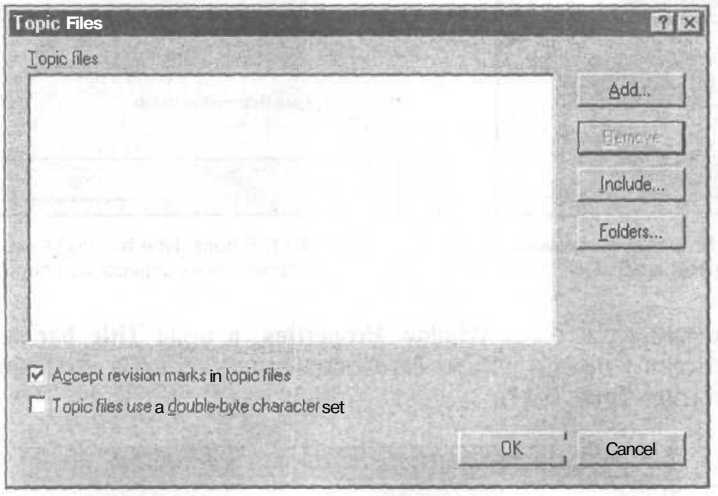


Рис. 8.8. Чтобы добавить в проект rtf-файл, щелкните на кнопке **Add**

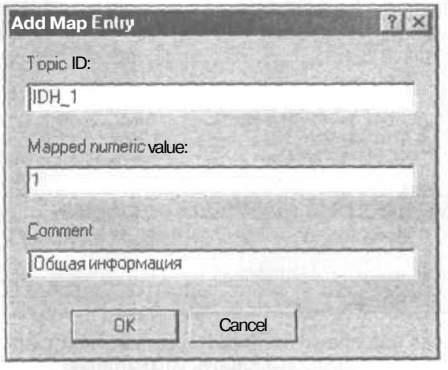


Рис. 8.9. Назначение идентификатору раздела числового значения

Последнее, что надо сделать, — это настроить вид окна справочной информации. Для этого надо в окне проекта щелкнуть на кнопке **Windows**, в поле **Create a window named** открывшегося окна **Create a window** (рис. 8.10) ввести слово main и щелкнуть на кнопке OK.

Рис. 8.10. Диалоговое окно **Create a window**

Рис. 8.11. В поле **Title bar text** надо ввести заголовок окна справочной системы

На экране появится окно **Window Properties,** в поле **Title bar text** вкладки **General** которого нужно ввести заголовок главного окна создаваемой справочной системы (рис. 8.11).

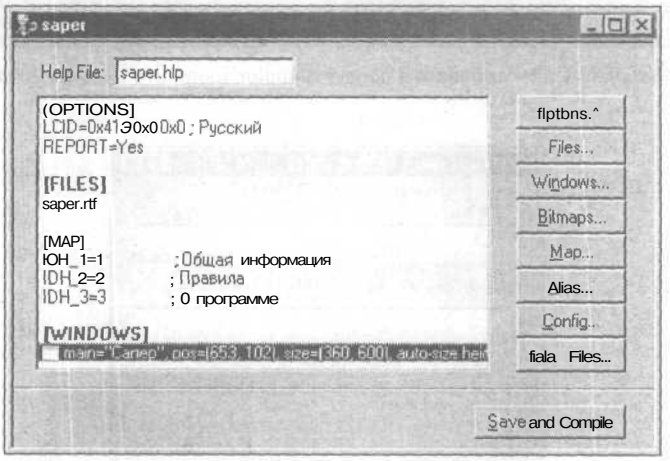


Рис. 8.12. Работа над проектом простой справочной системы завершена; можно выполнить компиляцию

На этом процесс создания проекта простой справочной системы можно считать завершенным (рис. 8.12). Теперь можно выполнить компиляцию. Для этого надо в меню **File** выбрать команду **Compile,** в появившемся диалоговом окне **Compile a Help File** установить флажок **Automatically display Help file in WinHelp when done** (Автоматически показывать созданную справочную

систему по завершении компиляции) и щелкнуть на кнопке **Compile** (рис. 8.13).

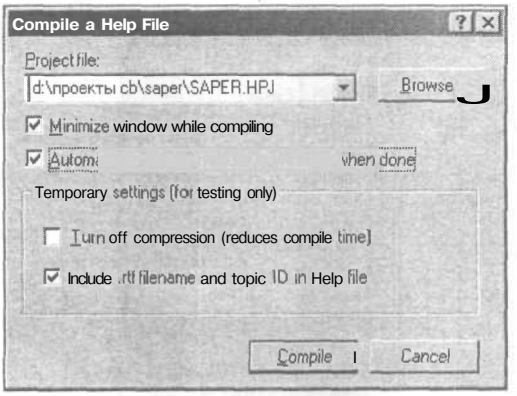


Рис. 8.13. Чтобы выполнить компиляцию, надо щелкнуть на кнопке **Compile**

По завершении компиляции на экране появится окно с информационным сообщением о результатах компиляции и, если компиляция выполнена успешно, окно созданной справочной системы. Файл справочной системы (HLP-файл) компилятор поместит в ту папку, в которой находится файл проекта.

Вывод справочной информации

Обычно окно справочной системы становится доступным в результате нажатия клавиши <F1> или выбора в меню **Справка** команды ?.

Для того чтобы во время работы программы пользователь, нажав клавишу <F1>, мог получить справочную информацию, надо чтобы свойство `HelpFile` главного окна приложения содержало имя файла справочной системы, а свойство `HelpContext` - числовой идентификатор нужного раздела (рис. 8.14).

Для каждого компонента формы можно задать свой раздел справки. Раздел справки, который появляется, если фокус находится на компоненте и пользователь нажимает клавишу <F1>, определяется значением свойства HeipContext этого компонента. Если значение свойства `HelpContext` элемента управления равно нулю, то при нажатии клавиши <F1> появляется тот раздел справки, который задан для формы приложения.

Для того чтобы справочная информация появилась на экране в результате выбора в меню ? команды **Справка,** надо создать функцию обработки события Onclick для соответствующей команды меню. Процесс создания функции обработки события для команды меню ничем не отличается от процесса создания функции обработки события для элемента управления, например, для командной кнопки: в списке объектов надо выбрать объект типа

TmenuItem, для которого создается функция обработки события, а во вкладке **Events** — событие.

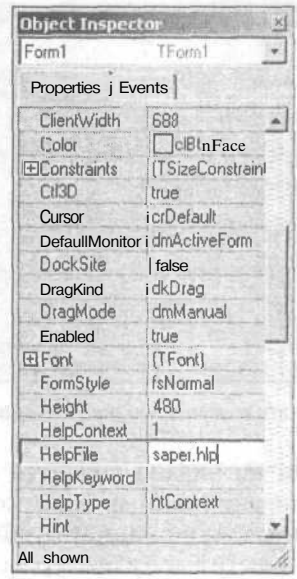


Рис. 8.14. Свойство HelpFile должно содержать имя файла справки, а свойство HelpContext — идентификатор раздела

Ниже приведена функция обработки события Onclick для команды **Справка** меню ?.

```
// выбор в меню ? команды Справка
void __fastcall TForm1::N3Click(TObject *Sender)
{
  WinHelp(Form1->Handle,"saper.hlp",HELP_CONTEXT,1);
}
```

Вывод справочной информации выполняет функция WinHelp, которой в качестве параметра передается идентификатор окна программы, которая запрашивает вывод справочной информации, файл справки, константу HELP_CONTEXT и идентификатор раздела, содержимое которого должно быть отражено.

HTML Help Workshop

Помимо стандартного, классического способа представления справочной информации, в современных программах все чаще используется представление информации в интернет-стиле (рис. 8.15).

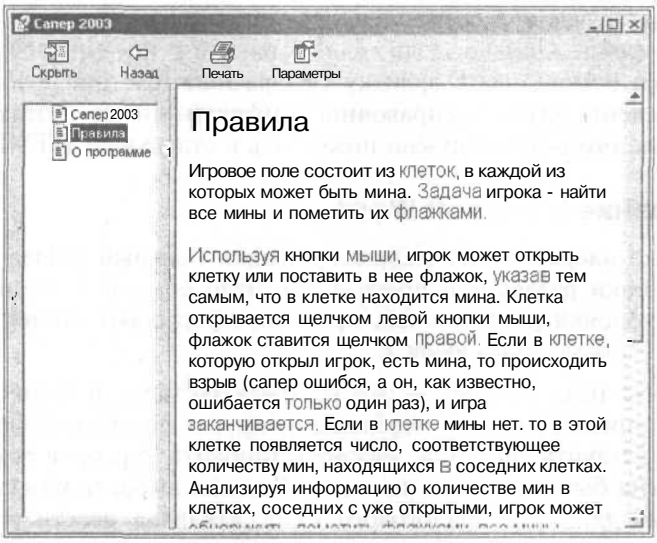


Рис. 8.15. Современный, в интернет-стиле способ отображения справочной информации

Основой современной справочной системы являются *компилированные* HTML-документы — файлы с расширением chm. Chm-файл получается путем компиляции (объединения) файлов, составляющих HTML-документ.

Отображение справочной информации обеспечивает операционная система.

Создать chm-файл можно при помощи утилиты Microsoft HTML Help Workshop. Исходной информацией для компилятора справочной системы являются файлы HTML, файлы иллюстраций и файл проекта.

Чтобы получить chm-файл, надо:

О подготовить справочную информацию в виде набора HTML-документов;

- ❒ создать файл проекта;
- ❒ создать файл контекста (содержания);
- ❒ выполнить компиляцию.

Последние три из перечисленных выше шагов выполняются в программе HTML Help Workshop.

Подготовка справочной информации

Подготовить справочную информацию в HTML-формате можно при помощи любого редактора текста. Наиболее быстро это можно сделать, если редактор позволяет сохранить набранный текст как HTML-документ. Можно воспользоваться и встроенным редактором Microsoft HTML Help Workshop, но для этого надо знать язык HTML (по крайней мере, его основы).

В простейшем случае вся справочная информация может быть помещена в один HTML-файл. Однако если для навигации по справочной системе предполагается использовать вкладку **Содержание** (см. рис. 8.15), в которой будут перечислены разделы справочной информации, то в этом случае информацию каждого раздела нужно поместить в отдельный HTML-файл.

Использование Microsoft Word

Сначала нужно набрать текст разделов справки (каждый раздел в отдельном файле). Заголовки разделов и подразделов надо оформить одним из стилей **Заголовок.** Заголовки разделов, как правило, оформляют стилем **Заголовок1**, подразделов — стилем **Заголовок 2.**

Следующее, что надо сделать, — это вставить закладки в те точки документа, в которые предполагаются переходы из других разделов справочной системы. Чтобы вставить закладку, нужно установить курсор в точку текста, в которой должна быть закладка, из меню **Вставка** выбрать команду **Закладка** и в поле **Имя закладки** диалогового окна **Закладка** ввести имя закладки (рис. 8.16).

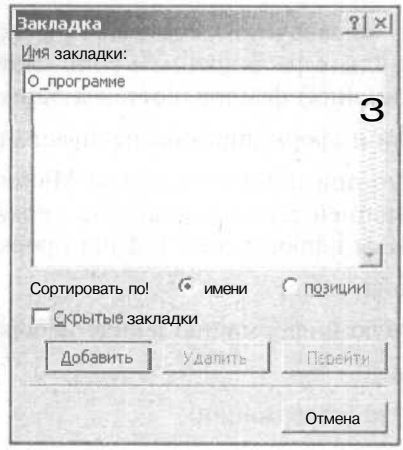


Рис. 8.16. Добавление закладки

Имя закладки должно отражать суть предполагаемого перехода к закладке, содержимое помечаемого фрагмента текста. В имени закладки пробел использовать нельзя. Вместо пробела можно поставить символ подчеркивания. Заголовки, оформленные стилем **Заголовок,** помечать закладками не надо. Таким образом, если в создаваемой справочной системе предполагаются переходы только к заголовкам разделов справочной информации, закладки можно не вставлять.

После того как будут расставлены закладки, можно приступить к расстановке ссылок. Различают ссылки, которые обеспечивают навигацию внутри

раздела, и те, которые обеспечивают переход к другому разделу справочной системы.

Чтобы вставить ссылку, обеспечивающую навигацию внутри раздела, надо выделить фрагмент текста (слово или фразу), при выборе которого должен быть выполнен переход, из меню **Вставка** выбрать команду **Гиперссылка,** в появившемся окне **Добавление гиперссылки** (рис. 8.17) сначала щелкнуть на кнопке **Связать с местом в этом документе,** затем — выбрать закладку или заголовок, к которому должен быть выполнен переход.

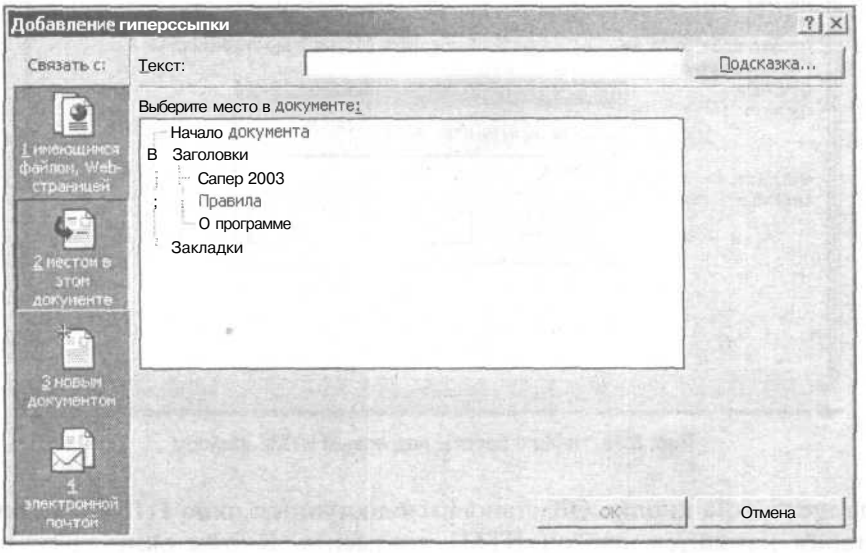


Рис. 8.17. Выбор точки документа для перехода по ссылке

Если нужно вставить в документ ссылку на раздел справки, который находится в другом файле, то в диалоговом окне **Добавление гиперссылки** надо щелкнуть на кнопке **Файл** и в появившемся стандартном окне выбрать имя нужного HTML-файла.

После того как в документ будут помещены все необходимые гиперссылки, документ нужно сохранить в HTML-формате.

Использование HTML Help Workshop

Использование HTML-редактора, входящего в состав HTML Help Workshop, предполагает знание основ HTML — языка гипертекстовой разметки (далее приведены краткие сведения об HTML, которых достаточно для того, чтобы создать вполне приличную справочную систему).

Чтобы создать HTML-файл, надо запустить HTML Help Workshop, из меню **File** выбрать команду New | **HTML File** и в появившемся окне **HTML Title**

(рис. 8.18) задать название раздела справки, текст которого будет находиться в создаваемом файле.

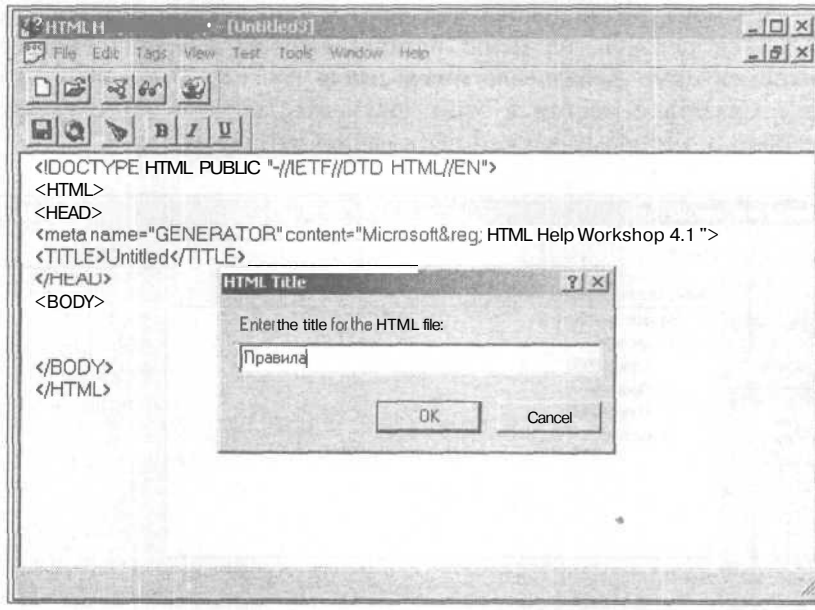


Рис. 8.18. Начало работы над новым HTML-файлом

После щелчка на кнопке OK становится доступным окно HTML-редактора, в котором находится шаблон HTML-документа. В этом окне, сразу после строки <BODY>, можно набирать текст.

Основы HTML

HTML-документ представляет собой текст, в который помимо обычного текста включены специальные последовательности символов — *теги*. Тег начинается символом < и заканчивается символом >. Теги используются программами отображения HTML-документов для форматирования текста в окне просмотра (сами теги не отображаются).

Большинство тегов парные. Например, пара тегов <H2> </H2> сообщает программе отображения HTML-документа, что текст, который находится между этими тегами, является заголовком второго уровня и должен быть отображен соответствующим стилем.

В табл. 8.1 представлен минимальный набор тегов, используя которые можно подготовить HTML-файл с целью дальнейшего его преобразования в chm-файл справочной системы.

Таблица 8.1. *HTML-теги*

Тег	Пояснение
<TITLE> Название </TITLE>	Задает название HTML-документа. Программы отображения HTML-документов, как правило, выводят название документа в заголовке окна, в котором документ отображается. Если название не задано, то в заголовке окна будет выведено название файла
<BODY BACKGROUND = "Файл" BGCOLOR="Цвет" TEXT="Цвет">	Параметр BACKGROUND задает фоновый рисунок, BGCOLOR — цвет фона, TEXT — цвет символов HTML-документа
<BASEFONT FACE="Шрифт" SIZE=n>	Задает основной шрифт, который используется для отображения текста: FACE — название шрифта, SIZE — размер в относительных единицах. По умолчанию значение параметра SIZE равно 3. Размер шрифта заголовков (см. тег <н>) берется от размера, заданного параметром SIZE
<H1> </H1>	Определяет текст, находящийся между тегами <H1> и </H1>, как заголовок уровня 1. Пара тегов <H2> </H2> определяет заголовок второго уровня, а пара <нз> </нз> — третьего
 	Конец строки. Текст, находящийся после этого тега, будет выведен с начала новой строки
<P> </P>	Текст, находящийся внутри этих тегов, является параграфом
<B> </B>	Текст, находящийся внутри этой пары тегов, будет выделен полужирным
<I> </I>	Текст, находящийся внутри этой пары тегов, будет выделен курсивом
<A NAME="Закладка"> </A>	Помечает фрагмент документа закладкой. Имя закладки задает параметр NAME. Это имя используется для перехода к закладке
<A HREF="Файл.htm#Закладка"> </A>	Выделяет фрагмент документа как гиперссылку, при выборе которой происходит перемещение к закладке, имя которой указано в параметре HREF
<IMG SRC="Иллюстрация">	Выводит иллюстрацию, имя файла которой указано в параметре SRC
<!-- -->	Комментарий. Текст, находящийся между дефисами, на экран не выводится

Набирается HTML-текст обычным образом. Теги можно набирать как прописными, так и строчными буквами. Однако, для того чтобы лучше была видна структура документа, рекомендуется записывать все теги строчными (большими) буквами. Следующее, на что надо обратить внимание — это то, что программы отображения HTML-документов игнорируют "лишние" пробелы и другие "невидимые" символы (табуляция, новая строка). Это значит, что для того чтобы фрагмент документа начинался с новой строки, в конце предыдущей строки надо поставить тег
, а для того чтобы между строками текста появилась пустая строка, в HTML-текст надо вставить два тега
 подряд.

Работая с HTML-редактором в программе HTML Help Workshop, уже в процессе набора HTML-текста можно увидеть, как будет выглядеть набираемый текст. Для этого надо из меню View выбрать команду In Browser или щелкнуть на командной кнопке, на которой изображен стандартный значок Internet Explorer.

В качестве примера на рис. 8.19 приведен текст одного из разделов справочной системы программы "Сапер 2003".

```
<HTML>
<TITLE>Правила</TITLE>
<BODY BGCOLOR=#FFFFFF BACKGROUND =""">
<BASEFONT FACE="Arial" SIZE=2>
<A NAME="Правила"><H2>Правила</H2></A><P>Игровое поле состоит из
клеток, в каждой из которых может быть мина. Задача игрока — найти все
мины и пометить их флажками.</P> <P>Используя кнопки мыши, игрок может
открыть клетку или поставить в нее флажок, указав тем самым, что в
клетке находится мина. Клетка открывается щелчком левой кнопки мыши,
флажок ставится щелчком правой. Если в клетке, которую открыл игрок,
есть мина, то происходить взрыв (сапер ошибся, а он, как известно,
ошибается только один раз), и игра заканчивается. Если в клетке мины
нет, то в этой клетке появляется число, соответствующее количеству мин,
находящихся в соседних клетках. Анализируя информацию о количестве мин
в клетках, соседних с уже открытыми, игрок может обнаружить, пометить
флажками, все мины. Ограничений на количество клеток, помеченных
флажками, нет. Однако, для завершения игры (выигрыша) флажки должны
быть установлены только в тех клетках, в которых есть мины. Ошибочно
установленный флажок можно убрать, щелкнув правой кнопкой мыши
в клетке, в которой он находится.</P>
<I>См.</I><BR>
<A HREF="saper_01.htm#Сапер_2003">Сапер 2003</AXBR>
<A HREF="saper_03.htm#О_программе">О программе</A><BR>
</BODY>
</HTML>
```

Рис. 8.19. HTML-текст раздела справочной системы

Создание файла справки

После того как справочная информация будет подготовлена, можно приступить к непосредственному созданию справочной системы.

Сначала надо запустить HTML Help Workshop, из меню **File** выбрать команду New | **Project** и в окне New **Project** задать имя файла проекта создаваемой справочной системы (рис. 8.20). После щелчка на кнопке **Далее** в этом и следующем окнах окно **HTML Help Workshop** должно выглядеть так, как показано на рис. 8.21.

Рис. 8.20. Начало работы над новым проектом

Первое, что надо сделать, — это сформировать раздел [FILES], который должен содержать имена HTML-файлов, в которых находится справочная информация. Чтобы добавить в раздел [FILES] имя файла, надо щелкнуть на кнопке **Add/Remove topic files** (см. рис. 8.21), а затем в появившемся диалоговом окне **Topic Files** (рис. 8.22) — на кнопке **Add,** после чего в появившемся стандартном диалоговом окне **Открыть** выбрать HTML-файл раздела справки. Если справочная информация распределена по нескольким файлам, то операцию добавления нужно повторить несколько раз. После того как в диалоговом окне **Topic Files** будут перечислены все необходимые для создания справочной информации HTML-файлы, нужно щелкнуть на кнопке OK. В результате этих действий в файле проекта появится раздел [FILES], в котором будут перечислены HTML-файлы, используемые для создания справочной системы (рис. 8.23).

Следующее, что надо сделать, — это задать главный (стартовый) раздел и заголовок окна справочной системы. Заголовок и имя файла главного раздела вводятся соответственно в поля **Title и Default file** вкладки **General** диа-

логового окна **Options** (рис. 8.24), которое появляется в результате щелчка на кнопке **Change project options** (см. рис. 8.21).

Если для навигации по справочной системе предполагается использовать вкладку **Содержание,** то надо создать *файл контекста.* Чтобы это сделать, нужно щелкнуть на вкладке **Contents,** подтвердить создание нового файла и

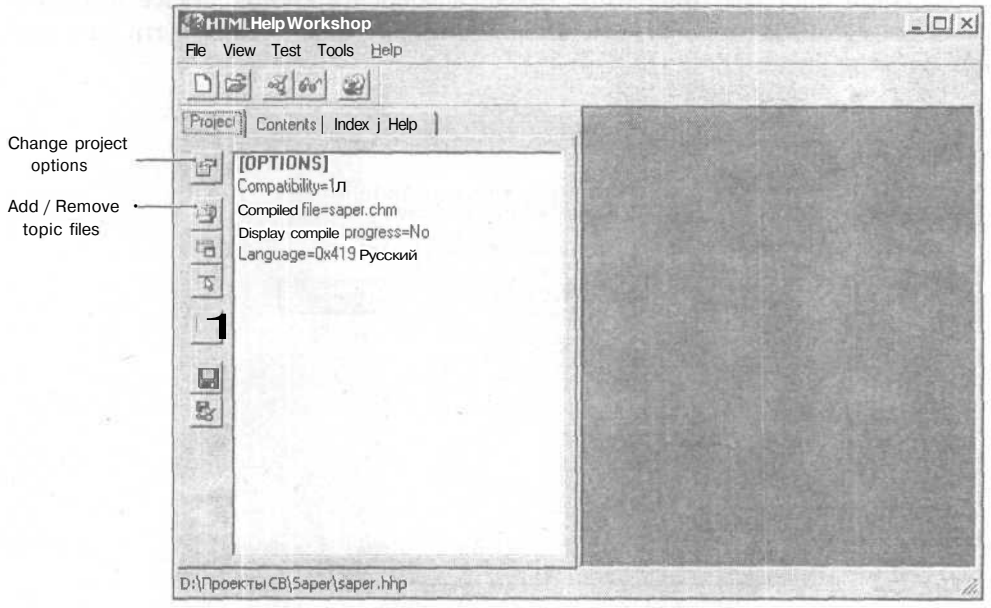


Рис. 8.21. Окно **HTML Help Workshop** в начале работы над новым проектом

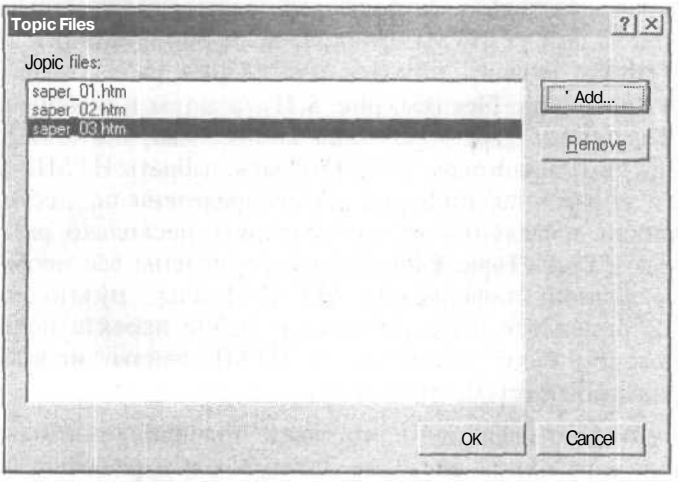


Рис. 8.22. Диалоговое окно **Topic Files**

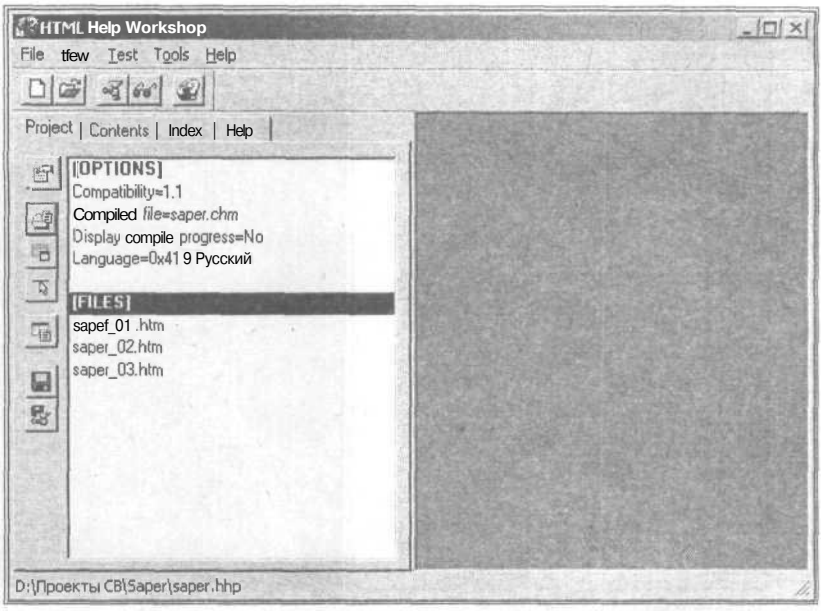


Рис. 8.23. В разделе [FILES] перечислены файлы, используемые для создания chm-файла

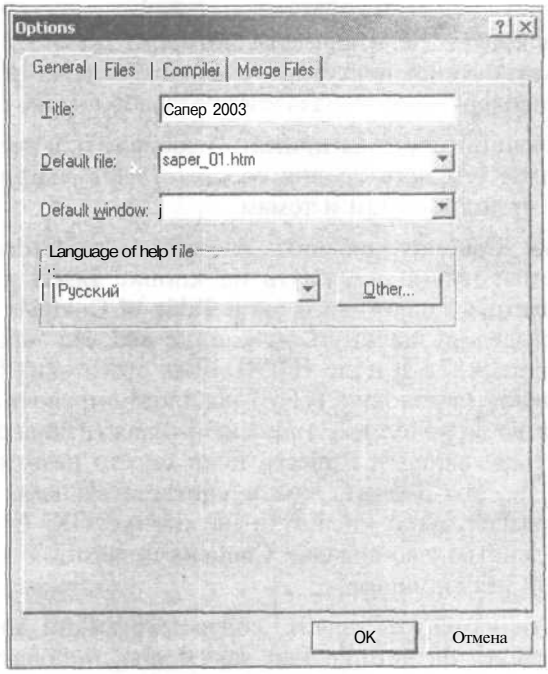


Рис. 8.24. В диалоговом окне **Options** надо задать заголовок окна справочной системы и файл главного раздела

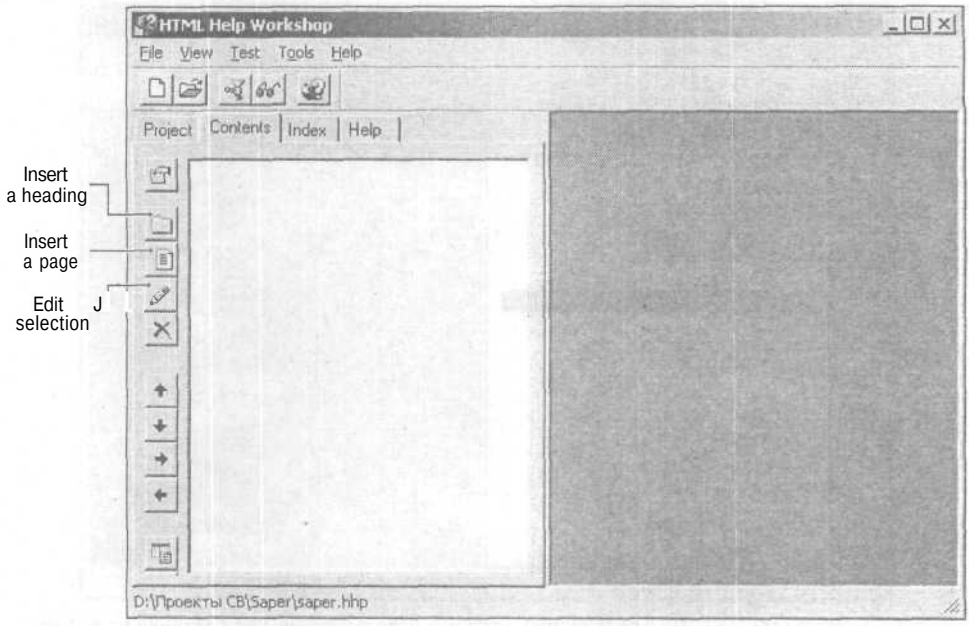


Рис. 8.25. Вкладка **Contents**

задать имя файла контекста, в качестве которого можно использовать имя проекта. В результате станет доступной вкладка **Contents** (рис. 8.25), в которую нужно ввести содержание — названия разделов справочной системы.

Содержание справочной системы принято изображать в виде иерархического списка. Элементы верхнего уровня соответствуют разделам, а подчиненные им элементы — подразделам и темам.

Чтобы во вкладку **Contents** добавить элемент, соответствующий разделу справочной системы, нужно щелкнуть на кнопке **Insert a heading,** в поле **Entry title** появившегося диалогового окна **Table of Contents Entry** (рис. 8.26) ввести название раздела и щелкнуть на кнопке Add. На экране появится окно **Path or URL** (рис. 8.27). В поле **HTML titles** этого окна будут перечислены названия разделов (заголовки HTML-файлов) справочной информации, которая находится во включенных в проект файлах (имена этих файлов указаны в разделе [FILES] вкладки **Project).** Если вместо названия раздела будет указано имя файла, это значит, что в соответствующем файле нет тега <TITLE>. Выбрав раздел, надо щелкнуть на кнопке OK. В результате перечисленных выше действий во вкладке **Contents** появится строка с названием раздела справочной информации.

При необходимости изменить значок, соответствующий добавленному разделу, следует щелкнуть на кнопке **Edit selection** и, используя список **Image index** вкладки **Advanced** окна **Table of Contents,** выбрать нужный значок (обычно рядом с названием раздела или подраздела изображена книжка).

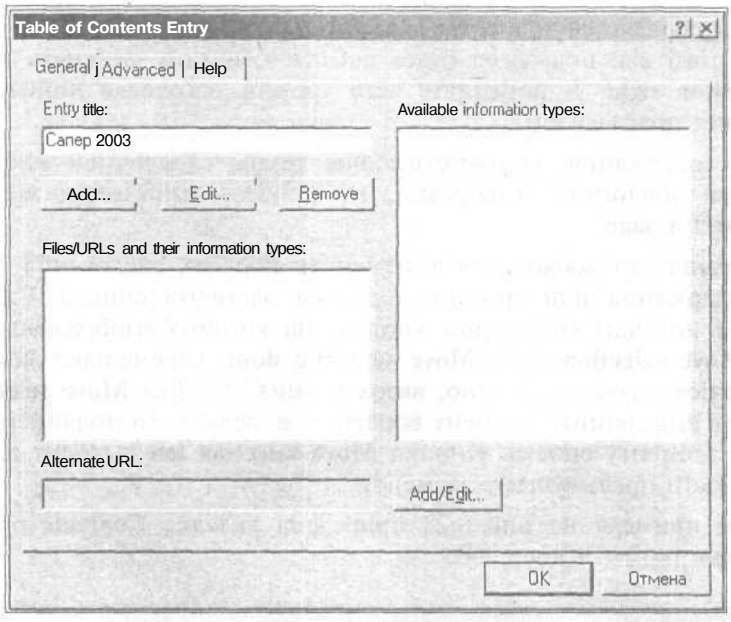


Рис. 8.26. Добавление элемента в список разделов

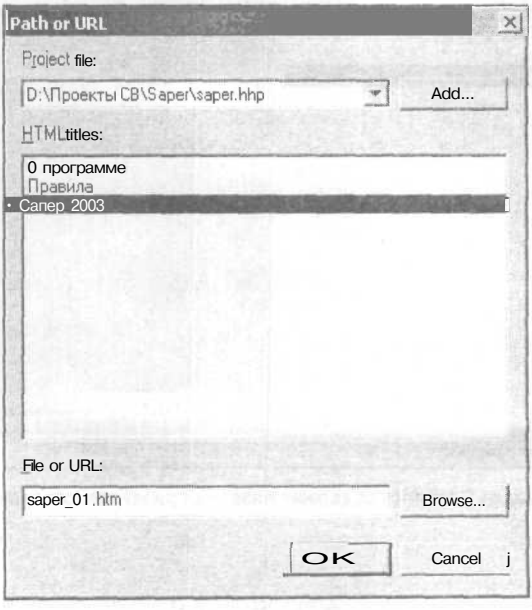


Рис. 8.27. Выбор файла, соответствующего элементу списка разделов

Подраздел добавляется точно так же, как и раздел, с тем только отличием, что после того как подраздел будет добавлен, нужно щелкнуть на кнопке **Move selection right.** В результате чего уровень заголовка понизится, т. е. раздел станет подразделом.

Элементы содержания, соответствующие темам справочной информации, добавляются аналогичным образом, но процесс начинается щелчком на кнопке **Insert a page.**

Иногда возникает необходимость изменить порядок следования элементов списка содержания или уровень иерархии элемента списка. Сделать это можно при помощи командных кнопок, на которых изображены стрелки. Кнопки **Move selection up и Move selection down** перемещают выделенный элемент списка, соответственно, вверх и вниз. Кнопка **Move selection right** перемещает выделенный элемент вправо, т. е. делает его подчиненным предыдущему элементу списка. Кнопка **Move selection left** выводит элемент из подчиненности предыдущему элементу.

В качестве примера на рис. 8.28 приведена вкладка **Contents** справочной системы программы "Сапер 2003".

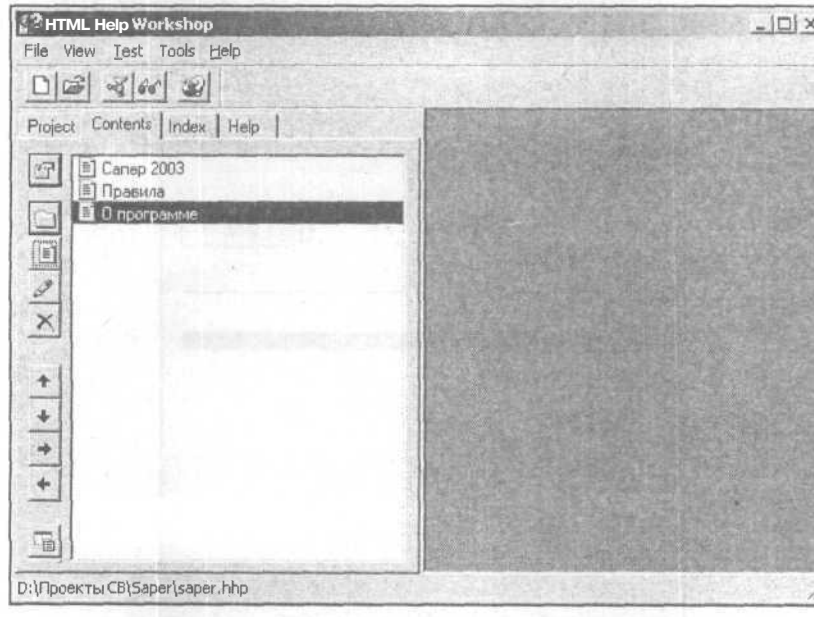


Рис. 8.28. Вкладка **Contents** содержит названия разделов справочной системы

Компиляция

После того как будут определены файлы, в которых находится справочная информация (сформирован раздел `[FILES]`) и подготовлена информация для

формирования вкладки **Содержание** (создан файл контекста), можно выполнить компиляцию — преобразовать исходную справочную информацию в файл справочной системы (chm-файл).

Исходной информацией для HTML Help компилятора являются:

- файл проекта (hhp-файл);
- файл контекста (hhc);
- файлы справочной информации (htm-файлы);
- файлы иллюстраций (gif- и jpg-файлы).

Результатом компиляции является файл справочной системы (chm-файл).

Чтобы выполнить компиляцию, надо в меню **File** выбрать команду **Compile,** в появившемся диалоговом окне **Create a compiled file** (рис. 8.29) установить переключатель **Automatically display compiled help file when done** (после компиляции показать созданный файл справки) и щелкнуть на кнопке **Compile.** В результате этого будет создан файл справки и на экране появится окно справочной системы, в котором будет выведена информация главного раздела.

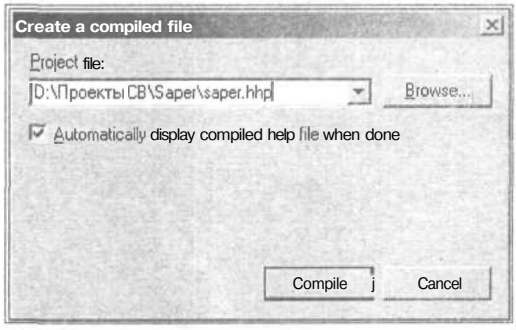


Рис. 8.29. Диалоговое окно **Create a compiled file**

Вывод справочной информации

Вывести справочную информацию, которая находится в chm-файле, можно несколькими способами. Наиболее просто это сделать при помощи утилиты hh.exe, являющейся составной частью Windows. Вызвать утилиту отображения справочной информации и передать ей в качестве параметра имя файла справочной системы можно при помощи функции `WinExec`. У функции `WinExec` два параметра. Первый — имя выполняемого файла программы, которую надо запустить, и командная строка. Второй параметр определяет способ отображения окна запускаемой программы. Окно запускаемой программы может быть развернуто на весть экран (`SW_MAXIMIZE`), запущенная

программа может работать в свернутом окне (SW_MINIMIZE) или окно программы может иметь размер и положение такие, какими они были в предыдущем сеансе работы (SW_RESTORE). Ниже в качестве примера приведена функция обработки события click на кнопке **Справка.** Функция обеспечивает вывод справочной информации, которая находится в файле saper.chm.

```
// Щелчок на кнопке Справка
void _fastcall TForm1::Button2Click(TObject *Sender)
{
  WinExec("hh saper.chm",SW_RESTORE);
}
```

ГЛАВА 9

Создание установочного диска

Современные программы распространяются на компакт-дисках. Процесс установки программы, который, как правило, предполагает не только создание каталога и перенос в него выполняемых файлов и файлов данных с промежуточного носителя, но и настройку системы, для многих пользователей является довольно трудной задачей. Поэтому установку прикладной программы на компьютер пользователя обычно возлагают на специальную программу, которая находится на том же диске, что и файлы программы, которую надо установить. Таким образом, разработчик прикладной программы помимо основной задачи должен создать программу установки -- инсталляционную программу.

Инсталляционная программа может быть создана точно так же, как и любая другая программа. Задачи, решаемые во время инсталляции, являются типовыми. Поэтому существуют инструментальные средства, используя которые можно быстро создать инсталляционную программу, точнее, установочный диск, не написав ни одной строчки кода.

Программа InstallShield Express

Одним из популярных инструментов создания инсталляционных программ является пакет InstallShield Express. Borland настоятельно рекомендует использовать именно эту программу, поэтому она есть на установочном диске C++ Builder.

Процесс создания инсталляционного диска (CD-ROM) при помощи InstallShield Express рассмотрим на примере.

Пусть нужно создать инсталляционную дискету для программы "Сапер". Перед тем как непосредственно приступить к созданию установочной программы в InstallShield Express, нужно выполнить подготовительную работу — составить список файлов, которые должны быть установлены на ком-

пьютер пользователя; используя редактор текста, подготовить rtf-файлы лицензионного соглашения (EULA — End User Licensia Agreement) и краткой справки (readme-файл). Список файлов программы "Сапер", которые должны быть перенесены на компьютер пользователя, приведен в табл. 9.1.

Таблица 9.1. *Файлы программы "Сапер", которые нужно установить на компьютер пользователя*

Файл	Назначение	Куда устанавливать
Saper.exe	Программа	Program Files\Saper
Saper.chm	Файл справочной информации	Program Files\Saper
Readme.rtf	Краткая справка о программе	Program Files\Saper
Eula.rtf	Лицензионное соглашение	Program Files\Saper

Новый проект

После того как будет составлен список файлов, нужно запустить InstallShield Express, из меню **File** выбрать команду New и в поле **Project Name and Location** ввести имя файла проекта (рис. 9.1).

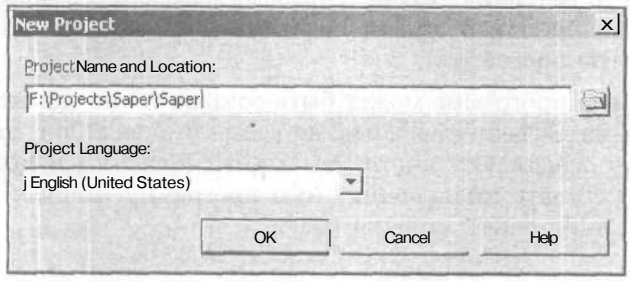


Рис. 9.1. Начало работы над новым проектом

После щелчка на кнопке **OK** открывается окно проекта создания инсталляционной программы (рис. 9.2). В левой части окна перечислены этапы процесса создания и команды, при помощи которых задаются параметры создаваемой инсталляционной программы.

Команды настройки объединены в группы, название и последовательность которых отражает суть процесса создания инсталляционной программы. Заголовки групп пронумерованы. Настройка программы установки выполняется путем последовательного выбора команд. В результате выбора команды в правой части главного окна появляется список параметров. Команды, которые были выполнены, помечаются галочками.

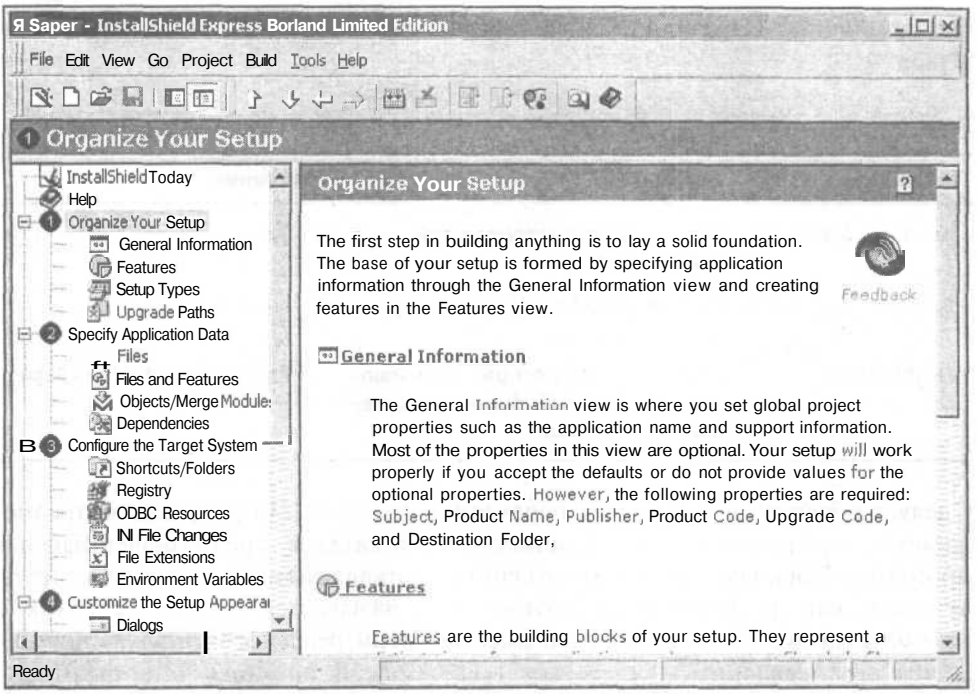


Рис. *9.2.* В левой части окна проекта перечислены этапы и команды процесса создания инсталляционной программы

Структура

Команды группы **Organize Your Setup** (рис. 9.3) позволяют задать структуру программы установки.

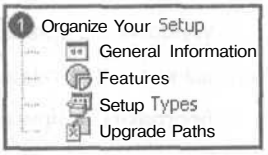


Рис. 9.3. Команды группы Organize Your Setup

В результате выбора команды **General Information** в правой части окна раскрывается список, в который нужно ввести информацию об устанавливаемой программе.

Значения большинства параметров, за исключением тех, которые идентифицируют устанавливаемую программу и ее разработчика, можно оставить

без изменения. Параметры, значения которых нужно изменить, приведены в табл. 9.2.

Таблица 9.2. Параметры команды ***General Information***

Параметр	Определяет	Значение
Product Name	Название устанавливаемой программы	Saper
Product Version	Версия устанавливаемой программы	1.01.0001
INSTALLDIR	Каталог компьютера пользователя, в который будет установлена программа	[ProgramFilesFolder]Saper

Следует обратить внимание на параметр INSTALLDIR. По умолчанию предполагается, что программа будет установлена в каталог, предназначенный для программ. Поскольку во время создания инсталляционной программы нельзя знать, как на компьютере пользователя называется каталог программ и на каком диске он находится, то вместо имени реального каталога используется его псевдоним — [ProgramFilesFolder]. В процессе установки программы на компьютер пользователя инсталляционная программа получит из реестра Windows имя каталога программ и заменит псевдоним на это имя.

Другие псевдонимы, которые используются в программе InstallShield Express, приведены в табл. 9.3

Таблица 9.3. Некоторые псевдонимы каталогов Windows

Псевдоним	Каталог
[WindowsVolume]	Корневой каталог диска, на котором находится Windows
[WindowsFolder]	Каталог Windows, например C:\Winnt
[SystemFolder]	Системный каталог Windows, например C:\Winnt\System32
[ProgramFilesFolder]	Каталог программ, например C:\Program Files
[PersonalFolder]	Папка **Мои документы** на рабочем столе (расположение папки зависит от версии ОС и способа входа в систему)

Очевидно, что *возможности* инсталлированной программы определяются составом установленных компонентов. Например, если установлены файлы справочной системы, то пользователю в процессе работы с программой доступна справочная информация. Команда **Features** (возможности) позволяет создать (определить) группы компонентов, которые определяют *воз-*

можности программы и которые могут устанавливаться по отдельности. Разделение компонентов на группы позволяет организовать многовариантную, в том числе и определяемую пользователем, установку программы.

В простейшем случае группа **Features** состоит из одного элемента **Always Install.** Чтобы добавить элемент в группу **Features,** нужно щелкнуть правой кнопкой мыши на слове **Features,** из появившегося контекстного меню выбрать команду New **Feature Ins** и ввести имя новой группы, например Help Files and samples. После этого в поле **Description** нужно ввести краткую характеристику элемента, а в поле **Comments** — комментарий (рис. 9.4).

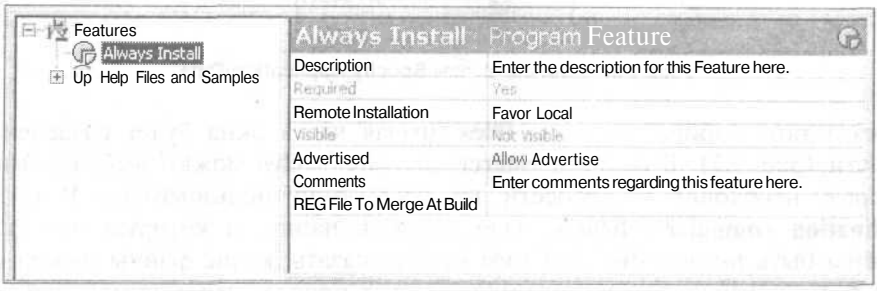


Рис. 9.4. Несколько элементов в группе **Features** обеспечивают возможность многовариантной установки

Команда **Setup Types** позволяет задать, будет ли пользователю во время установки программы предоставлена возможность выбрать (в диалоговом окне **Setup Type)** вариант установки. Установка может быть обычной **(Typical),** минимальной **(Minimal)** или выборочной **(Custom).** Если устанавливаемая программа сложная, т. е. состоит из нескольких независимых компонентов, то эта возможность обычно предоставляется.

Для программы "Сапер" предполагается только один вариант установки -- **Typical.** Поэтому флажки **Minimal и Custom** нужно сбросить (рис. 9.5).

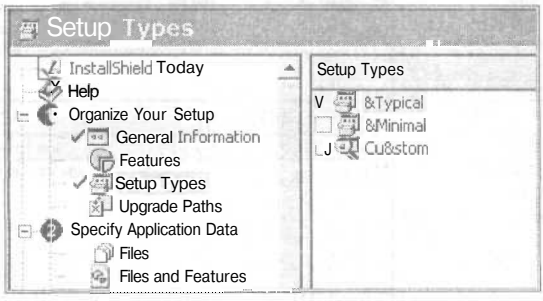


Рис. 9.5. Команда **Setup Types** позволяет задать возможные варианты установки программы

Выбор устанавливаемых компонентов

Команды группы **Specify Application Data** (рис. 9.6) позволяют определить компоненты программы, которые должны быть установлены на компьютер пользователя. Если в проекте определены несколько групп компонентов (см. команду **Features),** то нужно определить компоненты для каждой группы.

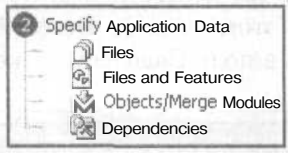


Рис. 9.6. Команды группы **Specify Application Data**

В результате выбора команды **Files** правая часть окна будет разделена на области (рис. 9.7). В области **Source computer's files** можно выбрать файлы, которые необходимо перенести на компьютер пользователя. В области **Destination computer's folders** надо выбрать папку, в которую эти файлы должны быть помещены. Для того чтобы указать, какие файлы нужно установить на компьютер пользователя, следует просто "перетащить" требуемые файлы из области **Source computer's files** в область **Destination computer's files.** Если в группе **Features** несколько элементов, то надо определить файлы для каждого элемента.

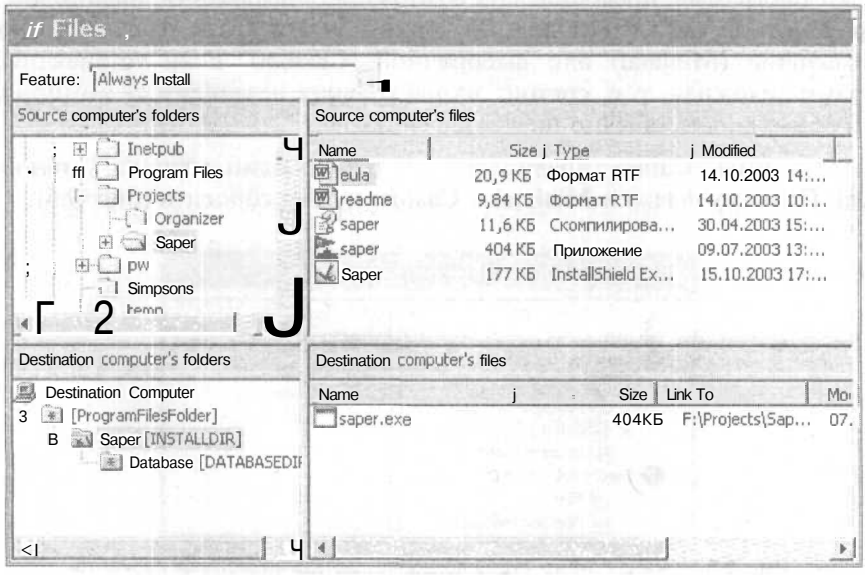


Рис. 9.7. Выбор файлов, которые нужно перенести на компьютер пользователя

Команда **Object/Merge Modules** позволяет задать, какие объекты, например динамические библиотеки или пакеты компонентов, должны быть помещены на компьютер пользователя и, следовательно, на установочную дискету. Объекты, которые нужно поместить на установочную дискету, выбираются в списке **InstallShield Objects/Merge Modules** (рис. 9.8).

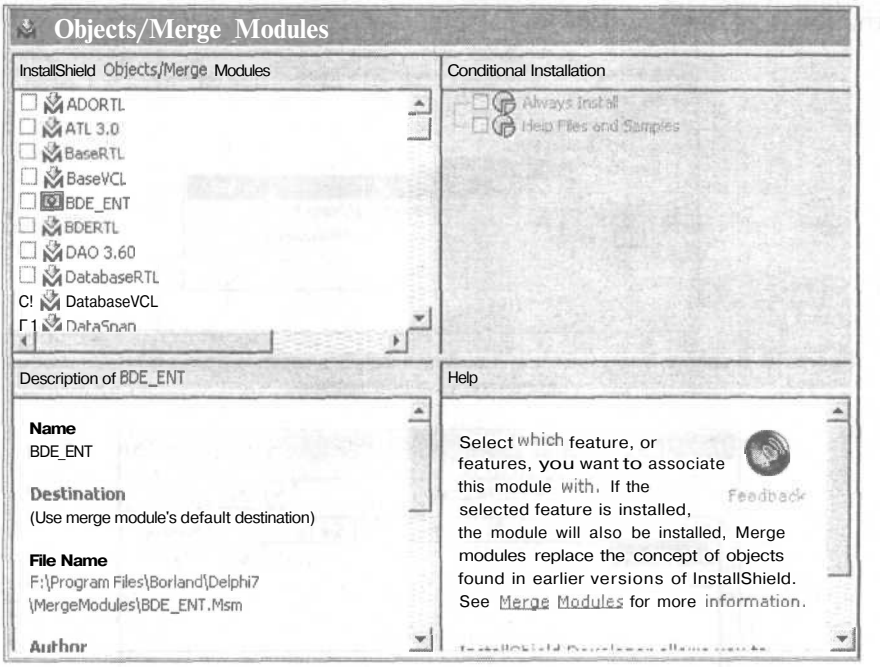


Рис. 9.8. Выбор объектов, которые должны быть установлены на компьютер пользователя

Конфигурирование системы пользователя

Команды группы **Configure the Target System** (рис. 9.9) позволяют задать, какие изменения нужно внести в систему пользователя, чтобы настроить систему на работу с устанавливаемой программой.

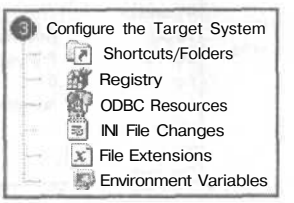


Рис. 9.9. Команды группы **Configure the Target System**

Команда **Shortcuts/Folders** позволяет указать, куда нужно поместить ярлык, обеспечивающий запуск устанавливаемой программы. В результате выбора этой команды в правой части окна открывается иерархический список, в котором перечислены меню и папки, куда можно поместить ярлык программы. В этом списке нужно выбрать меню, в которое должен быть помещен ярлык, щелкнуть правой кнопкой мыши и в появившемся списке выбрать команду New **Shortcut** (рис. 9.10).

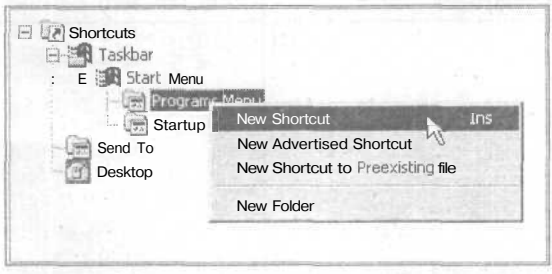


Рис. 9.10. В списке **Shortcuts** нужно выбрать меню, в которое должен быть помещен ярлык запуска программы

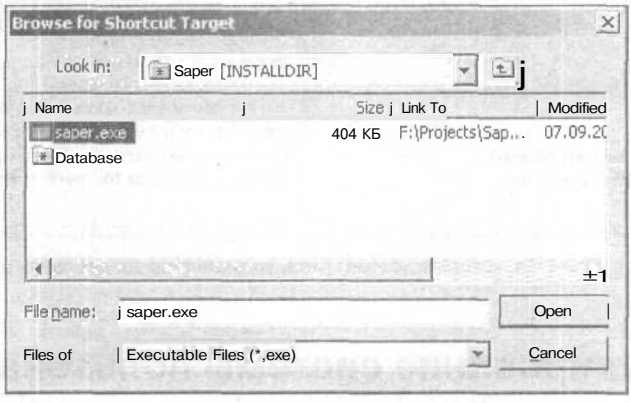


Рис. 9.11. Выбор файла, для которого создается ярлык

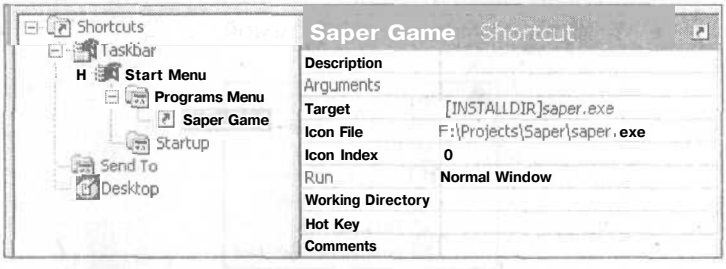


Рис. 9.12. Ярлык создан, теперь можно выполнить его настройку

Затем, в диалоговом окне **Browse for Shortcut Target,** нужно выбрать файл программы (рис. 9.11), щелкнуть на кнопке **Open** и ввести имя ярлыка. После этого можно выполнить окончательную настройку ярлыка, например, в поле **Arguments** ввести параметры командной строки, а в поле **Working Directory** — рабочий каталог (рис. 9.12).

Настройка диалогов

Для взаимодействия с пользователем программа установки использует стандартные диалоговые окна. Разрабатывая программу инсталляции, программист может задать, какие диалоги увидит пользователь в процессе инсталляции программы.

Чтобы задать диалоговые окна, которые будут появляться на экране монитора во время работы инсталляционной программы, надо в группе **Customize the Setup Appearance** (рис. 9.13) выбрать команду **Dialogs** и в открывшемся списке **Dialogs** (рис. 9.14) отметить диалоги, которые нужно включить в программу установки.

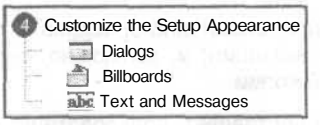


Рис. 9.13. Команды группы Customize the Setup Appearance

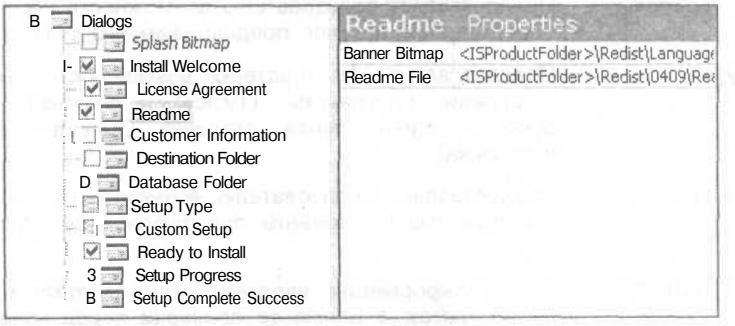


Рис. 9.14. В списке Dialogs нужно отметить диалоги, которые должны появиться в процессе установки программы на компьютер пользователя

В таблице **Properties** (справа от списка диалогов) перечислены свойства выбранного диалога. Программист может изменить значение этих свойств и, тем самым, выполнить настройку диалога. Например, для диалога **Readme** нужно задать имя файла (свойство Readme File), в котором находится краткая справка об устанавливаемой программе.

Для большинства диалогов можно определить баннер (свойство Banne Bitmap) — иллюстрацию, которая отображается в верхней части окна диало га. Формат файла баннера — BMP, размер — 499x58 пикселов.

В табл. 9.4 перечислены диалоговые окна, которые могут появиться во вре мя работы инсталляционной программы.

Таблица 9.4. *Диалоговые окна процесса установки*

Диалоговое окно	Назначение
Splash Bitmap	Вывод иллюстрации, которая может служить в качестве информации об устанавливаемой программе. Размер иллюстрации — 465x281 пиксел, формат — BMP
Install Welcome	Вывод информационного сообщения на фоне иллюстрации (размер 499×312 пикселов)
License Agreement	Вывод находящегося в rft-файле лицензионного сообщения. Позволяет прервать процесс установки программы в случае несогласия пользователя с предлагаемыми условиями
Readme	Вывод краткой информации об устанавливаемой программе
Customer Information	Запрашивает информацию о пользователе (имя, название организации) и, возможно, серийный номер устанавливаемой копии
Destination Folder	Предоставляет пользователю возможность изменить предопределенный каталог, *в* который устанавливается программа
Database Folder	Предоставляет пользователю возможность изменить предопределенный каталог, предназначенный для баз данных
Setup Type	Предоставляет пользователю возможность выбрать тип установки программы (Typical— обычная установка, Minimal — минимальная установка, Custom — выборочная установка)
Custom Setup	Предоставляет пользователю возможность выбрать устанавливаемые компоненты при выборочной (Custom) установке
Ready to Install	Вывод информации, введенной пользователем на предыдущих шагах, с целью ее проверки перед началом непосредственной установки программы
Setup Progress	Показывает процент выполненной работы во время установки программы
Setup Complete Success	Информирует пользователя о завершении процесса установки. Позволяет задать программу, которая должна быть запущена после завершения установки (как правило, это сама установленная программа), а также возможность вывода содержимого Readme-файла

Для того чтобы диалоговое окно появлялось во время работы инсталляционной программы, необходимо установить флажок, расположенный слева от названия диалогового окна. Для окон **License Agreement и Readme** нужно задать имена rtf-файлов, в которых находится соответствующая информация.

В простейшем случае профамма инсталляции может офаничиться выводом следующих диалогов:

- Readme;
- Destination Folder;
- Ready to Install;
- Setup Progress;
- Setup Complete Success.

Системные требования

Если устанавливаемая профамма предъявляет определенные требования к ресурсам системы, то, используя команды группы **Define Setup Requirements and Actions** (рис. 9.15), эти требования можно задать.

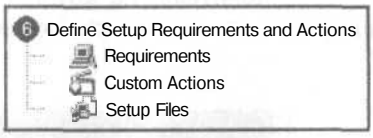


Рис. 9.15. Команды группы **Define Setup Requirements and Actions**

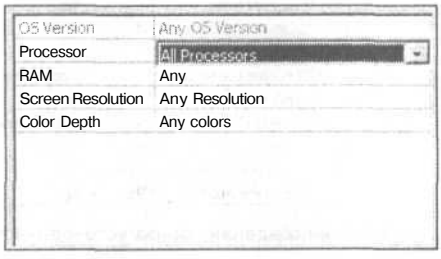


Рис. 9.16. Параметры, характеризующие систему

В результате выбора команды **Requirements** на экране появляется таблица (рис. 9.16), в которую надо ввести значения параметров, характеризующих систему: версию операционной системы (OS Version), тип процессора (Processor), объем оперативной памяти (RAM), разрешение экрана (Screen Resolution) и цветовую палитру (Color Depth). Значения характеристик за-

даются путем выбора из раскрывающегося списка, значок которого появляется в результате щелчка в поле значения параметра.

Если программа не предъявляет особых требований к конфигурации системы, то команды группы **Define Setup Requirements and Actions** можно пропустить.

Создание образа установочной дискеты

Команды группы **Prepare for Release** (рис. 9.17) позволяют создать образ установочной дискеты (CD-ROM) и проверить, как работает программа установки.

Рис. 9.17. Команды группы **Prepare for Release**

Для того чтобы активизировать процесс создания образа установочной дискеты (CD-ROM), нужно выбрать команду **Build Your Release,** щелкнуть правой кнопкой мыши на значке носителя, на который предполагается поместить программу установки, и из появившегося контекстного меню выбрать команду **Build** (рис. 9.18).

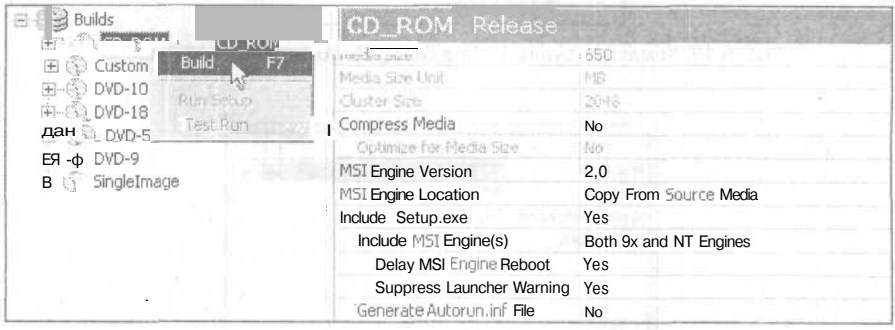


Рис. 9.18. Активизация создания образа установочного CD-ROM

В результате этих действий на диске компьютера в папке проекта будет создан образ установочного диска. Если в качестве носителя выбран CD-ROM, то образ будет помещен в подкаталог \Express\Cd_rom\DiskImages\Disk1.

Можно, не завершая работу с InstallShield Express, проверить, как функционирует программа установки. Для этого надо щелкнуть на одной из командных кнопок **Run** или **Test** (рис. 9.19). Команда Run устанавливает

программу, для которой создана программа установки, на компьютер разработчика. Команда **Test** только имитирует установку, что позволяет проверить работоспособность интерфейса.

Рис. 9.19. Используя команды **Run** и **Test** можно проверить, как работает программа установки

После того как программа установки будет проверена, можно создать реальный установочный диск. Для этого надо просто скопировать (записать) содержимое каталога \Express\Cd_rom\DiskImages\Disk1на CD-ROM.

ГЛАВА 10

Примеры программ

Система проверки знаний

Тестирование широко применяется для оценки уровня знаний в учебных заведениях, при приеме на работу, для оценки квалификации персонала учреждений, т. е. практически во всех сферах деятельности человека. Испытуемому предлагается ряд вопросов (тест), на которые он должен ответить.

Обычно к каждому вопросу дается несколько вариантов ответа, из которых надо выбрать правильный. Каждому варианту ответа соответствует некоторая оценка. Суммированием оценок за ответы получается общий балл, на основе которого делается вывод об уровне подготовленности испытуемого.

Рассмотрим программу, которая позволяет автоматизировать процесс тестирования.

Требования к программе

В результате анализа используемых на практике методик тестирования были сформулированы следующие требования к программе:

- О программа должна обеспечить работу с тестом произвольной длины, т. е. не должно быть ограничения на количество вопросов в тесте;
- О вопрос может сопровождаться иллюстрацией;
- ❐ для каждого вопроса может быть до четырех возможных вариантов ответа со своей оценкой в баллах;
- П результат тестирования должен быть отнесен к одному из четырех уровней, например: "отлично", "хорошо", "удовлетворительно" или "плохо";
- ❐ вопросы теста должны находиться в текстовом файле;
- П в программе должна быть заблокирована возможность возврата к предыдущему вопросу. Если вопрос предложен, то на него должен быть дан ответ.

На рис. 10.1 приведен пример окна программы тестирования во время ее работы.

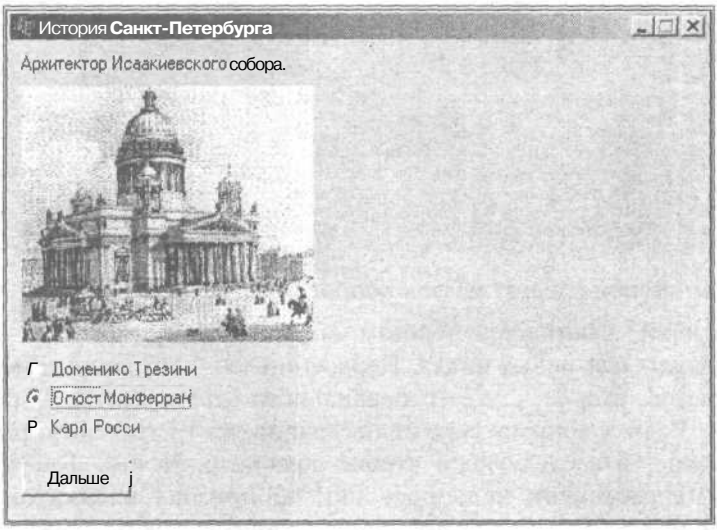


Рис. 10.1. Диалоговое окно программы тестирования

Файл теста

Тест представляет собой последовательность вопросов, на которые испытуемый должен ответить путем выбора правильного ответа из нескольких предложенных вариантов.

Файл теста состоит из трех разделов:

- раздел заголовка;
- раздел оценок;
- раздел вопросов.

Заголовок содержит название теста и общую информацию о тесте — например, о его назначении. Заголовок состоит из двух абзацев: первый абзац - название теста, второй — вводная информация.

Вот пример заголовка:

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и архитектурных сооружениях Санкт-Петербурга. Вы должны из предложенных нескольких вариантов ответа выбрать правильный.
```

За заголовком следует раздел оценок, в котором указывается количество баллов, необходимое для достижения уровня, и сообщение, информирую-

шее испытуемого о достижении уровня. В простейшем случае сообщение — это оценка. Для каждого уровня надо указать балл (количество правильных ответов) и в следующей строке — сообщение. Вот пример раздела оценок:

```
100
Отлично
85
Хорошо
60
Удовлетворительно
50
Плохо
```

За разделом оценок следует раздел вопросов теста.

Каждый вопрос начинается текстом вопроса, за которым (в следующей строке) следуют три целых числа. Первое число — это количество альтернативных ответов, второе — номер правильного ответа, третье — признак иллюстрации. Если к вопросу есть иллюстрация, то третье число должно быть равно единице, и в следующей строке должно быть имя файла иллюстрации. Если иллюстрации к вопросу нет, то признак иллюстрации должен быть равен нулю. Далее следуют альтернативные ответы, каждый из которых должен представлять собой один абзац текста.

Вот пример вопроса:

```
Архитектор Зимнего дворца
3 2 1
herm.bmp
Бартоломео
Карл Росси
Огюст Монферран
```

В приведенном примере к вопросу даны три варианта ответа, правильным является второй ответ (архитектор Зимнего дворца — Карл Росси). К вопросу есть иллюстрация (третье число во второй строке — единица), которая находится в файле herm.bmp.

Ниже в качестве примера приведен текст файла вопросов для контроля знания истории памятников и архитектурных сооружений Санкт-Петербурга.

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и архитектурных
сооружениях Санкт-Петербурга. Вы должны из предложенных нескольких вариантов ответа
выбрать правильный.
7
Вы прекрасно знаете историю Санкт-Петербурга!
6
Вы много знаете о Санкт-Петербурге, но на некоторые вопросы ответили неверно.
```

```
5
Вы недостаточно хорошо знаете историю Санкт-Петербурга?
4
Вы, вероятно, только начали знакомиться с историей Санкт-Петербурга?
Архитектор Исаакиевского собора:
3 2 1
isaak.bmp
Доменико Трезини
Огюст Монферран
Карл Росси
Александровская колонна воздвигнута в 1836 году по проекту Огюста Монферрана
как памятник, посвященный:
2 1 0
деяниям императора Александра I
подвигу русского народа в Отечественной войне 1812 года
Архитектор Зимнего дворца
3 2 1
herm.bmp
Бартоломео Растрелли
Карл Росси
Огюст Монферран
Михайловский (Инженерный) замок — жемчужина архитектуры Петербурга — построен
по проекту:
3 1 0
Андрея Никифоровича Воронихина
Ивана Егоровича Старова
Василия Ивановича Баженова
Остров, на котором находится Ботанический сад, основанный императором Петром I,
называется:
3 3 1
bot.bmp
Заячий
Медицинский
Аптекарский
Невский проспект получил свое название:
3 2 0
по имени реки, на которой стоит Санкт-Петербург
по имени близко расположенного монастыря, Александро-Невской лавры
в память о знаменитом полководце — Александре Невском
Скульптура знаменитого памятника Петру I выполнена:
2 1 0
Фальконе
Клодтом
```

Файл теста может быть подготовлен в текстовом редакторе Notepad или в Microsoft Word. В случае использования Microsoft Word при сохранении текста следует указать, что надо сохранить только текст. Для этого в диалоговом окне **Сохранить** в списке **Тип файла** следует выбрать **Только текст** (*.txt).

Форма приложения

На рис. 10.2 приведен вид формы программы тестирования.

Поле `Label1` предназначено для вывода начальной информации, вопроса и результатов тестирования. Компонет `Image1` предназначен для вывода иллюстрации, сопровождающей вопрос. Кнопка `Button1` используется для подтверждения выбора ответа и перехода к следующему вопросу.

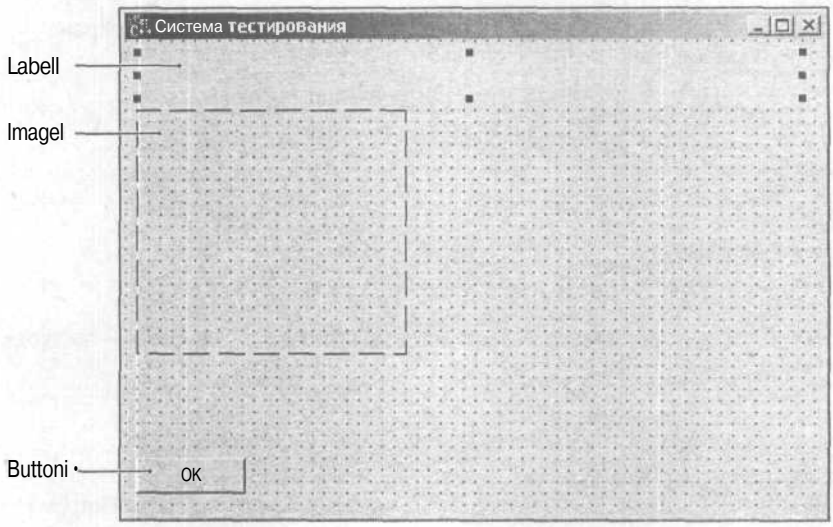


Рис. 10.2. Форма программы тестирования

Нетрудно заметить, что в форме нет радиокнопок - компонентов RadioButton, обеспечивающих вывод альтернативных ответов и прием ответа испытуемого. В рассматриваемой программе компоненты RadioButton будут созданы *динамически,* во время работы программы.

В табл. 10.1 и 10.2 приведены значения свойств формы и компонента Label1. Значения остальных свойств этих и других компонентов можно оставить без изменений.

Таблица 10.1. *Значения свойств формы*

Свойство	Значение	Пояснение
`BorderIcons.biSystemMenu`	true	Есть кнопка системного меню
`BorderIcons.biMinimize`	false	Нет кнопки **Свернуть окно**
`BorderIcons.biMaximize`	false	Нет кнопки **Развернуть окно**

Таблица 10.1 *(окончание)*

Свойство	Значение	Пояснение
BorderStyle	bsSingle	Тонкая граница окна, нельзя изменить размер окна

Таблица 10.2. *Значения свойств компонента Labell*

Свойство	Значение	Пояснение
AutoSize	false	Запрет изменения размера поля в соответствии с его содержимым
Wordwrap	true	Разрешить перенос слов в следующую строку поля

Следует обратить внимание, что несмотря на то, что свойства `BorderIcons.biMinimize` И `BorderIcons.biMaximize` имеют значение false, кнопки **Свернуть окно и Развернуть окно** отображены в форме. Реальное воздействие значений этих свойств на вид окна проявляется только во время работы программы. Значение свойства BorderStyle также проявляет себя только во время работы программы.

Отображение иллюстрации

Для отображения иллюстраций используется компонент `Image1`.

Размер и положение компонента image и, следовательно, размер и положение поля, используемого для отображения иллюстрации, наиболее просто задать во время разработки формы. В рассматриваемой программе применяется другой подход.

Очевидно, что размер области формы, которая может быть использована для вывода иллюстрации, зависит от длины (количества слов) вопроса, а также от длины и количества альтернативных ответов. Чем длиннее вопрос и ответы, тем больше места в поле формы они занимают, и тем меньше места остается для иллюстрации.

Размер и положение областей (компонентов), предназначенных для вывода вопроса, альтернативных ответов и иллюстрации, можно задать в процессе создания формы. Однако можно поступить иначе — задать размер и положение областей во время работы программы, после того как из файла будет прочитан очередной вопрос, когда будет получена информация о количестве альтернативных ответов. После того как вопрос прочитан, можно вычислить, сколько места займет текст вопроса и вариантов ответа и сколько места можно выделить для отображения иллюстрации (рис. 10.3).

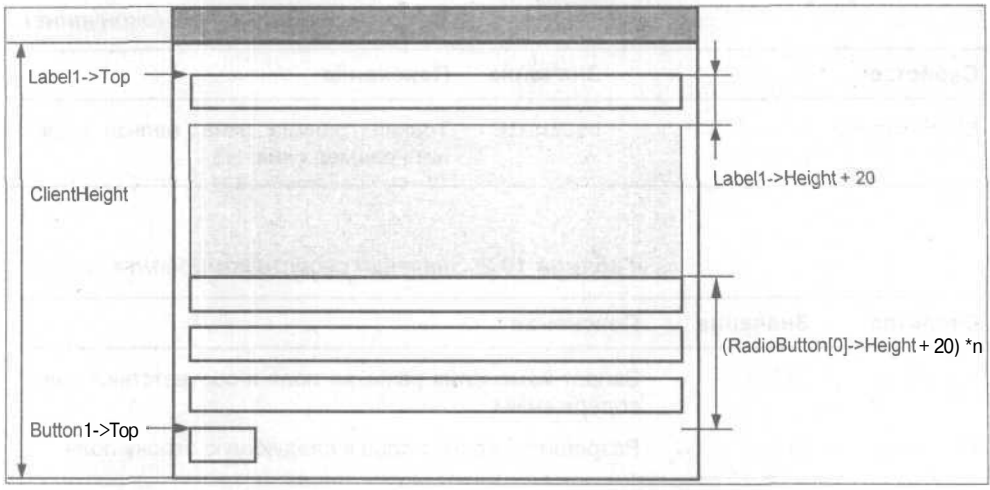


Рис. 10.3. Вычисление размера области вывода иллюстрации

Если реальный размер иллюстрации превышает размер области, выделенной для ее отображения, то необходимо выполнить масштабирование иллюстрации. Для этого надо сначала присвоить максимально возможные значения свойствам width и Height, а затем — присвоить значение true свойству Proportional. Следует обратить внимание, что для того чтобы масштабирование было выполнено без искажения, значение свойства stretc должно быть false.

Доступ к файлу теста

Передать имя файла теста программе тестирования можно через параметр командной строки.

При запуске программы из операционной системы при помощи команды **Пуск | Выполнить** параметры командной строки указывают после имени выполняемого файла программы (рис. 10.4).

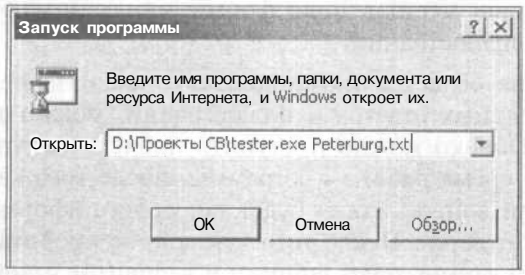


Рис. 10.4. Передача параметра при запуске программы командой **Пуск | Выполнить**

Если запуск программы выполняется при помощи значка, изображающего программу на рабочем столе или в папке, то параметр командной строки задают в окне **Свойства** этого значка. Например, для настройки программы тестирования на работу с файлом теста Peterburg.txt надо раскрыть окно свойств значка (щелкнуть правой кнопкой мыши на значке и из появившегося контекстного меню выбрать команду **Свойства)** и в поле **Объект** (после имени выполняемого файла программы) ввести имя файла теста (Peterburg.txt), заключив его в двойные кавычки (рис. 10.5).

Рис. 10.5. Настройка программы тестирования на работу с файлом Peterburg.txt

Программа может получить информацию о количестве параметров командной строки, обратившись к функции `ParamCount`. Доступ к конкретному параметру обеспечивает функция `ParamStr`, которой в качестве параметра передается номер параметра, значение которого надо получить. Параметры командной строки нумеруются с единицы. Следует обратить внимание, что значением `ParamStr(0)` является полное имя выполняемого файла программы.

Ниже приведен фрагмент программы, который демонстрирует доступ к параметрам командной строки.

```
int n = ParamCount ( ) ;
if ( n < 1)
{
    Label1->Font->Style = TFontStyles ( ) << fsBold;
    Label1->Caption =
         "В командной строке надо указать имя файла теста";
    Button1->Tag = 2;
    return;
}

// открыть файл теста
f = FileOpen(ParamStr(1), fmOpenRead);
```

При запуске программы, использующей параметры командной строки, из среды разработки параметры нужно ввести в поле Parameters диалогового окна **Run Parameters** (рис. 10.6), которое открывается в результате выбора из меню Run команды **Parameters.**

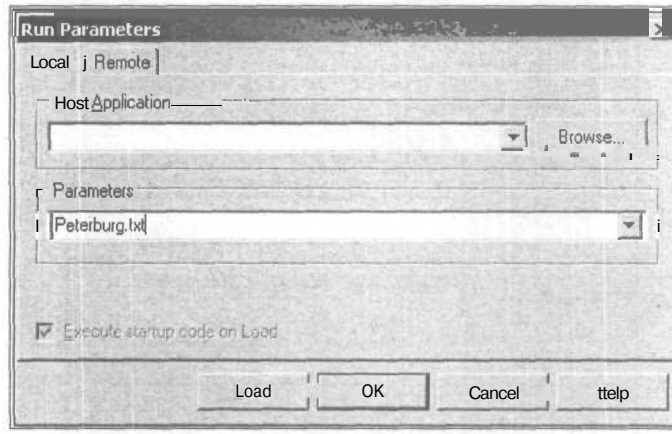


Рис. 10.6. Параметры командной строки надо ввести в поле Parameters

Текст программы

После того как будет создана форма программы, можно приступить к *кодированию* (набору текста). Сначала надо внести дополнения в объявление формы (листинг 10.1) — объявить массив компонентов RadioButton, функцию обработки события click на кнопке выбора ответа и функции, обеспечивающие отображение и удаление вопроса. Следует обратить внимание на то, что объявление массива компонентов RadioButton (указателей на компоненты) только устанавливает факт существования компонентов, сами же компоненты будут созданы в начале работы программы. Делает это конст-

руктор формы. Он же задает функции обработки события click для компонентов массива. Другой важный момент, на который следует обратить внимание, это объявление функций swowvopros и Erasevopros как методов объекта Form1. Это сделано для того, чтобы обеспечить этим функциям прямой доступ к компонентам формы.

Текст модуля главной формы приведен в листинге 10.2.

Листинг 10.1. Программа тестирования (заголовочный файл)

```
#ifndef tester_H
#define tester_H

linclude <Classes.hpp>
#include <Controls.hpp>
#include<StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls .hpp>
#include <Dialogs.hpp>
#include <Graphics.hpp>

// вопрос
struct TVopros {
    AnsiString Vopr;    // вопрос
    AnsiString Img;    // иллюстрация (имя BMP-файла)
    AnsiString Otv[4]; // варианты ответа
    int        nOtv;    // кол-во вариантов ответа
    int        rOtv;    // номер правильного ответа
};

// форма
class TForm1 : public TForm
{
__published: // IDE-managed Components
    TLabel *Label1;   // информационное сообщение, вопрос
    TImage *Image1;    // иллюстрация к вопросу
    TButton *Button1; // кнопка OK / Дальше
    void __fastcall FormActivate (TObject *Sender) ;
    void __fastcall Button1Click (TObject* Sender );

private:
    TRadioButton *RadioButton[4];  // варианты ответа - кнопки выбора
    void __fastcall RadioButtonClick (TObject * Sender ); // щелчок на
                                                  // кнопке выбора ответа
    void __fastcall ShowVopros (TVopros v) ; // выводит вопрос
    void __fastcall EraseVopros(void) ;      // удаляет вопрос
```

```
public:
    __fastcall TForm1 (TComponent* Owner) ;
};

extern PACKAGE TForm1 *Form1;

#endif
```

Листинг 10.2. Программа тестирования

```
/* Универсальная программа тестирования. Тест загружается
   из файла, имя которого должно быть указано в командной
   строке.
   Программа демонстрирует создание и настройку компонентов
   во время работы программы.
*/

#include <vcl.h>
#pragma hdrstop

#include "tester_.h"
#include <stdio.h>     // для доступа к функции sscanf

ttpragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;   // форма

int f;           // дискриптор файла теста
                 // имя файла теста берем из командной строки

int level[4];    // кол-во правильных ответов, необходимое
                 // для достижения уровня
AnsiString mes[4];// сообщение о достижении уровня

TVopros Vopros;  // вопрос
int otv;         // номер выбранного ответа

int right = 0;   // кол-во правильных ответов

// функции, обеспечивающие чтение вопроса из файла теста
int GetInt(int f) ;                        // читает целое
int GetString(int f, AnsiString *st) ;  // читает строку


// конструктор
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
```

```
{
    int i;
    int left = 10;

    // создадим радиокнопки для выбора
    // правильного ответа, но сделаем их невидимыми
    for (i = 0; i < 4; i++)
    {
        // создадим радиокнопку
        RadioButton[i] = new TRadioButton(Form1);
        // установим значения свойств
        RadioButton[i]->Parent = Form1;
        RadioButton[i]->Left = left;
        RadioButton[i]->Width = Form1->ClientWidth - left - 20;
        RadioButton[i]->Visible = false;
        RadioButton[i]->Checked = false;

        // зададим функцию обработки события Click
        RadioButton[i]->OnClick = RadioButtonClick;
    }
}

void __fastcall TForm1::FormActivate(TObject *Sender)
{
    AnsiString st;

    // имя файла теста должно быть указано в командной строке
    int n = ParamCount();
    if ( n < 1)
    {
        Label1->Font->Style = TFontStyles() << fsBold;
        Label1->Caption =
            "В командной строке запуска надо задать имя файла теста";
        Button1->Tag = 2;
        return;
    }

    // открыть файл теста
    f = FileOpen(ParamStr(1), fmOpenRead);
    if ( f == -1)
    {
            Label1->Font->Style = TFontStyles() << fsBold;
            Label1->Caption =
                "Ошибка доступа к файлу теста " + ParamStr(1);
```

```
            Button1->Tag = 2;
            return;
    }

    //  вывести информацию о тесте
    GetString(f, &st); // прочитать название теста
    Form1->Caption = st;

    GetString(f, &st) ; // прочитать вводную информацию
    Label1->Width = Form1->ClientWidth - Label1->Left -20;
    Label1->Caption = st;
    Label1->AutoSize = true;

    // прочитать информацию об уровнях оценки
    for (int i=0; i<4; i++)
    {
        level[i] = GetInt(f);
        GetString(f, &mes[i]);
    }
}

// читает из файла очередной вопрос
bool GetVopros(TVopros *v)
{
    AnsiString st;
    int p;    // если p=1, то к вопросу есть иллюстрация

    if ( GetString(f, &(v->Vopr)) != 0)
    {
        // прочитать кол-во вариантов ответа, номер правильного ответа
        // и признак наличия иллюстрации
        v->nOtv = GetInt(f);
        v->rOtv = GetInt(f);
        p       = GetInt(f);

        if (p) // к вопросу есть иллюстрация
            GetString(f,&(v->Img));
        else v->Img = "";

        // читаем варианты ответа
        for (int i = 0; i < v->nOtv; i++)
        {
            GetString(f,&(v->Otv[i]));
        }
        return true;
    }
```

```
    else return false;
}

// выводит вопрос
void _ fastcall TForm1: : ShowVopros (TVopros v)
{
    int top;
    int i;

    // вопрос
    Label1->Width = ClientWidth - Label1->Left -20;
    Label1->Caption = v.Vopr;
    Label1->AutoSize = true;

    if (v.Img != "") // к вопросу есть иллюстрация
    {
       /* определим высоту области, которую можно
          использовать для вывода иллюстрации */
       int RegHeight =  Button1->Top
           -- (Label1->Top + Label1->Height +10) // область вывода вопроса
           -- (RadioButton[1]->Height + 10) * v.nOtv;

        Image1->Top = Label1->Top + Label1->Height + 10;
        // загрузим картинку и определим ее размер
        Image1->Visible = false;
        Image1->AutoSize = true;
        Image1->Picture->LoadFromFile(v.Img);
        if (Image1->Height > RegHeight) // картинка не помещается
        {
            Image1->AutoSize = false;
            Image1->Height = RegHeight;
            Image1->Proportional = true;
        }
        Image1->Visible = true;
        // положение полей отсчитываем от иллюстрации
         top = Image1->Top + Image1->Height + 10;
    }
    else // положение полей отсчитываем от вопроса
         top = Label1-ХГор + Label1->Height + 10;

    // варианты ответа
    for (i = 0; i < v.nOtv; i++)
    {
        RadioButton[i]->Top = top;
        RadioButton[i]->Caption = v.Otv[i];
```

```
        RadioButton[i]->Visible = true;
        RadioButton[i]->Checked = false;
        top += 20;
    }
}

// щелчок на радиокнопке выбора ответа
void _ fastcall TForm1: :RadioButtonClick (TObject *Sender)
{
    int i =0;
    while {  ! RadioButton[i] ->Checked)
        i++;

    otv = i+1;
    // ответ выбран, сделаем доступной кнопку Дальше
    Button1->Enabled = true;
}

// удаляет вопрос с экрана
void _ fastcall TForm1 : :EraseVopros (void)
{
    Image1->Visible = false;  // скрыть поле вывода иллюстрации

    // скрыть поля выбора ответа
    for (int i = 0; i <4; i++)
    {
        RadioButton[i]->Visible = false;
        RadioButton[i]->Checked = false;
    }
    Button1->Enabled = false; // сделать недоступной кнопку Дальше
}

// щелчок на кнопке OK/Дальше/OK
void _ fastcall TForm1 : :Button1Click (TObject *Sender)
{
    bool ok;  // результат чтения из файла очередного вопроса

    switch (Button1->Tag) {
        case 0: // щелчок на кнопке OK в начале работы программы

                // прочитать и вывести первый вопрос
                GetVopros(&Vopros);
                ShowVopros(Vopros);

                Button1->Caption = "Дальше";
                Button1->Enabled = false;
```

```
            Button1->Tag = 1;
            break;

        case 1 : // щелчок на кнопке Дальше
            if (otv = Vopros.rOtv) // выбран правильный ответ
                right++;
            EraseVopros();
            ok = GetVopros (&Vopros) ;
            if ( ok)
                ShowVopros(Vopros);
            else
            // вопросов больше нет
             {
                FileClose(f);
                // вывести результат
                AnsiString st; // сообщение
                int i; // номер достигнутого уровня

                Form1->Caption = "Результат тестирования";
                st.printf("Правильныхответов: %i\n",right);

                // определим оценку
                i = 0; // предположим, что испытуемый
                       // ответил на все опросы
                while (( right < level[i]) && (i < 3))
                    i++;

                st = st + mes[i];
                Label1->Caption = st;

                Button1->Caption = "OK";
                Button1->Enabled = true;
                Button1->Tag = 2;
            }
            break;

        case 2 : // щелчок на кнопке OK в конце работы программы
            Form1->Close() ; // завершить работу программы
    }
}

// Функция GetString читает строку из файла
// значение функции — количество прочитанных символов
int GetString(int f, AnsiString *st)
{
    unsigned char buf[300]; // строка (буфер)
    unsigned char *p = buf ; // указатель на строку
```

```
    int n;          // кол-во прочитанных байт (значение функции FileRead)
    int len = 0; // длина строки

    n = FileRead(f, p, 1) ;
    while ( n != 0)
    {
        if ( *p == '\r')
        {
            n = FileRead(f, p, 1); // прочитать '\n'
            break;
        }
        len++;
        p++;
        n = FileRead(f, p, 1);
    }

    *p = '\0';
    if ( len !=0)
        st->printf("%s", buf);
    return len;
}

// читает из файла целое число
int GetInt(int f)
{
    char buf[20];    // строка (буфер)
    char *p = buf;  // указатель на строку

    int n;          // кол-во прочитанных байт (значение функции FileRead)
    int a;          // число, прочитанное из файла

    n = FileRead(f, p, 1) ;
    while ( (*p >= '0') && (*p <= '9') && (n > 0))
    {
        p++;
        n = FileRead(f, p, 1);
    }

    if ( *p == '\r')
        n = FileRead(f, p, 1); // прочитать '\n'

    *p = '\0';

    // изображение числа в буфере, преобразуем строку в целое
    sscanf(buf,"%i", &a);
    return a;
}
```

Как было сказано ранее, объявление массива компонентов не создает компоненты, а только устанавливает факт их существования. Создает и настраивает компоненты RadioButton КОНСТруКТОр формы (функция TForm1::TForm1). Непосредственное создание компонента (элемента массива) выполняет оператор .

```
RadioButton[i] = new TRadioButton(Form1)
```

Следующие за этим оператором инструкции обеспечивают настройку компонента. В том числе, они путем присваивания значения свойству onclick задают функцию обработки события click. В рассматриваемой программе для обработки события click на всех компонентах RadioButton используется одна и та же функция, которая путем опроса значения свойства checked фиксирует номер выбранного ответа и делает доступной кнопку **Дальше** (Button1).

После запуска программы и вывода на экран стартовой формы происходит событие OnActivate. Функция обработки этого события проверяет, указан ли в командной строке параметр — имя файла теста. Реализация программы предполагает, что если имя файла теста задано без указания пути доступа к нему, то файл теста и файлы с иллюстрациями находятся в том же каталоге, что и программа тестирования. Если же путь доступа указан, то файлы с иллюстрациями должны находиться в том же каталоге, что и файл теста. Такой подход позволяет сгруппировать все файлы одного теста в одном каталоге.

Если файл теста задан, функция открывает его, читает название теста и вводную информацию и затем выводит их в диалоговое окно, причем название выводится в заголовок, а вводная информация — в поле Label1.

Непосредственное чтение строк из файла выполняет функция Getstring. Значением функции является длина строки. Следует обратить внимание на **ТО, ЧТО фуНКЦИЯ** GetString **Возвращает СТрОКу** AnsiString.

После того как прочитана общая информация о тесте, программа считывает из файла теста информацию об уровнях оценки и фиксирует ее в массивах level и mes. Критерий достижения уровня (количество правильных ответов) считывает функция GetInt.

После вывода информационного сообщения программа ждет, пока пользователь не нажмет кнопку **Ok** (Button1).

Командная кнопка Button1 используется:

- для завершения работы программы, если в командной строке не указан файл теста;
- для активизации процесса тестирования (после вывода информационного сообщения);

- для перехода к следующему вопросу (после выбора варианта ответа);
- для завершения работы программы (после вывода результата тестирования).

Таким образом, реакция программы на нажатие кнопки Button1 зависит от состояния программы. Состояние программы фиксирует свойство Tag кнопки Buttoni.

После вывода информации о тесте значение свойства Tag кнопки Buttoni равно нулю. Поэтому в результате щелчка на кнопке Buttoni выполняется та часть программы, которая обеспечивает вывод первого вопроса и замену находящегося на кнопке текста **OK** на текст **Дальше,** и заменяет значение свойства Tag на единицу.

В процессе тестирования значение свойства Tag кнопки Buttoni равно единице. Поэтому функция обработки события click сравнивает номер выбранного ответа (увеличенный на единицу номер компонента RadioButton) с номером правильного ответа, увеличивает на единицу счетчик правильных ответов (в том случае, если выбран правильный ответ) и активизирует процесс чтения следующего вопроса. Если попытка чтения очередного вопроса завершилась неудачно (это значит, что вопросы исчерпаны), функция выводит результаты тестирования, заменяет текст на командной кнопке на OK и подготавливает операцию завершения работы программы (свойству Tag присваивает значение 2).

Чтение вопроса (вопрос, информация о количестве альтернативных ответов, номер правильного ответа и признак наличия иллюстрации, а также имя файла иллюстрации и альтернативные ответы) из файла теста выполняет **ФУНКЦИЯ** GetVoproS.

Вывод вопроса, иллюстрации и альтернативных ответов выполняет функция showvopros. Сначала функция выводит вопрос — присваивает значение свойству Caption компонента Label1. Затем, если к вопросу есть иллюстрация, функция вычисляет размер области, которую можно выделить для отображения иллюстрации, и загружает иллюстрацию. Если размер иллюстрации больше размера области, функция устанавливает максимально возможный размер компонента Image1 и присваивает значение false свойству AutoSize и true — свойству Proportional, обеспечивая тем самым масштабирование иллюстрации. После этого функция выводит альтернативные ответы. Положение компонентов, обеспечивающих вывод альтернативных ответов, отсчитывается от нижней границы компонента Image1, если к вопросу есть иллюстрация, или компонента Labeii, если иллюстрации нет.

Сразу после вывода вопроса кнопка **Дальше** (Buttoni) недоступна. Сделано это для того, чтобы блокировать возможность перехода к следующему вопросу, если не дан ответ на текущий. Доступной кнопку **Дальше** делает функция обработки события click на одном из компонентов RadioButton.

Кроме того, функция путем сканирования (проверки значения свойства checked компонентов массива RadioButton) определяет, на каком из компонентов массива испытуемый сделал щелчок и, следовательно, какой из вариантов ответа выбран. Окончательная фиксация номера выбранного ответа и сравнение его с номером правильного ответа происходит в результате нажатия кнопки **Дальше.**

Игра "Сапер"

Всем, кто работает с операционной системой Windows, хорошо знакома игра "Сапер". В этом разделе рассматривается аналогичная программа.

Пример окна программы в конце игры (после того, как игрок открыл клетку, в которой находится мина) приведен на рис. 10.7.

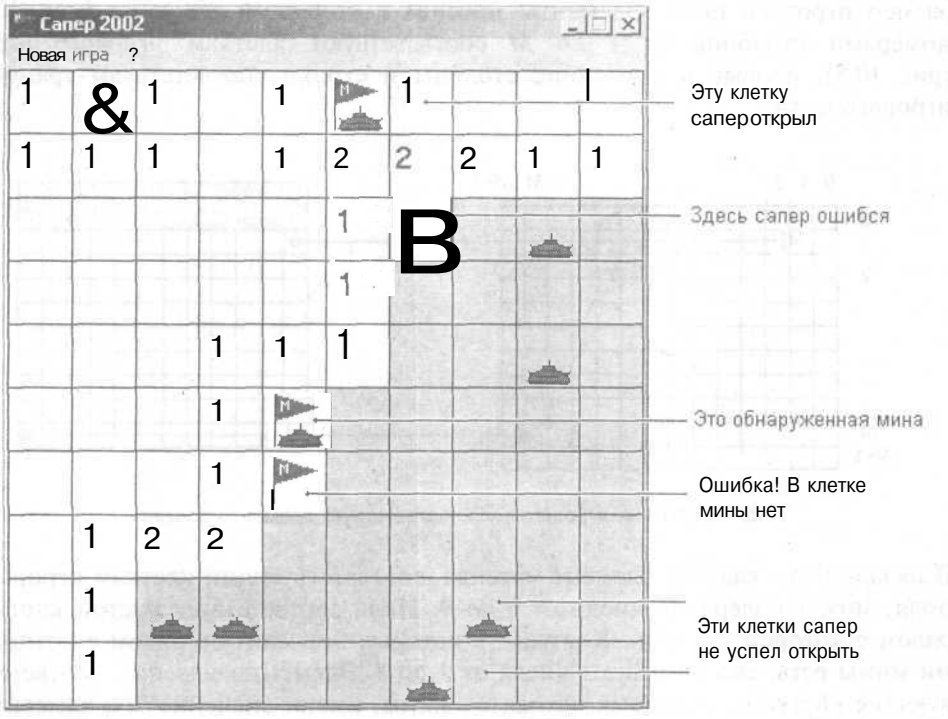


Рис. 10.7. Окно программы "Сапер"

Правила игры и представление данных

Игровое поле состоит из клеток, в каждой из которых может быть мина. Задача игрока — найти все мины и пометить их флажками.

Используя кнопки мыши, игрок может открыть клетку или поставить в нее флажок, указав тем самым, что в клетке находится мина. Клетка открывается щелчком левой кнопки мыши, флажок ставится щелчком правой. Если в клетке, которую открыл игрок, есть мина, то происходит взрыв (сапер ошибся, а он, как известно, ошибается только один раз) и игра заканчивается. Если в клетке мины нет, то в этой клетке появляется число, соответствующее количеству мин, находящихся в соседних клетках. Анализируя информацию о количестве мин в клетках, соседних с уже открытыми, игрок может обнаружить и пометить флажками все мины. Ограничений на количество клеток, помеченных флажками, нет. Однако для завершения игры (для выигрыша) флажки должны быть установлены только в тех клетках, в которых есть мины. Ошибочно установленный флажок можно убрать, щелкнув правой кнопкой мыши в клетке, в которой он находится.

В программе игровое поле представлено массивом *N+2* на *M+2,* где *N*×*M* – размер игрового поля. Элементы массива с номерами строк от 1 до *N* и номерами столбцов от 1 до *M* соответствуют клеткам игрового поля (рис. 10.8); первые и последние столбцы и строки соответствуют границе игрового поля.

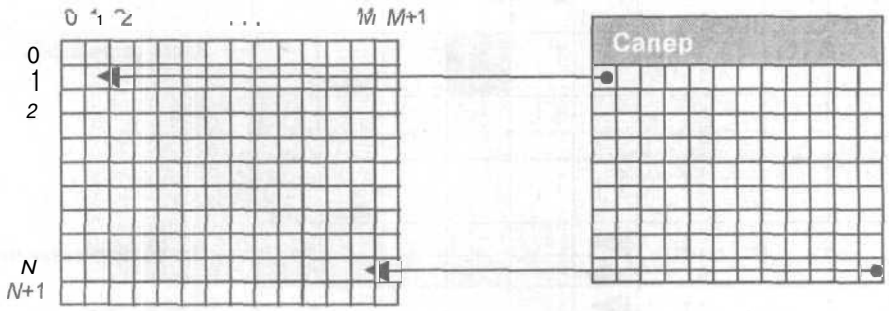


Рис. 10.8. Клетке игрового поля соответствует элемент массива

В начале игры каждый элемент массива, соответствующий клеткам игрового поля, может содержать число от 0 до 9. Ноль соответствует пустой клетке, рядом с которой нет мин. Клеткам, в которых нет мин, но рядом с которыми мины есть, соответствуют числа от 1 до 8. Элементы массива, соответствующие клеткам, в которых находятся мины, имеют значение 9, а элементы массива, соответствующие границе поля, содержат −3.

В качестве примера на рис. 10.9 изображен массив, соответствующий состоянию поля в начале игры.

В процессе игры состояние игрового поля меняется (игрок открывает клетки и ставит флажки, и, соответственно, меняются значения элементов массива. Если игрок поставил в клетку флажок, то значение соответствующего

элемента массива увеличивается на 100. Например, если флажок поставлен правильно, т. е. в клетку, в которой есть мина, значение соответствующего элемента массива станет равным 109. Если флажок поставлен ошибочно (например, в пустую клетку), элемент массива будет содержать число 100. Если игрок открыл клетку, значение элемента массива увеличивается на 200. Такой способ кодирования позволяет сохранить информацию об исходном состоянии клетки.

-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
-3	9	1	0	0	0	0	0	0	0	0	-3
-3	1	1	0	0	0	0	0	0	0	0	-3
-3	1	2	2	1	0	0	0	1	1	1	-3
-3	1	9	9	1	0	0	0	2	9	2	-3
_3	1	2	2	1	0	0	0	2	9	3	-3
-3	0	0	0	0	0	0	0	2	3	9	-3
-3	0	1	2	2	1	0	0	1	9	2	-3
-3	0	2	9	9	1	0	0	1	1	1	-3
-3	0	2	9	3	1	0	0	0	0	0	-3
-3	0	1	1	1	0	0	0	0	0	0	-3
-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3

Рис. 10.9. Массив в начале игры

Форма приложения

Главная (стартовая) форма ифы "Сапер" приведена на рис. 10.10.

Рис. 10.10. Главная форма программы "Сапер"

Следует обратить внимание на то, что размер формы не соответствует размеру игрового поля. Нужный размер формы будет установлен во время работы программы. Делает это функция обработки события OnFormActivate, которая на основе информации о размере игрового поля (количестве клеток по вертикали и горизонтали) и размере клеток устанавливает значения свойств ClientHeight и ClientWidth, определяющие размер клиентской области главного окна программы.

Главное окно программы содержит компонент MainMenu1, который представляет собой главное меню программы. Значок компонента MainMenu находится на вкладке **Standard** (рис. 10.11).

Рис. 10.1.1. Компонент MainMenu

Значок компонента MainMenu можно поместить в любое место формы, т. к. во время работы программы он не виден. Пункты меню появляются в верхней части формы в результате настройки компонента. Для настройки меню используется *редактор меню,* который запускается двойным щелчком левой кнопкой мыши на значке компонента или путем выбора из контекстного меню компонента команды **Menu Designer.** В начале работы над новым меню, сразу после добавления компонента к форме, в окне редактора находится один-единственный прямоугольник — *заготовка* пункта меню. Чтобы превратить эту заготовку в меню, нужно в поле **Caption** окна **Object Inspector** ввести название меню.

Если перед какой-либо буквой в названии меню ввести знак &, то во время работы программы можно будет активизировать этот пункт меню путем нажатия комбинации клавиши <Alt> и клавиши, соответствующей символу, перед которым стоит знак &. В названии меню эта буква будет подчеркнута.

Чтобы добавить в главное меню элемент, нужно в окне редактора меню выбрать последний (пустой) элемент меню и ввести название нового пункта.

Чтобы добавить в меню команду, нужно выбрать тот пункт меню, в который надо добавить команду, переместить указатель активного элемента меню в конец списка команд меню и ввести название команды.

На рис. 10.12 приведено окно редактора меню, в котором находится меню программы "Сапер".

После того как будет сформирована структура меню, нужно, используя окно **Object Inspector,** выполнить настройку элементов меню (выбрать настраиваемый пункт меню можно в окне формы приложения или из списка объектов в верхней части окна **Object Inspector).** Каждый элемент меню (пунк-

ты и команды) — это объект типа TMenuItem. Свойства объектов TMenuItem (табл. 10.3) определяют вид меню во время работы программы.

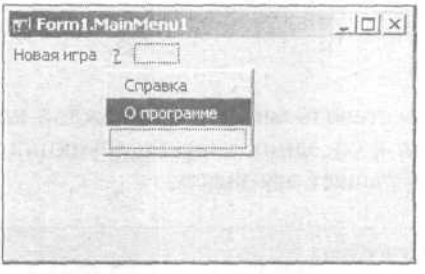


Рис. 10.12. Структура меню программы "Сапер"

Таблица 10.3. *Свойства объекта TMenuitem*

Свойство	Определяет
Name	Имя элемента меню. Используется для доступа к свойствам
Caption	Название меню или команды
Bitmap	Значок, который отображается слева от названия элемента меню
Enabled	Признак доступности элемента меню. Если значение свойства равно false, то элемент меню недоступен, при этом название меню отображается серым цветом

При выборе во время работы программы элемента меню происходит событие Click. Чтобы создать процедуру обработки этого события, нужно в окне формы выбрать пункт меню и щелкнуть левой кнопкой мыши — C++ Builder создаст шаблон процедуры обработки этого события. В качестве примера ниже приведена функция обработки события click, которое возникает в результате выбора из меню ? команды **Справка.** N3 — это имя элемента меню, соответствующего этой команде.

```
// выбор в меню "?" команды Справка
void__fastcall TForm1::N3Click(TObject *Sender)
{
    WinHelp(Form1->Handle,"saper.hlp",HELP_CONTEXT,1);
}
```

Игровое поле

На разных этапах игры игровое поле выглядит по-разному. Вначале поле просто разделено на клетки. Во время игры, в результате щелчка правой кнопкой мыши, в клетке появляется флажок. Щелчок левой кнопкой тоже

меняет вид клетки: клетка меняет цвет и в ней появляется цифра или мина (игра на этом заканчивается). Рассмотрим объекты, свойства и методы, обеспечивающие работу с графикой.

Начало игры

В начале игры надо расставить мины и для каждой клетки поля подсчитать, сколько мин находится в соседних клетках. Функция NewGame (ее текст приведен в листинге 10.3), решает эту задачу.

Листинг 10.3. Функция NewGame

```
// новая игра — генерирует новое поле
void _ fastcall NewGame ()
{
   // Очистим элементы массива, соответствующие отображаемым
   // клеткам, а в неотображаемые, по границе игрового поля,
   // запишем число -3. Уникальное значение клеток границы
   // используется функцией Open для завершения рекурсивного
   // процесса открытия соседних пустых клеток.
   int row, col;
   for (row=0; row <= MR+1; row++)
        for (col=0; col <= MC+1; col++)
               Pole[row][col] = -3;
   for (row=1; row <= MR; row++)
        for (col=1; col <= MC; col++)
               Pole[row][col] = 0;

   // расставим мины
   time_t t;      // используется генератором случайных чисел (ГСЧ)
   srand((unsigned)time(&t)); // инициализация ГСЧ
   int n = 0; // количество мин
   do
   {
       row = rand() % MR +1;
       col = rand() % MC +1;
       if ( Pole[row][col] != 9)
       {
           Pole[row][col] = 9;
          n++;
       }
   }
   while ( n < 10);
```

```
    // вычисление количества мин в соседних клетках
    int k;
    for ( row = 1; row <= MR; row++)
        for ( col = 1; col <= MC; col++)
           if ( Pole[row][col] != 9) {
         k =0;
               if ( Pole[row-1][col-1] == 9) k++;
               if ( Pole[row-1][col]   == 9) k++;
               if ( Pole[row-1][col+1] == 9) k++;
               if ( Pole[row][col-1]    = 9) k++;
               if ( Pole[row][col+1]   == 9) k++;
               if ( Pole[row+1][col-1] == 9) k++;
               if ( Pole[row+1][col]   == 9) k++;
               if ( Pole[row+1][col+1] == 9) k++;
               Pole[row][col] = k;
           }
    status = 0; // начало игры
    nMin = 0;   // нет обнаруженных мин
    nFlag = 0;  // нет флагов
}
```

После того как функция NewGame расставит мины, функция ShowPole (ее текст приведен в листинге 10.4) выводит изображение игрового поля.

Листинг 10.4. Функция ShowPole

```
// показывает поле
void__fastcall TForm1::ShowPole( int status)
{
    for ( int row = 1; row <= MR; row++)
         for ( int col = 1; col <= MC; col++)
              Kletka(row, col, status);
}
```

Функция ShowPole выводит изображение поля последовательно, клетка за клеткой. Вывод изображения отдельной клетки выполняет функция Kletka, ее текст приведен в листинге 10.5. Функция Kletka используется для вывода изображения поля в начале игры, во время игры и в ее конце. В начале игры (значение параметра status равно нулю) функция выводит только контур клетки, во время игры — количество мин в соседних клетках или флажок, а в конце она отображает исходное состояние клетки и действия пользователя. Информацию о фазе игры функция Kletka получает через параметр status.

Листинг 10.5. Функция Kletka

```
// рисует на экране клетку
void _ fastcall TForm1 :: Kletka (int row, int col, int status)
{
     int x = LEFT + (col-1)* W;
     int y = TOP + (row-1)* H;

     if (status == 0) // начало игры
     {
        // клетка — серый квадрат
        Canvas->Brush->Color = clLtGray;
        Canvas->Rectangle(x-1,y-1,x+W,y+H);
        return;
     }

    // во время (status = 1) и в конце (status = 2} игры
    if ( Pole[row][col] < 100)
    {
         // клетка не открыта
         Canvas->Brush->Color = clLtGray;  // не открытые — серые
         Canvas->Rectangle(x-1,y-1,x+W,y+H);
         if (status == 2 && Pole[row][col] == 9)
             Mina ( x, y) ; // игра закончена, показать мину
         return;
    }

    // клетка открыта
    Canvas->Brush->Color = clWhite;      // открытые белые
    Canvas->Rectangle(x-1,y-1,x+W,y+H);
    if  ( Pole[row][col] == 100) return; // клетка пустая

    if ( Pole[row][col] >= 101 && Pole[row][col] <= 108)
    {
             Canvas->Font->Size = 14;
             Canvas->Font->Color = clBlue;
             Canvas->TextOutA(x+3,y+2,IntToStr(Pole[row][col] -100));
             return;
    }

    if ( Pole[row][col] >= 200)
         Flag(x,y);

    if (Pole[row][col] == 109) // на этой мине подорвались!
    {
```

```
        Canvas->Brush->Color = clRed;
        Canvas->Rectangle(x,y,x+W,y+H);
    }
    if (( Pole[row][col] % 10 == 9)  &&  (status == 2))
            Mina( x, y);
}
```

Игра

Во время игры программа воспринимает нажатия кнопок мыши и, в соответствии с правилами игры, открывает клетки или ставит в клетки флажки.

Основную работу выполняет функция обработки события OnMouseDown (ее текст приведен в листинге 10.6). Функция получает координаты точки формы, в которой игрок щелкнул кнопкой мыши, а также информацию о том, какая кнопка была нажата. Сначала функция преобразует координаты точки, в которой игрок нажал кнопку мыши, в координаты клетки игрового поля. Затем она вносит необходимые изменения в массив Pole и, если нажата правая кнопка, вызывает функцию Flag, которая рисует в клетке флажок. Если нажата левая кнопка в клетке, в которой нет мины, то эта клетка открывается, и на экране появляется ее содержимое. Если нажата левая кнопка в клетке, в которой есть мина, то вызывается функция ShowPole, которая показывает все мины, в том числе и те, которые игрок не успел найти.

Листинг 10.6. Обработка события OnMouseDown на поверхности игрового поля

```
// нажатие кнопки мыши на игровом поле
void__fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton
Button,
      TShiftState Shift, int x, int y)
{
    if ( status == 2) return;

    if ( status == 0) status = 1;

    x -= LEFT;
    y -= TOP;
    if (x > 0 && y > 0)
    {
        // преобразуем координаты мыши
        // в индексы клетки поля
        int row = y/H + 1;
        int col = x/W + 1;
```

```
        if (Button == mbLeft)
        {
          if ( Pole[row][col] == 9)
          {
            Pole[row][col] +=100;
            status = 2;  // игра закончена
            ShowPole(status);
          }
          else if ( Pole[row][col] < 9)
              i
                Open(row,col);
                ShowPole(status);
              }
        }
       else if (Button == mbRight)
       {
         nFlag++;
         if ( Pole[row][col] == 9)
           nMin++;
         Pole[row][col] += 200;  // поставили флаг
         if (nMin == NM && nFlag == NM)
         {
           status = 2;  // игра закончена
           ShowPole(status);
         }
         else Kletka(row, col, status);
       }
    }
}
```

Функция Flag (листинг 10.7) рисует флажок. Флажок (рис. 10.13) состоит из четырех примитивов: линии (древко), замкнутого контура (флаг) и ломаной линии (буква "М"). Функция Flag рисует флажок, используя метод базовой точки, т. е. координаты всех точек, определяющих положение элементов рисунка, отсчитываются от базовой точки.

Функция Mina (листинг 10.8) рисует мину. Мина (рис. 10.14) состоит из восьми примитивов: два прямоугольника и сектор образуют корпус мины, остальные элементы рисунка — линии ("усы" и полоски на корпусе).

Обеим функциям в качестве параметров передаются координаты базовой точки рисунка и указатель на объект, на поверхности которого надо рисовать.

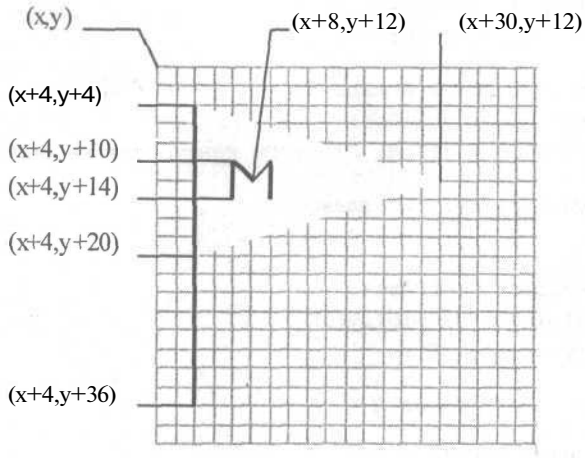


Рис. 10.13. Флажок

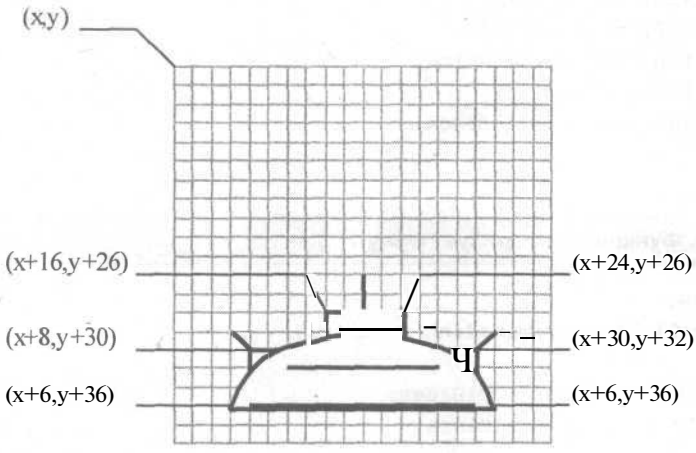


Рис. 10.14. Мина

Листинг 10.7. Функция Flag рисует флажок

```
// рисует флаг
void __fastcall TForm1::Flag( int x, int y)
{
    TPoint p[4]; // координаты флажка и нижней точки древка

    // точки флажка
    p[0].x=x+4;   p[0].y=y+4;
```

```
    p[1].x=x+30;  p[1].y=y+12;
    p[2].x=x+4;   p[2].y=y+20;

    // установим цвет кисти и карандаша
    Canvas->Brush->Color = clRed;
    Canvas->Pen->Color = clRed;  // чтобы контур флажка был красный

    Canvas->Polygon(p, 2);  // флажок

    // древко
    Canvas->Pen->Color = clBlaek;
    Canvas->MoveTo(p[0].x, p[0].y);
    Canvas->LineTo(x+4,y+36);

    TPoint m[5];      // буква М

    m[0].x=x+8; m[0].y=y+14;
    т[1].x=x+8;  т[1].y=y+8;
    m[2].x=x+10; m[2].y=y+10;
    m[3].x=x+12; m[3].y=y+8;
    m[4].x=x+12; m[4].y=y+14;
    Canvas->Pen->Color = clWhite;
    Canvas->Polyline(m,4);
    Canvas->Pen->Color = clBlaek;
}
```

Листинг 10.8. Функция Mina рисует мину

```
// рисует мину
void__fastcall TForm1::Mina(int x, int y)
f
    Canvas->Brush->Color = clGreen;
    Canvas->Pen->Color = clBlaek;

    // корпус
    Canvas->Rectangle(x+16,y+26,x+24,y+30);
    Canvas->Rectangle(x+8,y+30,x+32,y+34);
    Canvas->Pie(x+6,y+28,x+34,y+44,x+34,y+36,x+6,y+36);

    // полоса на корпусе
    Canvas->MoveTo(x+12,y+32); Canvas->LineTo(x+28,y+32);

    // основание
      Canvas->MoveTo(x+8,y+36);  Canvas->LineTo(x+32,y+36);

    // вертикальный "ус"
    Canvas->MoveTo(x+20,y+22); Canvas->LineTo(x+20,y+26);
```

```
    // боковые "усы"
    Canvas->MoveTo(x+8, y+30); Canvas->LineTo(x+6,y+28);
    Canvas->MoveTo(x+32,y+30); Canvas->LineTo(x+34,y+28);
}
```

Справочная информация

В результате выбора в меню ? команды **Справка** или нажатия клавиши <F1> должна появляться справочная информация — правила ифы (рис. 10.15).

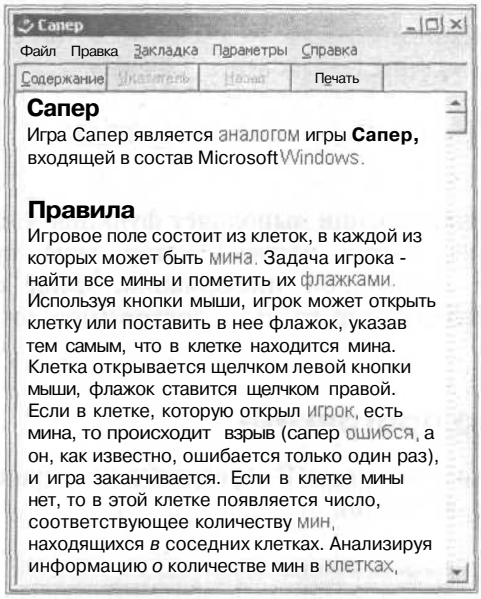


Рис. 10.15. Окно справочной системы программы "Сапер"

Для того чтобы во время работы программы пользователь, нажав клавишу <F1>, мог получить справочную информацию, свойствам `HelpFile` и `HelpContext` главной формы надо присвоить значения, указанные в табл. 10.4.

Таблица 10.4. *Значения свойств главной формы*

Свойство	Значение	Пояснение
`HeipFile`	saper.hlp	Файл справки
`HelpContext`	1	Идентификатор раздела, содержимое которого отображается в результате нажатия <F1>

Для того чтобы справочная информация появилась на экране в результате выбора в меню ? команды **Справка,** надо создать функцию обработки события Onclick для соответствующей команды меню. Процесс создания функции обработки события для команды меню ничем не отличается от процесса создания функции обработки события для элемента управления, например, для командной кнопки: в списке объектов надо выбрать объект типа TMenuItem, для которого создается функция обработки события, а во вкладке **Events** — событие.

Ниже приведена функция обработки события Onclick для команды **Справка** меню ?.

```
// выбор в меню ? команды Справка
void __fastcall TForm1::N3Click(TObject *Sender)
{
    WinHelp(Form1->Handle,"saper.hlp",HELP_CONTEXT,1);
}
```

Вывод справочной информации выполняет функция WinHelp, которой в качестве параметра передается идентификатор окна программы, которая запрашивает вывод справочной информации, файл справки, константу HELP_CONTEXT и идентификатор раздела, содержимое которого должно быть отражено.

Информация о программе

При выборе из меню ? команды **О программе** на экране должно появиться одноименное окно (рис. 10.16).

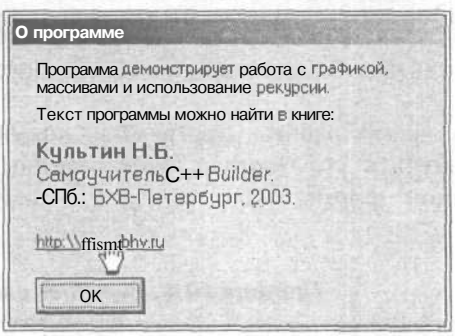


Рис. 10.16. Выбрав ссылку, можно активизировать браузер и перейти на страницу издательства "БХВ-Петербург"

Чтобы программа во время своей работы могла вывести на экран окно, отличное от главного (стартового), нужно добавить в проект форму. Делается это выбором из меню **File** команды New **form**. В результате выполнени

команды New form в проект добавляется новая форма и соответствующий ей модуль.

Если в проекте несколько форм, то для того чтобы получить доступ к нужной форме и, соответственно, к модулю, надо выбрать имя нужной формы в списке диалогового окна View Form (рис. 10.17), которое становится доступным в результате щелчка на командной кнопке View **Form** (рис. 10.18) или нажатия комбинации клавиш <Shift>+<F12>.

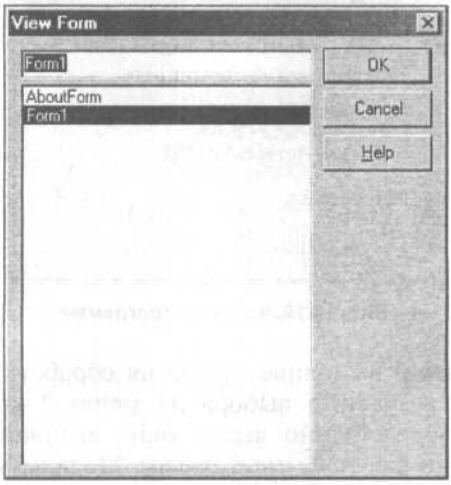


Рис. 10.17. Выбрать нужную форму можно в списке окна **View Form**

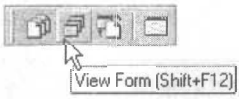


Рис. 10.18. Командная кнопка **View Form**

Вид формы AboutForm после добавления необходимых компонентов приведен на рис. 10.19, значения ее свойств — в табл. 10.5.

Таблица 10.5. *Значения свойств формы* ***О программе***

Свойство	Значение
Name	AboutForm
Caption	О программе
BorderStyle	bsSingle
BorderIcons.biSystemMenu	false

Таблица 10.5 *(окончание)*

Свойство	Значение
BorderIcons.biMininize	false
BorderIcons.biMaximize	false

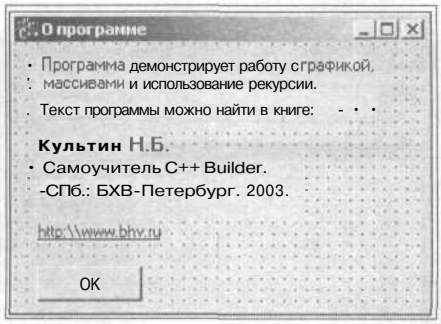


Рис. 10.19. Форма О программе

Вывод окна **О программе** выполняет функция обработки события click, которое происходит в результате выбора из меню ? команды **О программе** (листинг 10.9). Непосредственно вывод окна выполняет метод ShowModal, который выводит окно как *модальный диалог.* Модальный диалог перехватывает все события, адресованные другим окнам приложения, в том числе и главному. Таким образом, пока модальный диалог находится на экране, продолжить работу с приложением, которое вывело модальный диалог, нельзя.

Листинг 10.9. Вывод окна О программе

```
void _ fastcall TForm1 : :N4Click (TObject *Sender)
{
    //  "привяжем" окно О программе к главному окну приложения
    AboutForm->Top = Form1->Top + Form1->Height/2  - AboutForm->Height/2;
    AboutForm->Left = Form1->Left + Form1->Width/2 — AboutForm->Width/2;
    AboutForm->ShowModal();
}
```

Если не предпринимать никаких усилий, то окно **О программе** появится в той точке экрана, в которой находилась форма во время ее разработки. Вместе с тем, можно "привязать" это окно к главному окну программы так, чтобы оно появлялось в центре главного окна. Привязка осуществляется на основании информации о текущем положении главного окна программы (свойства тор и Left) и о размере окна **О программе.**

На поверхности формы **О программе** есть ссылка на сайт издательства "БХВ-Петербург". Предполагается, что в результате щелчка на ссылке в окне браузера будет открыта указанная страница. Для того чтобы это произошло, надо создать функцию обработки события onclick для компонента Label5. Значения свойств компонента Label5 приведены в табл. 10.6, а текст функции обработки события — в листинге 10.10.

Таблица 10.6. *Значения свойств компонента* Label5

Свойство	Значение	Комментарий
Font.Color	clBlue	Цвет — синий
Font.Style.Underline	true	Подчеркивание
Cursor	crHandPoint	При позиционировании указателя мыши на текст указатель принимает форму руки

Для запуска браузера использована функция ShellExecute, которая открывает указанный файл при помощи программы, предназначенной для работы с файлами указанного типа. Так как имя файла в данном случае представляет собой URL-адрес, то будет запущен браузер (тот, который установлен на компьютере пользователя).

Наиболее просто передать URL-адрес в функцию SheilExecute можно как строку-константу, например:

```
ShellExecute(AboutForm->Handle, "open", "http:\\\\www.bhv.ru",
  NULL, NULL, SW_RESTORE);
```

Но лучше URL-адрес брать из поля метки. В функцию ShellExute надо передать указатель на обычную строку, т. е. завершающуюся нулевым символом. Однако свойство caption — это AnsiString. Преобразование строки Ansi **В указатель на** null terminated string выполняет МеТОД c_str() .

Листинг 10.10. Щелчок в поле URL

```
void __fastcall TAboutForm::Label5Click(TObject *Sender)
{
  /* наиболее просто передать в функцию SheilExecute
     строку-константу (URL-адрес) так, как показано ниже:
    ShellExecute (AboutForm->Handle,
                  "open",
                  "http:\\\\www.bhv.ru",
                  NULL, NULL)
```

```
    Лучше URL-адрес брать из поля метки.
    В функцию ShellExecute надо передать указатель на
    null terminated-строку, но свойство Caption - это AnsiString.
    Преобразование Ansi-строки в char* выполняет метод c_str()
*/
        \
    // открыть файл, имя которого находится в поле Label5
    ShellExecute(AboutForm->Handle,"open",Label5->Caption.c_str(),
                 NULL,NULL,SW_RESTORE);
}
```

Окно **О программе** закрывается в результате щелчка на кнопке OK. Функция обработки этого события приведена ниже.

```
void __fastcall TAboutForm::Button1Click(TObject *Sender)
{
    ModalResult = mrOk; // убрать окно О программе
}
```

Текст программы

Полный текст профаммы "Сапер" приведен ниже: в листингах 10.11 и 10.12 – заголовочный файл и модуль главной формы; в листинге 10.13 — модуль формы **О программе.**

Листинг 10.11. Заголовочный файл главной формы (saper_.h)

```
#ifndef saper_H
#define saper_H

#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Menus.hpp>

class TForm1 : public TForm
{
__published:
    TMainMenu *MainMenu1;
    // команды главного меню
    TMenuItem *N1; // Новая игра
    TMenuItem *N2; // команда "?"
```

```
    // команды меню "?"
    TMenuItem *N3; // Справка
    TMenuItem *N4; // О программе

    void __fastcall FormMouseDown (TObject *Sender, TMouseButton Button,
          TShiftState Shift, int X, int Y);
    void __fastcall FormPaint (TObject *Sender);
    void __fastcall FormCreate (TObject *Sender);

    // выбор команды в меню
    void __fastcall N1Click (TObject *Sender); // Новая игра
    void __fastcall N3Click (TObject *Sender); // Справка
    void __fastcall N4Click (TObject *Sender); // О программе

private:
    void __fastcall ShowPole (int status); // отображает поле
    // отображает содержимое клетки
    void __fastcall Kletka (int row, int col, int status);
    void __fastcall Mina (int x, int y); // рисует мину
    void __fastcall Flag( int x, int y); // рисует флаг

public:
    __fastcall TForm1 (TComponent* Owner);
};

extern PACKAGE TForm1 *Form1;

#endif
```

Листинг 10.12. Модуль главной формы (saper_.cpp)

```
/*
    Игра "Сапер". Демонстрирует работу с графикой,
    использование рекурсии, доступ к файлу справочной
    информации.
*/
#include <vcl.h>
ttinclude <stdlib.h> // для доступа к ГСЧ
#include <time.h>
#include <stdio.h>

#pragma hdrstop

#include "saper_.h"
#include "saper_2.cpp"
```

```
ttpragma package(smart_init)
ttpragma resource "*.dfm"

TForm1 *Form1; // главное окно

_ fastcall TForm1: : TForm1 (TComponent*Owner)
        : TForm(Owner)
{
}

#define MR 10 .// кол-во клеток по вертикали
#define MC 10 // кол-во клеток по горизонтали
Idefine NM 10 // кол-во мин

int Pole[MR+2] [MC+2]; // минное поле
                       // 0. .8 — количество мин в соседних клетках
                       // 9 — в клетке мина
                       // 100..109 — клетка открыта
                       // 200..209 — в клетку поставлен флаг

int nMin;   // кол-во найденных мин
int nFlag;  // кол-во поставленных флагов

int status = 0; // 0 — начало игры; 1 — игра; 2 — результат

// смещение игрового поля относительно левого верхнего угла
// поверхности формы
ttdefine LEFT 0   // по X
fdefine TOP  1    // по Y

Idefine W    40   // ширина клетки поля
#define H    40   // высота клетки поля

void _ fastcall NewGame() ; // новая игра — "разбрасывает" мины
void _ fastcall Open (int row, int col); /* открывает текущую и соседние
                                            пустые клетки */

// нажатие кнопки мыши на игровом поле
void _ fastcall TForm1 :: FormMouseDown (TObject *Sender, TMouseButton
Button,
      TShiftState Shift, int x, int y)
(
     if ( status == 2) return;

     if ( status == 0) status = 1,-

     x -= LEFT;
     y -= TOP;
```

```
    if (x > 0 && y > 0)
    {
        // преобразуем координаты мыши
        // в индексы клетки поля
        int row = y/H + 1;
        int col = x/W + 1;

        if (Button == mbLeft)
        {
           if ( Pole[row][col] == 9)
           {
              Pole[row][col] +=100;
              status = 2;  // игра закончена
              ShowPole(status);
           }
           else if ( Pole[row][col] < 9)
                {
                   Open(row,col);
                   ShowPole(status);
                }
        }
        else if (Button == mbRight)
             {
               nFlag++;
               if ( Pole[row][col] == 9)
                  nMin++;
               Pole[row][col] += 200;  // поставили флаг
               if (nMin == NM && nFlag == NM)
               {
                 status = 2;  // игра закончена
                 ShowPole(status);
               )
               else Kletka(row, col, status);
             }
    }
}

// Функция обработки события OnCreate обычно используется .
// для инициализации глобальных переменных
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   // В неотображаемые эл-ты массива, которые соответствуют
   // клеткам по границе игрового поля, запишем число -3.
```

```
    // Это значение используется функцией Open для завершения
    // рекурсивного процесса открытия соседних пустых клеток.
    for ( int row=0; row <= MR+1; row++)
         for ( int col=0; col <= MC+1; col++)
        Pole[row][col] = -3;

    NewGame();  // "разбросать" мины
    Form1->ClientWidth = W*MC;
    Form1->ClientHeight = H*MR+TOP+1;
}

// Вывод поля как результат обработки события Paint
// позволяет проводить любые манипуляции с формой
// во время работы программы
void _ fastcall TForm1: : FormPaint (TObject *Sender)
{
        ShowPole(status);
}

// Показывает поле
void _ fastcall TForm1 : : ShowPole ( int status)
{
    for ( int row = 1; row <= MR; row++)
         for ( int col = 1; col <= MC; col++)
              Kletka(row, col, status);
}

// рисует на экране клетку
void _ fastcall TForm1: :Kletka (int row, int col, int status)
{
     int x = LEFT + (col-1)* W;
     int y = TOP + (row-1)* H;

     if (status == 0)  // начало игры
     {
        // клетка — серый квадрат
        Canvas->Brush->Color = clLtGray;
        Canvas->Rectangle(x-1,y-1,x+W,y+H);
        return;
     }

    // во время (status = 1) и в конце (status = 2) игр
    if ( Pole[row][col] < 100)
    {
         // клетка не открыта
         Canvas->Brush->Color = clLtGray;  // не открытые — серые
```

```
        Canvas->Rectangle(x-1,y-1,x+W,y+H);
        if (status == 2 && Pole[row] [col] == 9)
            Mina ( x, y) ; // игра закончена, показать мину
        return;
    }

    // клетка открыта
    Canvas->Brush~>Color = clWhite;        // открытые - белые
    Canvas->Rectangle(x-1,y-1,x+W,y+H) ;
    if  ( Pole [row] [col] == 100) // клетка открыта, но она пустая
        return;

    if ( Pole[row] [col] >= 101 && Pole [row] [col] <= 108)
    {
            Canvas->Font->Size = 14;
            Canvas->Font->Color = clBlue;
            Canvas->TextOutA(x+3,y+2,IntToStr(Pole[row][col]-100)) ;
            return;
    }

    if ( Pole[row] [col] >= 200)
        Flag(x, y) ;

    if (Pole[row] [col] == 109) // на этой мине подорвались!
    (
        Canvas->Brush->Color = clRed;
        Canvas->Rectangle(x,y,x+W,y+H);
    }
    if (( Pole[row] [col] % 10 == 9)  &&  (status == 2))
            Mina( x, y) ;
}


// рекурсивная функция открывает текущую и все соседние
// клетки, в которых нет мин
void__fastcall Open(int row, int col)
{
    if (Pole[row][col] == 0)
    {
        Pole[row][col] = 100;
        // открываем клетки слева, справа, снизу и сверху
        Open(row,col-1);
        Open(row-1,col);
        Open(row,col+1);
        Open(row+1,col);
```

```
        // открываем примыкающие диагонально
        Open(row-1,col-1);
        Open(row-1,col+1);
        Open(row+1,col-1);
        Open(row+1,col+1);
    }

    else
        // -3 — это граница игрового поля
        if (Pole[row][col] < 100 && Pole[row][col] != -3)
            Pole[row][col] += 100;
}

// новая игра — генерирует новое поле
void _fastcall NewGame()
{
   // Очистим эл-ты массива, соответствующие отображаемым
   // клеткам, а в неотображаемые (по границе игрового поля)
   // запишем число -3. Уникальное значение клеток границы
   // используется функцией Open для завершения рекурсивного
   // процесса открытия соседних пустых клеток.
   int row,col;
   for (row=0; row <= MR+1; row++)
        for (col=0; col <= MC+1; col++)
            Pole[row][col] = -3;
   for (row=1; row <= MR; row++)
        for (col=1; col <= MC; col++)
            Pole[row][col] = 0;

   // расставим мины
   time_t t;    // используется ГСЧ
   srand((unsigned) time(&t)); // инициализация ГСЧ
   int n = 0; // кол-во мин
   do
   {
       row = rand() % MR +1;
       col = rand() % MC +1;
       if ( Pole[row][col] != 9)
       {
           Pole[row][col] = 9;
       n++;
       }
   }
   while ( n < 10);
```

```
    // вычисление кол-ва мин в соседних клетках
    int k;
    for ( row = 1; row <= MR; row++)
        for ( col = 1; col <= MC; col++)
        if ( Pole[row][col] != 9) {
         k =0;
         if ( Pole[row-1] [col-1] == 9) k++;
         if ( Pole[row-1] [col]   == 9) k++;
         if ( Pole[row-1] [col+1] == 9) k++;
         if ( Pole[row] [col-1]   == 9) k++;
         if ( Pole[row] [col+1]   == 9) k++;
         if ( Pole[row+1] [col-1] == 9) k++;
         if ( Pole[row+1] [col]    = 9) k++;
         if ( Pole[row+1] [col+1] == 9) k++;
         Pole[row] [col] = k;
        }
    status = 0; // начало игры
    nMin = 0;   // нет обнаруженных мин
    nFlag = 0;  // нет флагов
}

// рисует мину
void _ fastcall TForm1: :Mina (int x, int y)
{
    Canvas->Brush->Color = clGreen;
    Canvas->Pen->Color = clBlack;
    Canvas->Rectangle(x+16,y+26,x+24,y+30);

    // корпус
    Canvas->Rectangle(x+8,y+30,x+32,y+34);
    Canvas->Pie(x+6,y+28,x+34,y+44,x+34,y+36,x+6,y+36);

    // полоса на корпусе
    Canvas->MoveTo(x+12,y+32); Canvas->LineTo(x+28,y+32);

    // основание
    Canvas->MoveTo(x+8,y+36); Canvas->LineTo(x+32,y+36);

    // вертикальный "ус"
    Canvas->MoveTo(x+20,y+22); Canvas->LineTo(x+20,y+26);

    // боковые "усы"
    Canvas->MoveTo(x+8, y+30); Canvas->LineTo(x+6,y+28);
    Canvas->MoveTo(x+32,y+30); Canvas->LineTo(x+34,y+28);
}
```

```
// рисует флаг
void _ fastcall TForm1 : : Flag ( int x, int y)
{
    TPoint p[4]; // координаты флажка и нижней точки древка

    // точки флажка
    p[0].x=x+4;   p[0].y=y+4;
    p[1].x=x+30;  p[1].y=y+12;
    p[2].x=x+4;   p[2].y=y+20;

    // установим цвет кисти и карандаша
    Canvas->Brush->Color = clRed;
    Canvas->Pen->Color = clRed; // чтобы контур флажка был красный

    Canvas->Polygon(p, 2); // флажок

    // древко
    Canvas->Pen->Color = clBlack;
    Canvas->MoveTo(p[0].x, p[0].y);
    Canvas->LineTo(x+4,y+36);

    TPoint m[5];      // буква M

    m[0].x=x+8;m[0].y=y+14;
    m[1].x=x+8; m[1].y=y+8;
    m[2].x=x+10;m[2].y=y+10;
    m[3].x=x+12;m[3].y=y+8;
    m[4].x=x+12;m[4].y=y+14;
    Canvas->Pen->Color = clWhite;
    Canvas->Polyline(m,4);
    Canvas->Pen->Color = clBlack;
}

// команда главного меню Новая игра
void _ fastcall TForm1: :N1Click(TObject *Sender)
{
        NewGame();
        ShowPole(status);
}

// выбор в меню "?" команды О программе
void _ fastcall TForm1: :N4Click (TObject *Sender)
{
    AboutForm->Top = Form1->Top + Form1->Height/2
                          - AboutForm->Height/2;
```

```
    AboutForm->Left = Form1->Left + Form1->Width/2
                      - AboutForm->Width/2;
    AboutForm->ShowModal();
}

// выбор в меню "?" команды Справка
void__fastcall TForm1::N3Click(TObject *Sender)
{
    WinHelp(Form1->Handle,"saper.hlp",HELP_CONTEXT,1);
}
```

Листинг 10.13. Модуль формы О программе (saper2_.cpp)

```
#include <vcl.h>
ttpragmahdrstop

#include "saper_2.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TAboutForm *AboutForm;

_fastcall TAboutForm: : TAboutForm(TComponent*Owner)
        : TForm(Owner)
{
}

// Выбор URL-адреса
void __fastcall TAboutForm::Label5Click(TObject *Sender)
{
/*   В функцию ShellExute надо передать указатель на null terminated
    строку (char*). Свойство Caption — это AnsiString. Преобразование
    Ansi-строки в указатель на nt-строку выполняет метод c_str()
*/

    // открыть файл, имя которого находится в поле Label5
    ShellExecute(AboutForm->Handle,"open",Label5->Caption.c_str(),
                 NULL,NULL,SW_RESTORE);
}

// щелчок на кнопке OK
void __fastcall TAboutForm: :Button1Click(TObject *Sender)
{
    ModalResult = mrOk;
}
```

Очистка диска

Открыв каталог какого-либо проекта C++ Builder, можно увидеть, что помимо файлов, составляющих *проект* и используемых компилятором для генерации выполняемого файла (bpr-, dfm-, h- и cpp-файлы), в каталоге есть файлы с расширением obj, tds и еще несколько файлов, расширение которых начинается значком ~ (рис. 10.20).

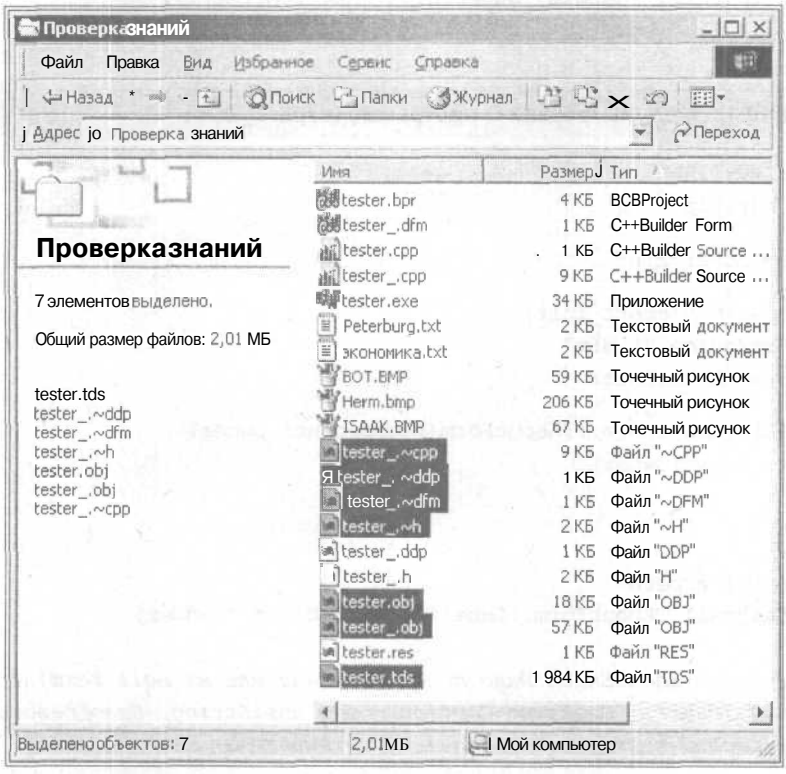


Рис. 10.20. Содержимое каталога проекта "Проверка знаний". Файлы, выделенные цветом, по окончании работы над проектом можно удалить

Файлы с расширением tds и obj создает компилятор в процессе генерации выполняемого файла. Это временные файлы. По окончании работы над проектом они не нужны, и их можно удалить, тем более что размер tds-файла может достигать нескольких мегабайт. Файлы, расширение которых начинается значком ~, — это резервные копии соответствующих файлов проекта. По окончании работы над проектом их тоже можно удалить с диска.

Программа "Очистка диска" удаляет из текущего каталога и всех его подкаталогов файлы, которые по окончании работы над проектом не нужны

(если удалить временные файлы проекта, работа над которым еще не завершена, то в процессе компиляции они будут созданы снова).

Окно программы "Очистка диска" в начале ее работы приведено на рис. 10.21. Для выбора каталога, который надо "почистить", используется стандартное диалоговое окно **Обзор папок,** которое появляется в результате щелчка на кнопке **Каталог.**

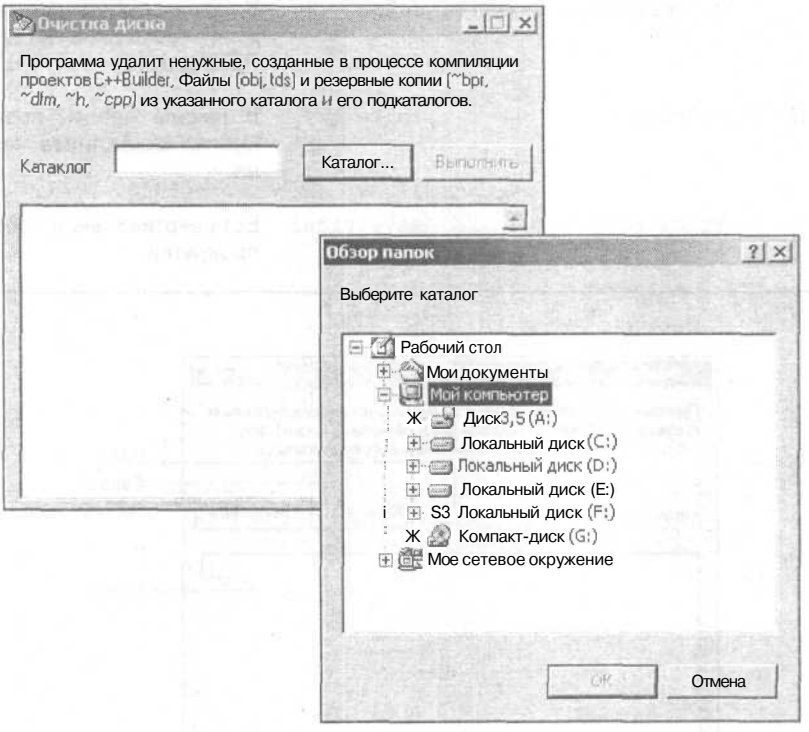


Рис. 10.21. Окно программы "Очистка диска" в начале ее работы

Процесс активизации процесса очистки активизирует кнопка **Выполнить,** которая становится доступной только после того, как пользователь выберет каталог. В процессе очистки диска программа выводит в поле Memo имена удаленных файлов.

Вид формы программы приведен на рис. 10.22. После того как компоненты будут добавлены в форму, необходимо выполнить их настройку — задать значения свойств (табл. 10.7).

Текст программы "Очистка диска" приведен в листинге 10.14.

Таблица 10.7. *Настройка компонентов*

Компонент	Свойство	Значение	Комментарий
Form1	BorderStyle	bsSingle	Тонкая граница окна. Во время работы программы нельзя изменить размер окна, захватив границу мышью
Form1	BorderIcins.biMaximize	false	Во время работы программы в заголовке окна нет кнопки **Развернуть**
Button2	Enabled	false	В начале работы программы кнопка **Выполнить** недоступна
Memo1	ScrollBars	ssVertical	Есть вертикальная полоса прокрутки

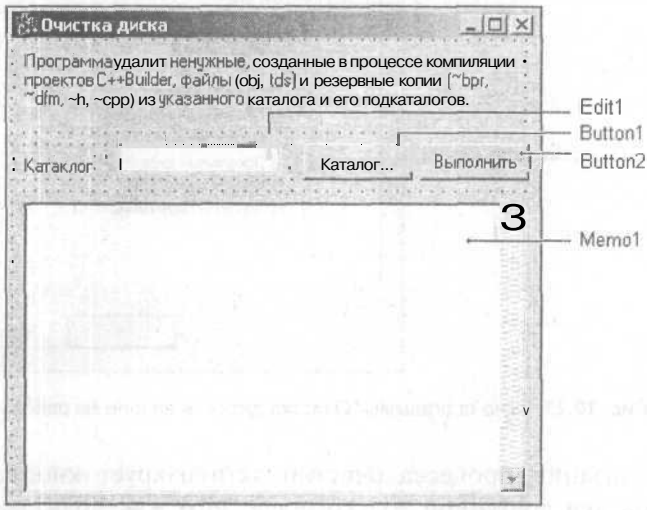


Рис. 10.22. Форма программы "Очистка диска"

Листинг 10.14. Очистка диска

```
#include <vcl.h>
#pragma hdrstop

#include "clear_.h"
```

```
#include <FileCtrl.hpp> // для доступа к SelectDirectory

#pragma package (smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

AnsiString Directory; // каталог, в котором находятся проекты C++ Builder
AnsiString cDir;      // текущий каталог
AnsiString FileExt;   // расширение файла

int n = 0;            // количество удаленных файлов


// щелчок на кнопке Каталог
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString dir;  // каталог, который выбрал пользователь
    if ( SelectDirectory("Выберите каталог","", dir))
    {
        // диалог Выбор файла завершен щелчком на кнопке OK
        Edit1->Text = dir;
        Button2->Enabled = true; // теперь кнопка Выполнить доступна
    };
}

// удаляет ненужные файлы из текущего каталога и его подкаталогов
void __fastcall Clear(void)
{

   TSearchRec SearchRec; // информация о файле или каталоге

   cDir = GetCurrentDir()+"\\";

   if ( FindFirst("*.*", faArchive, SearchRec) == 0)
       do {
            // проверим расширение файла
            int p = SearchRec.Name.Pos(".");
            FileExt = SearchRec.Name.SubString(p+1,MAX_PATH);
            if ( ( FileExt[1] == '~') || ( FileExt == "obj") ||
                 ( FileExt == "tds"))
```

```
            {
                Form1->Memo1->Lines->Add(cDir+SearchRec.Name);
                DeleteFile(SearchRec.Name);
                n++;
            }
        }
        while ( FindNext(SearchRec) == 0);

        // обработка подкаталогов текущего каталога
        if ( FindFirst ("*", faDirectory, SearchRec) == 0)
           do
              if ( (SearchRec.Attr & faDirectory) == SearchRec.Attr)
              {
                 // каталоги ".. " и "." тоже каталоги,
                 // но в них входить не надо !!!
                 if (( SearchRec.Name != ".") &&
                                    (SearchRec.Name != ".."))
                 {
                       ChDir(SearchRec.Name); // войти в подкаталог
                       Clear();               // очистить каталог
                       ChDir("..");           // выйти из каталога
                 };
              }
           while ( FindNext(SearchRec) == 0);
}

// щелчок на кнопке Выполнить
void _ fastcall TForm1: :Button2Click (TObject *Sender)
{
    Memo1->Clear();          // очистить поле Memo1
    Directory = Edit1->Text;// каталог, который выбрал пользователь
    ChDir(Directory);        // войти в каталог

    Clear();                 // очистить текущий каталог и его подкаталоги

    Memo1->Lines->Add("");
    if (n)
       Memo1->Lines->Add("Удалено файлов: " + IntToStr(n));
     else
       Memo1->Lines->Add(
            "В указанном каталоге нет файлов, которые надо удалить.");
}
```

Основную работу (удаление файлов) выполняет *рекурсивная функция* clear (рекурсивной называют функцию, которая в процессе работы вызывает сама

себя). Решение реализовать функцию clear как рекурсивную не случайно: функция обрабатывает каталоги компьютера, которые являются *рекурсивными объектами*. Рекурсивным называют объект, частично состоящий из объектов этого же типа.

Алгоритм функции clear приведен на рис. 10.23.

Рис. 10.23. Алгоритм функции Clear

Сначала функция clear обрабатывает текущий каталог: просматривает все файлы и удаляет те, которые надо удалить. Просмотр файлов обеспечивают функции `FindFirst` и `FindNext`. Функция `FindFirst` просматривает каталог, указанный при ее вызове, и записывает в структуру `SearchRec` имя первого из найденных файлов, имя которого соответствует маске. В данной программе маска *.*, т. е. функция выбирает первый по порядку файл. Если файл найден, то выполняется проверка его расширения. Если расширение файла obj, tds или начинается со значка ~, то имя файла добавляется в поле

Memo1, а сам файл удаляется с диска. Удаляет файл функция DeleteFile. После обработки первого файла для поиска следующего вызывается функция FindNext. После того как все файлы текущего каталога будут обработаны, функция clear проверяет, есть ли в текущем каталоге подкаталоги. Проверку выполняет функция FindFirst, которой в качестве параметра передается константа faDirectory, информирующая функцию о том, что надо искать имена каталогов, а не файлов. Если в текущем каталоге нет подкаталогов, то функция clear завершает работу. Если в текущем каталоге есть подкаталоги, то выполняется вход в подкаталог (делает это функция ChDir) и вызов функции clear (для обработки подкаталога функция вызывает саму себя). Если в текущем каталоге нет необработанных каталогов, то она завершает работу и возвращает управление функции clear, которая ее вызвала и которая после этого продолжает обработку "своих" подкаталогов.

Вывод окна **Обзор папок** выполняет функция SelectDirectory, которую вызывает функция обработки события click на кнопке **Каталог.** Для доступа к этой функции в текст программы надо включить директиву #include <FileCtrl.hpp>.

ПРИЛОЖЕНИЕ 1

C++ Builder - краткий справочник

Приложение представляет собой краткий справочник по компонентам и функциям C++ Builder.

Компоненты

В этом разделе приведено краткое описание базовых компонентов C++ Builder. Подробное описание этих и других компонентов можно найти в справочной системе.

Форма

Форма (объект тип `TForm`) является основой программы. Свойства формы (табл. П1.1) определяют вид окна программы.

Таблица П1.1. *Свойства формы (объекта TForm)*

Свойство	Описание
Name	Имя формы. В программе имя формы используется для управления формой и доступа к компонентам формы
Caption	Текст заголовка
Top	Расстояние от верхней границы формы до верхней границы экрана
`Left`	Расстояние от левой границы формы до левой границы экрана
Width	Ширина формы
Height	Высота формы
`ClientWidth`	Ширина рабочей (клиентской) области формы, т. е. без учета ширины левой и правой границ

***Таблица П1.1** (окончание)*

Свойство	Описание
ClientHeight	Высота рабочей (клиентской) области формы, т. е. без учета высоты заголовка и ширины нижней границы формы
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 при помощи мыши изменить размер окна. Изменить размер окна с тонкой границей нельзя. Если граница отсутствует, то на экран во время работы программы будет выведено окно без заголовка. Положение и размер такого окна во время работы программы изменить нельзя
BorderIcons	Кнопки управления окном. Значение свойства определяет, какие кнопки управления окном будут доступны пользователю во время работы программы. Значение свойства задается путем присвоения значений уточняющим свойствам biSystemMenu, biMinimize, biMaximize и biHelp. Свойство biSystemMenu определяет доступность кнопки **Свернуть** и кнопки системного меню, biMinimize— кнопки **Свернуть,** biMaximize — кнопки **Развернуть,** biHelp — кнопки вывода справочной информации
Icon	Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню
Color	Цвет фона. Цвет можно задать, указав название цвета или элемент цветовой схемы операционной системы. Во втором случае цвет компонента "привязан" к цветовой схеме операционной системы и будет изменяться при каждой смене цветовой схемы
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)
Canvas	Поверхность, на которую можно вывести графику

Label

Компонент Label (рис. П1.1) предназначен для вывода текста на поверхность формы. Свойства компонента (табл. П1.2) определяют вид и расположение текста.

Рис. П1.1. Компонент Label — поле вывода текста

Таблица П1.2. *Свойства компонента Label (поле вывода текста)*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
Caption	Отображаемый текст
Left	Расстояние от левой границы поля вывода до левой границы формы
Top	Расстояние от верхней границы поля вывода до верхней границы формы
Height	Высота поля вывода
Width	Ширина поля вывода
AutoSize	Признак того, что размер поля определяется его содержимым
wordwrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (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)

Edit

Компонент Edit (рис, П1.2) представляет собой поле ввода-редактирования строки символов. Свойства компонента приведены в табл. П1.3.

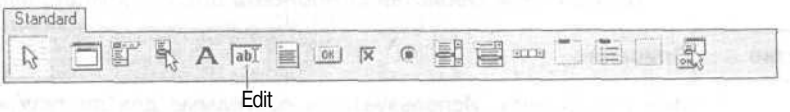


Рис. П1.2. Компонент Edit — поле ввода-редактирования строки символов

Таблица П1.3. *Свойства компонента Edit (поле редактирования)*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в частности для доступа к тексту, введенному в поле редактирования
Text	Текст, находящийся в поле ввода и редактирования
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы
Height	**Высота поля**
Width	Ширина поля
Font	Шрифт, используемый для отображения вводимого текста
ParentFont	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно true, то при изменении свойства Font формы автоматически меняется значение свойства Font компонента
Enabled	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно false, то текст в поле редактирования изменить нельзя
visible	Позволяет скрыть компонент (false) или сделать его видимым (true)

Button

Компонент Button (рис. П1.3) представляет собой командную кнопку. Свойства компонента приведены в табл. П1.4.

Рис. П1.3. Компонент Button — командная кнопка

Таблица П1.4. *Свойства компонента Button (командная кнопка)*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам

Таблица П1.4 (окончание)

Свойство	Описание
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 недоступна— например, в результате щелчка на кнопке, событие Click не возникает
Visible	Позволяет скрыть кнопку (false) или сделать ее видимой (true)
Hint	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, надо, чтобы значение свойства showHint было true)
ShowHint	Разрешает (true) или запрещает (false) отображение подсказки при позиционировании указателя на кнопке

Memo

Компонент Memo (рис. П1.4) представляет собой элемент редактирования текста, который может состоять из нескольких строк. Свойства компонента приведены в табл. П1.5.

Рис. П1.4. Компонент Memo

Таблица П1.5. Свойства компонента Memo

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента
Text	Текст, находящийся в поле Memo. Рассматривается как единое целое

Таблица П1.5 *(окончание)*

Свойство	Описание
Lines	Массив строк, соответствующий содержимому поля. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
Left	Расстояние от левой границы поля до левой границы формы
Top	Расстояние от верхней границы поля до верхней границы формы
Height	**Высота поля**
width	Ширина поля
Font	Шрифт, используемый для отображения вводимого текста
ParentFont	Признак наследования свойств шрифта родительской формы

RadioButton

Компонент RadioButton (рис. П1.5) представляет зависимую кнопку, состояние которой определяется состоянием других кнопок группы. Свойства компонента приведены в табл. **П1.6.**

Если в диалоговом окне надо организовать несколько групп радиокнопок, то каждую группу следует представить компонентом RadioGroup.

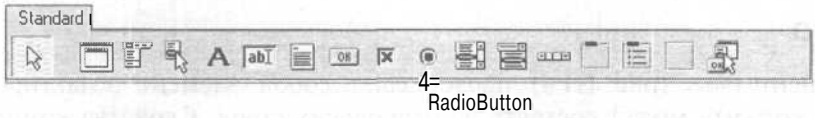


Рис. П1.5. Компонент RadioButton

Таблица П1.6. *Свойства компонента RadioButton*

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента
Caption	Текст, который находится справа от кнопки
Checked	Состояние, внешний вид кнопки. Если кнопка выбрана, то значение свойства Checked равно true, если кнопка не выбрана, то false
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

Таблица П1.6 *(окончание)*

Свойство	Описание
Width	Ширина поля вывода поясняющего текста
Font	Шрифт, используемый для отображения поясняющего текста
ParentFont	Признак наследования характеристик шрифта родительской формы

CheckBox

Компонент CheckBox (рис. П1.6) представляет собой независимую кнопку (переключатель). Свойства компонента приведены в табл. П1.7.

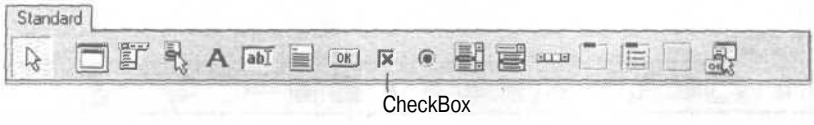


Рис. П1.6. Компонент CheckBox

Таблица П1.7. *Свойства компонента CheckBox*

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента
Caption	Текст, который находится справа от флажка
Checked	Состояние, внешний вид флажка. Если флажок установлен (в квадратике есть "галочка"), то значение свойства 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); cbGrayed (серый, неопределенное состояние); cbUnChecked (сброшен)
AllowGrayed	Свойство определяет, может ли флажок быть 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
Left	Расстояние от левой границы флажка до левой границы формы
Top	Расстояние от верхней границы флажка до верхней границы формы

Таблица П1.7 *(окончание)*

Свойство	Описание
Height	Высота поля вывода поясняющего текста
width	Ширина поля вывода поясняющего текста
Font	Шрифт, используемый для отображения поясняющего текста
ParentFont	Признак наследования характеристик шрифта родительской формы

ListBox

Компонент `ListBox` (рис. П1.7) представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. П1.8.

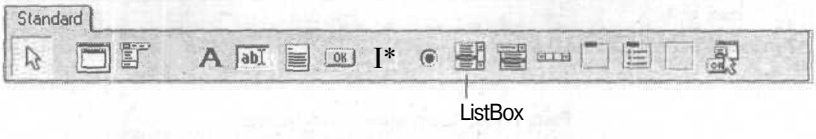


Рис. П1.7. Компонент ListBox

Таблица П1.8. *Свойства компонента ListBox*

Свойство	Описание
Name	Имя компонента. В программе используется для доступа к компоненту и его свойствам
Items	Элементы списка — массив строк
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` (минус один)
`Left`	Расстояние от левой границы списка до левой границы формы
Top	Расстояние от верхней границы списка до верхней границы формы
Height	Высота поля списка
Width	Ширина поля списка
Font	Шрифт, используемый для отображения элементов списка
ParentFont	Признак наследования свойств шрифта родительской формы

ComboBox

Компонент ComboBox (рис. П1.8) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. П1.9.

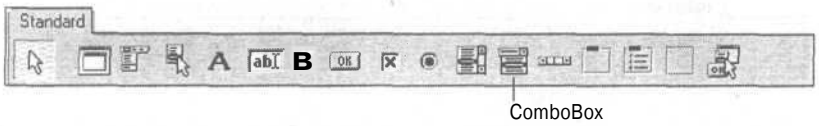


Рис. П1.8. Компонент ComboBox

Таблица П1.9. *Свойства компонента* `ComboBox`

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента
Text	Текст, находящийся в поле ввода/редактирования
Items	Элементы списка — массив строк
Count	Количество элементов списка
ItemIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не был выбран, то значение свойства равно -1 (минус один)
Sorted	Признак необходимости автоматической сортировки (true) списка после добавления очередного элемента
DropDownCount	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше, чем DropDownCount, то появляется вертикальная полоса прокрутки
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы
Height	Высота компонента (поля ввода/редактирования)
Width	Ширина компонента
Font	Шрифт, используемый для отображения элементов списка
ParentFont	Признак наследования свойств шрифта родительской формы

StringGrid

Компонент stringGrid (рис. П1.9) представляет собой таблицу, ячейки которой содержат строки символов. Свойства компонента stringGrid приведены в табл. ШЛО.

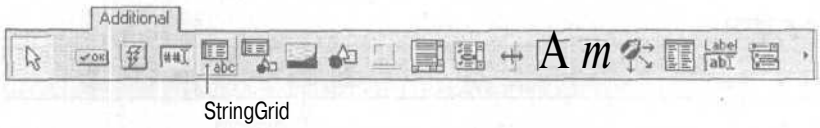


Рис. П1.9. Компонент StringGrid

Таблица П1.10. *Свойства компонента stringGrid*

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
ColCount	Количество колонок таблицы
RowCount	Количество строк таблицы
DefaultColWidth	Ширина колонок таблицы
DefaultRowHeight	Высота строк таблицы
FixedCols	Количество зафиксированных слева колонок таблицы. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте
FixedRows	Количество зафиксированных сверху строк таблицы. Зафиксированные строки выделяются цветом и при вертикальной прокрутке таблицы остаются на месте
Cells	Соответствующий таблице двумерный массив. Ячейке таблицы, находящейся на пересечении столбца с номером col и строки с номером row, соответствует элемент cells [col] [row]
GridLineWidth	Ширина линий, ограничивающих ячейки таблицы
Left	Расстояние от левой границы поля таблицы до левой границы формы
Top	Расстояние от верхней границы поля таблицы до верхней границы формы
Height	Высота поля таблицы
Width	Ширина поля таблицы

Таблица П1.10 (окончание)

Свойство	Описание
Options.goEditing	Признак допустимости редактирования содержимого ячеек таблицы (true — редактирование разрешено, false — запрещено)
Options.goTab	Разрешает (true) или запрещает (false) использование клавиши <TaЬ> для перемещения курсора в следующую ячейку таблицы
Options.goAlwaysShowEditor	Признак нахождения компонента в режиме редактирования. Если значение свойства false, то для того чтобы в ячейке появился курсор, надо или начать набирать текст или нажать клавишу <F2>, или сделать щелчок мышью в ячейке таблицы
Font	Шрифт, используемый для отображения содержимого ячеек таблицы
ParentFont	Признак наследования характеристик шрифта формы

Image

Компонент image (рис. П1.10) обеспечивает вывод на поверхность формы иллюстраций, представленных в формате BMP (чтобы компонент можно было использовать для отображения иллюстраций в формате JPG, надо подключить модуль JPEG — включить в текст программы директиву #include <jpeg.hpp>). Свойства компонента image приведены в табл. П1.11.

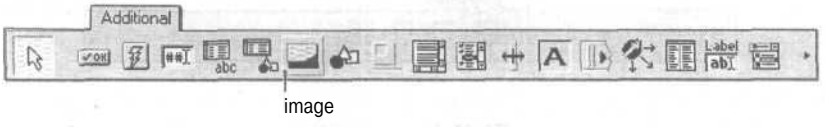


Рис. П1.10. Компонент Image

Таблица П1.11. Свойства компонента image

Свойство	Описание
Picture	Иллюстрация, которая отображается в поле компонента
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации, а значение свойств AutoSize, Strech и Proportional равно false, то отображается часть иллюстрации

Таблица П1.11 (окончание)

Свойство	Описание
Proportional	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства AutoSize должно быть false
Strech	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации
Center	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно false, то картинка прижата к правой границе компонента, если true — то картинка располагается по центру
Visible	Отображается ли компонент, и, соответственно, иллюстрация, на поверхности формы
Canvas	Поверхность, на которую можно вывести графику (см. **табл.** П1.25)

Timer

Компонент Timer (рис. П1.11) обеспечивает генерацию последовательности событий OnTimer. Свойства компонента приведены в табл. П1.12.

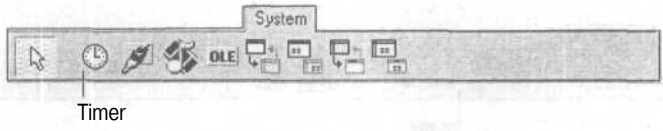


Рис. П1.11. Компонент Timer

***Таблица П1.12.** Свойства компонента Timer*

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту
interval	Период генерации события OnTimer. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение true) или запрещает (значение false) генерацию события OnTimer

Animate

Компонент Animate (рис. П1.12) позволяет воспроизводить простую, не сопровождаемую звуком анимацию, кадры которой находятся в AVI-файле. Свойства компонента приведены в табл. П1.13.

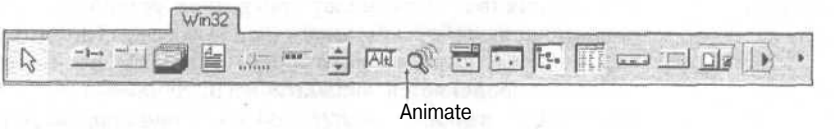


Рис. П1.12. Компонент Animate

Таблица П1.13. *Свойства компонента Animate*

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента и управления его поведением
FileName	Имя AVI-файла, в котором находится анимация, отображаемая при помощи компонента
StartFrame	Номер кадра, с которого начинается отображение анимации
StopFrame	Номер кадра, на котором заканчивается отображение анимации
Activate	Признак активизации процесса отображения кадров анимации
Color	Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация
Transparent	Режим использования "прозрачного" цвета при отображении анимации
Repetitions	Количество повторов отображения анимации

MediaPlayer

Компонент MediaPlayer (рис. П1.13) позволяет воспроизвести видеоролик, звук и сопровождаемую звуком анимацию. Свойства компонента приведены в табл. П1.14.

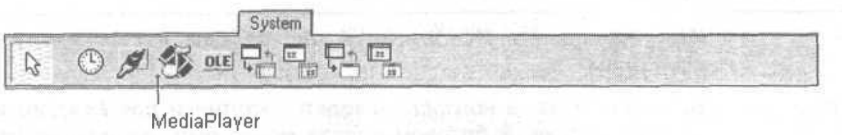


Рис. П1.13. Компонент MediaPlayer

Таблица П1.14. *Свойства компонента* `MediaPlayer`

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента и управлениея работой плеера
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент MediaPlayer. Тип устройства задается именованной константой: dtAutoSelect — тип устройства определяется автоматически; `dtVaweAudio` — проигрыватель звука; `dtAVIVideo` — видеопроигрыватель; dtCDAudio — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
AutoOpen	Признак автоматического открытия сразу после запуска программы файла видеоролика или звукового фрагмента
Display	Определяет компонент, на поверхности которого воспроизводится видеоролик (обычно в качестве экрана для отображения видео используют компонент Panel)
VisibleButtons	Составное свойство. Определяет видимые кнопки компонента. Позволяет сделать невидимыми некоторые кнопки

SpeedButton

Компонент SpeedButton (рис. П1.14) представляет собой кнопку, на поверхности которой находится картинка. Свойства компонента приведены в табл. П1.15.

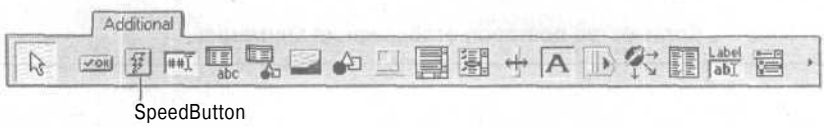


Рис. П1.14. Компонент SpeedButton

Таблица П1.15. *Свойства компонента SpeedButton*

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Glyph	Битовый образ, в котором находятся картинки для каждого из состояний кнопки. В битовом образе может быть до четырех изображений кнопки (рис. П1.15)

***Таблица П1.15** (окончание)*

Свойство	Описание
NumGlyphs	Количество картинок в битовом образе Glyph
Flat	Свойство Flat определяет вид кнопки (наличие границы). Если значение свойства равно true, то граница кнопки появляется только при позиционировании указателя мыши на кнопке
GroupIndex	Идентификатор группы кнопок. Кнопки, имеющие одинаковый идентификатор группы, работают подобно радиокнопкам: нажатие одной из кнопок группы вызывает срабатывание других кнопок этой группы. Чтобы кнопку можно было зафиксировать, значение свойства GroupIndex не должно быть равно нулю
Down	Идентификатор состояния кнопки. Изменить значение свойства можно, если значение свойства GroupIndex не равно нулю
AllowAllUp	Свойство определяет возможность отжать кнопку. Если кнопка нажата и значение свойства равно true, то кнопку можно отжать
Left	Расстояние от левой границы кнопки до левой границы формы
Top	Расстояние от верхней границы кнопки до верхней границы формы
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 недоступна
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, надо, чтобы значение свойства ShowHint было равно true)
ShowHint	Разрешает (true) или запрещает (false) отображение подсказки при позиционировании указателя на кнопке

Рис. П1.15. Структура и пример битового образа Glyph: картинки, соответствующие состоянию кнопки

UpDown

Компонент UpDown (рис. П1.16) представляет собой две кнопки, используя которые можно изменить значение внутренней переменной-счетчика на определенную величину. Увеличение или уменьшение значения происходит при каждом щелчке на одной из кнопок. Свойства компонента приведены в табл. П1.16.

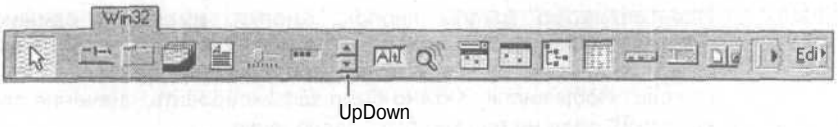


Рис. П1.16. Компонент UpDown

Таблица П1.16. *Свойства компонента UpDown*

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

Table

Компонент Table (рис. П1.17) представляет всю таблицу базы данных. Свойства компонента приведены в табл. П1.17.

Рис. П1.17. Компонент `Table` — таблица базы данных

Таблица П1.17. *Свойства компонента Table*

Свойство	Определяет
Name	Имя компонента. Используется для доступа к свойствам компонента
`DatabaseName`	Имя базы данных, частью которой является таблица (файл данных), для доступа к которой применяется компонент. В качестве значения свойства следует использовать псевдоним базы данных
`TableName`	Имя файла данных (таблицы данных), для доступа к которому используется компонент
`TableType`	Тип таблицы. Таблица может быть набором данных в формате Paradox (ttParadox), dBase (ttDBase), FoxPro (ttFoxPro) или представлять собой форматированный текстовый файл (`ttASCII`)
Active	Признак того, что таблица активна (файл данных открыт). В результате присваивания свойству значения `true` происходит открытие файла таблицы

Query

Компонент Query (рис. П1.18) представляет часть базы данных — записи, содержимое которых удовлетворяет критерию SQL-запроса к таблице. Свойства компонента приведены в табл. П1.18.

Таблица П1.18. *Свойства компонента Query*

Свойство	Определяет
Name	Имя компонента. Используется компонентом Datasource для связи результата выполнения запроса (набора записей) с компонентом, обеспечивающим просмотр записей, например `DBGrid`
SQL	Записанный на языке SQL-запрос к базе данных (к таблице)
Active	При присвоении свойству значения `true` активизирует выполнение запроса

Таблица П1.18 (окончание)

Свойство	Определяет
RecordCount	Количество записей в базе данных, удовлетворяющих критерию запроса

Рис. П1.18. Компонент Query обеспечивает выбор информации из базы данных

DataSource

Компонент DataSource (рис. П1.19) обеспечивает связь между данными, представленными компонентом Table или Query, и компонентами отображения данных (DBEdit, DBMemo, DBGrid). Свойства компонента приведены в табл. П1.19.

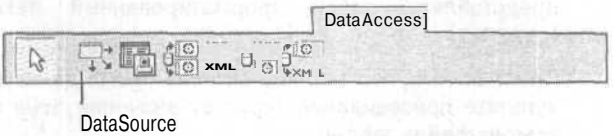


Рис. П1.19. Компонент DataSource обеспечиват связь между данными и компонентом просмотра-редактирования

Таблица П1.19. Свойства компонента DataSource

Свойство	Определяет
Name	Имя компонента. Используется компонентом отображения данных для доступа к компоненту и, следовательно, к данным, связь с которыми обеспечивает компонент
DataSet	Компонент, представляющий собой входные данные (Table ИЛИ Query)

DBEdit, DBMemo, DBText

Компоненты DBEdit и DBMemo (рис. П1.20) обеспечивают просмотр и редактирование полей записи базы данных, компонент DBText — только просмотр. Свойства компонентов приведены в табл. П1.20.

Рис. П1.20. Компоненты просмотра и редактирования полей БД

Таблица П1.20. *Свойства компонентов DBText, DBEdit и* `DBMemo`

Свойство	Определяет
Name	Имя компонента. Используется для доступа к свойствам компонента
DataSource	Компонент-источник данных
`DataField`	Поле базы данных, для отображения или редактирования которого используется компонент

DBGrid

Компонент DBGrid (рис. П1.21) используется для просмотра и редактирования базы данных в режиме таблицы. Свойства компонента приведены в табл. П1.21.

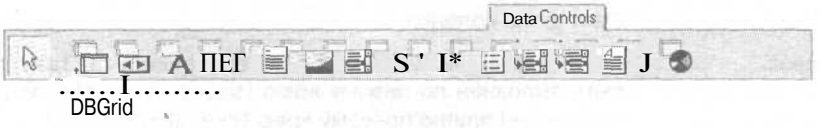


Рис. П1.21. Компонент DBGrid обеспечивает работу с базой данных в режиме таблицы

Таблица П1.21. *Свойства компонента DBGrid*

Свойство	Описание
Name	Имя компонента
DataSource	Источник отображаемых в таблице данных (компонент `DataSource`)
Columns	Свойство columns представляет собой массив объектов типа `TColumn`, каждый из которых определяет колонку таблицы и отображаемую в ней информацию (см. табл. П1.22)
Options	Свойство Options определяет настройку компонента
`Options.dgTitles`	Разрешает вывод строки заголовка столбцов

***Таблица П1.21** (окончание)*

Свойство	Описание
`Options.dgIndicator`	Разрешает вывод колонки индикатора. Во время работы с базой данных текущая запись помечается в колонке индикатора треугольником, новая запись — звездочкой, редактируемая — специальным значком
`Options.dgColumnResize`	Разрешает менять во время работы программы ширину колонок таблицы
`Options.dgColLines`	Разрешает выводить линии, разделяющие колонки таблицы
`Options.dgRowLines`	Разрешает выводить линии, разделяющие строки таблицы

***Таблица П1.22.** Свойства объекта `TColumn`*

Свойство	Определяет
`FieldName`	Поле записи, содержимое которого выводится в колонке
Width	Ширину колонки в пикселах
Font	Шрифт, используемый для вывода текста в ячейках колонки
Color	Цвет фона колонки
Alignment	Способ выравнивания текста в ячейках колонки. Текст может быть выровнен по левому краю (`taLeftJustify`), по центру (taCenter) или по правому краю (`taRightJustify`)
`Title.Caption`	Заголовок колонки. Значением по умолчанию является имя поля записи
`Title.Alignment`	Способ выравнивания заголовка колонки. Заголовок может быть выровнен по левому краю (taLeftJustify), по центру (taCenter) или по правому краю (`taRightJustify`)
`Title.Color`	Цвет фона заголовка колонки
`Title.Font`	Шрифт заголовка колонки

DBNavigator

Компонент DBNavigator (рис. П1.22 и П1.23) обеспечивает перемещение указателя текущей записи, активизацию режима редактирования, добавление и удаление записей. Компонент представляет собой совокупность командных кнопок (табл. П1.23). Свойства компонента приведены в табл. П1.24.

Рис. П1.22. Значок компонента DBNavigator

Рис. П1.23. Компонент DBNavigator

Таблица П1.23. *Кнопки компонента DBNavigator*

Кнопка	Обозначение	Действие
К первой	nbFirst	Указатель текущей записи перемещается к первой записи файла данных
К предыдущей	nbPrior	Указатель текущей записи перемещается к предыдущей записи файла данных
К следующей	nbNext	Указатель текущей записи перемещается к следующей записи файла данных
К последней	nbLast	Указатель текущей записи перемещается к последней записи файла данных
Добавить	nbInsert	В файл данных добавляется новая запись
Удалить	nbDelete	Удаляется текущая запись файла данных
Редактирование	nbEdit	Устанавливает режим редактирования текущей записи
Сохранить	nbPost	Изменения, внесенные в текущую запись, записываются в файл данных
Отменить	Cancel	Отменяет внесенные в текущую запись изменения
Обновить	nbRefresh	Записывает внесенные изменения в файл

Таблица ***П1.24.*** *Свойства компонента DBNavigator*

Свойство	Определяет
Name	Имя компонента. Используется для доступа к свойствам компонента

Таблица П1.24 *(окончание)*

Свойство	Определяет
DataSource	Имя компонента, являющегося источником данных. В качестве источника данных может выступать база данных (компонент Database), таблица (компонент Table) или результат выполнения запроса (компонент Query)
VisibleButtons	Видимые командные кнопки

Графика

Canvas

Canvas — это поверхность (формы или компонента image), на которой соответствующие методы (табл. П1.25) могут вычерчивать графические примитивы. Вид графических элементов определяют свойства поверхности, на которой эти элементы вычерчиваются (табл. П1.26).

Таблица П1.25. *Методы объекта canvas*

Метод	Описание
TextOut(x,y,s)	Выводит строку s от точки с координатами (x, y). Шрифт определяет свойство Font поверхности (Canvas), на которую выводится тест, цвет закраски области вывода текста — свойство Brush этой же поверхности
Draw(x,y,b)	Выводит от точки с координатами (x, y) битовый образ b. Если значение свойства Transparent поверхности, на которую выполняется вывод, равно true, то точки, цвет которых совпадает с цветом левой нижней точки битового образа, не отображаются
LineTo(x,y)	Вычерчивает линию из текущей точки в точку с указанными координатами. Вид линии определяет свойство Pen
MoveTo(x,y)	Перемещает указатель текущей точки в точку с указанными координатами
PolyLine(pl)	Вычерчивает ломаную линию. Координаты точек перегиба задает параметр pl — массив структур типа TPoint. Если первый и последний элементы массива одинаковы, то будет вычерчен замкнутый контур. Вид линии определяет свойство Реп
Polygon(pl)	Вычерчивает и закрашивает многоугольник. Координаты углов задает параметр pl — массив структур типа TPoint. Первый и последний элементы массива должны быть одинаковы. Вид границы определяет свойство Реп, цвет и стиль закраски внутренней области — свойство Brush

Таблица П1.25 (окончание)

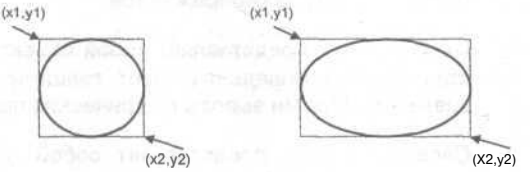
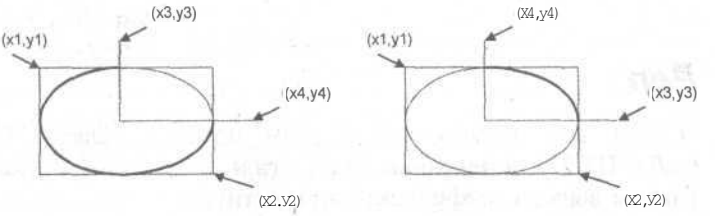
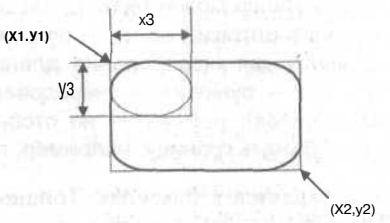
Метод	Описание
Ellipse(x1,y1, x2,y2) [V18]	Вычерчивает эллипс, окружность или круг. Параметры xl, y1, x2 и y2 задают размер прямоугольника, в который вписывается эллипс. Вид линии определяет свойство Pen 
Arc(x1,y1,x2,y2, x3,y3,x4,y4)	Вычерчивает дугу. Параметры xl, yl, x2, y2 определяют эллипс, из которого вырезается дуга, параметры x2, y2, x3, и y4 — координаты концов дуги. Дуга вычерчивается против часовой стрелки от точки (x3, y3) к точке (x4, y4). Вид линии (границы) определяет свойство Pen, цвет и способ закраски внутренней области — свойство Brush 
Rectangle(x1,y1, x2,y2)	Вычерчивает прямоугольник. Параметры xl, y1, x2 и y2 задают координаты левого верхнего и правого нижнего углов. Вид линии определяет свойство Pen, цвет и способ закраски внутренней области — свойство Brush
RoundRec(x1,y1, x2,y2, x3,y3)	Вычерчивает прямоугольник со скругленными углами. Параметры xl, yl, x2 и y2 задают координаты левого верхнего и правого нижнего углов, x3 и y3 — радиус скругления. Вид линии определяет свойство Pen, цвет и способ закраски внутренней области — свойство Brush 

Таблица П1.26. *Свойства объекта Canvas*

Свойство	Описание
Transparent	Признак использования "прозрачного" цвета при выводе битового образа методом Draw. Если значение свойства равно true, то точки, цвет которых совпадают с цветом левой нижней точки битового образа, не отображаются
Pen	Свойство Pen представляет собой объект (см. табл. (11.27), свойства которого определяют цвет, толщину и стиль линий, вычерчиваемых методами вывода графических примитивов
Brush	Свойство Brush представляет собой объект (см. табл. П 1.28), свойства которого определяют цвет и стиль закраски областей, вычерчиваемых методами вывода графических примитивов
Font	Свойство Font представляет собой объект, уточняющие свойства которого определяют шрифт (название, размер, цвет, способ оформления), используемый для вывода на поверхность холста текста

Pen

Объект Pen является свойством объекта canvas. Свойства объекта Pen (табл. П1.27) определяют цвет, стиль и толщину линий, вычерчиваемых методами вывода графических примитивов.

ТаблицаП1.27. *Свойства объекта Pen*

Свойство	Описание
Color	Цвет линии (clBlack — черный; clMaroon — каштановый; clGreen — зеленый; clOlive — оливковый; clNavy— темно-синий; clPurple — розовый; clTeal — зелено-голубой; clGray— серый; clSilver — серебристый; clRed — красный; clLime — салатный; clBlue — синий; clFuchsia — ярко-розовый; clAqua — бирюзовый; clWhite — белый)
Style	Стиль (вид) линии. Линия может быть: psSolid — сплошная; psDash — пунктирная (длинные штрихи); psDot — пунктирная (короткие штрихи); psDashDot — пунктирная (чередование длинного и короткого штрихов); psDashDotDot — пунктирная (чередование одного длинного и двух коротких штрихов); psClear— не отображается (используется, если не надо изображать границу, например, прямоугольника)
Width	Толщина линии задается в пикселах. Толщина пунктирной линии не может быть больше 1

Brush

Объект Brush является свойством объекта canvas. Свойства объекта Brush (табл. П1.28) определяют цвет, стиль закраски внутренних областей контуров, вычерчиваемых методами вывода графических примитивов.

Таблица П1.28. *Свойства объекта Brush*

Свойство	Определяет
Color	Цвет закрашивания замкнутой области
Style	Стиль (тип) заполнения области (bsSolid— сплошная заливка; `bsClear` — область не закрашивается; `bsHorizontal` — горизонтальная штриховка; bsVertical — вертикальная штриховка; bsFDiagonal — диагональная штриховка с наклоном линий вперед; bsBDiagonal — диагональная штриховка с наклоном линий назад; bsCross — горизонтально-вертикальная штриховка, в клетку; bsDiagCross — диагональная штриховка, в клетку)

Функции

В этом разделе приведено краткое описание наиболее часто используемых функций. Подробное их описание можно найти в справочной системе.

Функции ввода и вывода

Таблица П1.29. *Функции ввода и вывода*

Функция	Описание
`InputBox(`*`Заголовок`*`,` *`Подсказка`*`,` *`Значение`*`)`	В результате выполнения функции на экране появляется диалоговое окно, в поле которого пользователь может ввести строку символов. Значением функции является введенная строка. Параметр *`Значение`* задает значение функции "по умолчанию", т. е. строку, которая будет в поле редактирования в момент появления окна
`ShowMessage(s)`	Процедура ShowMessage выводит окно, в котором находится сообщение s и командная кнопка **OK**
`MessageDlg(s,t,b,h)`	Выводит на экран диалоговое окно с сообщением s и возвращает код кнопки, щелчком на которой пользователь закрыл окно. Параметр t определяет тип окна: `mtWarning`— Внимание; `mtError`— ошибка; `mtInformation` информация; `mtConfirmation` запрос; `mtCustom`— пользовательское (без значка).

Таблица П1.29 *(окончание)*

Функция	Описание
(прод.)	Параметр ь (множество — заключенный в квадратные скобки список констант) задает командные кнопки диалогового окна (mbYes, mbNo, mbOK, mbCancel, mbHelp, mbAbort, mbRetry, mbIgnore И mbAll). Параметр h задает раздел справочной системы программы, который появится в результате нажатия кнопки **Help** или клавиши <F1>. Если справочная система не используется, значение параметра должно быть 0. Значение функции равно коду кнопки, которую нажал пользователь (mrAbort, mrYes, mrOk, mrRetry, mrNo, mrCancel, mrIgnoreИЛИ mrAll)

Математические функции

Таблица П1.30. *Математические функции*

Функция	Значение
abs (n)	Абсолютное значение n
sqrt (n)	Квадратный корень из п
exp (п)	Экспонента п
random[V19] (n)	Случайное целое число в диапазоне от 0 до n-1 (перед первым обращением к функции необходимо вызвать функцию randomize (), которая выполнит инициализацию программного генератора случайных чисел)
sin (a)	Синус выраженного в радианах угла а
cos (a)	Косинус выраженного в радианах угла а
tan (a)	Тангенс выраженного в радианах угла *а*
asin (n)	Угол (в радианах), синус которого равен п
acos (n)	Угол (в радианах), косинус которого равен п
atan (n)	Угол (в радианах), тангенс которого равен п

Обратите внимание: для того чтобы в программе были доступны приведенные функции, в ее текст надо включить директиву #include <math.h>.

Величина угла тригонометрических функций должна быть выражена в радианах. Для преобразования величины угла из градусов в радианы используется формула *(a* ·3.1415256) / 180, где *a-* величина угла в градусах; 3.1415926 — число "пи". Вместо константы 3.1415926 можно использовать стандартную именованную константу M_PI. Константа `M_PI` определена в файле math.h.

Функции преобразования

Таблица П1.31. *Функции преобразования*

Функция	Значение функции
`IntToStr(k)`	Строка, являющаяся изображением целого k
`FloatToStr(n)`	Строка, являющаяся изображением вещественного n
`FloatToStrF(n,f,k,m)`	Строка, являющаяся изображением вещественного n. При вызове функции указывают: f — формат; k — точность; т— количество цифр после десятичной точки. Формат определяет способ изображения числа: `ffGeneral` — универсальный; `ffExponent` — научный; `ffFixed` — с фиксированной точкой; `ffNumber` — с разделителями групп разрядов; ffCurrency— финансовый. Точность — нужное общее количество цифр: 7 или меньше для значения типа single, 15 или меньше для значения типа Double и 18 или меньше для значения типа Extended
`StrToInt(s)`	Целое число, изображением которого является строка s
`StrToFloat(s)`	Дробное число, изображением которого является строка s

Функции манипулирования датами и временем

Большинству функций манипулирования датами в качестве параметра передается переменная типа TDateTime, которая хранит информацию о дате и времени.

ДЛЯ ТОГО чтобы **В Программе были ДОСТУПНЫ фуНКЦИИ** DayOf, `WeekOf`, MonthOf и др., в ее текст надо включить директиву `#include <DateUtils.hpp[L29]>`.

Таблица П1.32. *Функции манипулирования датами и временем*

Функция	Значение
`Now()`	Системная дата и время — значение типа `TDateTime`
`DateToStr(dt)`	Строка символов, изображающая дату в формате dd.mm.yyyy
`TimeToStr(dt)`	Строка символов, изображающая время в формате hh:mm:ss
`DayOf(dt)`	День (номер дня в месяце), соответствующий дате, указанной в качестве параметра функции
`MonthOf(dt)`	Номер месяца, соответствующий дате, указанной в качестве параметра функции
`WeekOf(dt)`	Номер недели, соответствующий дате, указанной в качестве параметра функции
`YearOf(dt)`	Год, соответствующий указанной дате
`DayOfWeek(dt)`	Номер дня недели, соответствующий указанной дате: 1 — воскресенье, 2 — понедельник, 3 — вторник и т. д.
`StartOfWeek(w)`	Дата первого дня указанной недели
`HourOf(dt)`	Количество часов
`MinuteOf(dt)`	Количество минут
`SecondOf(dt)`	Количество секунд
`DecodeDate(dt,y,m,d)`	Возвращает год, месяц и день, представленные отдельными числами
`DecodeTime(dt,h,m,s,ms)`	Возвращает время (часы, минуты, секунды и милисекунды), представленное отдельными числами
`FormatDateTime(s,dt)`	Строка символов, представляющая собой дату или время. Способ представления задает строка формата s, например, строка `dd/mm/yyyy` задает, что значением функции является дата, а строка `hh:mm` — время

События

Таблица П1.33. *События*

Событие	Происходит
OnClick	При щелчке кнопкой мыши

Таблица П1.33 (окончание)

Событие	Происходит
OnDblClick	При двойном щелчке кнопкой мыши
OnMouseDown	При нажатии кнопки мыши
OnMouseUp	При отпускании кнопки мыши
OnMouseMove	При перемещении мыши
OnKeyPress	При нажатии клавиши клавиатуры
OnKeyDown	При нажатии клавиши клавиатуры. События OnKeyDown и OnKeyPress — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие OnKeyUp)
OnKeyUp	При отпускании нажатой клавиши клавиатуры
OnCreate	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
OnPaint	При появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном и в других случаях. Событие сообщает о необходимости обновить (перерисовать) окно
OnEnter	При получении элементом управления фокуса
OnExit	При потере элементом управления фокуса

Исключения

ТаблицаП1.34. Типичные исключения

Типисключения	Возникает
EConvertError	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому виду. Наиболее часто возникает при преобразовании строки символов в число
EDivByZero	Целочисленное деление на ноль. При выполнении операции целочисленного деления, если делитель равен нулю
EZeroDivide	Деление на ноль. При выполнении операции деления над дробными операндами, если делитель равен нулю

Таблица П1.34 *(окончание)*

Тип исключения	Возникает
EFOpenError	При обращении к файлу, например при попытке загрузить файл иллюстрации при помощи метода LoadFrcmFile. 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 (инструкция reset) несуществующий файл
EDBEngineError	При выполнении операций с базой данных, например при попытке выполнить SQL-запрос к несуществующей таблице

ПРИЛОЖЕНИЕ 2

Содержимое компакт-диска

Прилагаемый к книге компакт-диск содержит проекты (см. табл. П2.1), приведенные в книге в качестве примеров. Каждый проект находится в отдельном каталоге.

Помимо файлов проекта, в каждом каталоге находится выполняемый файл, что позволяет увидеть, как работает программа, без загрузки проекта в C++ Builder.

Большинство программ не требуют для своей работы никаких дополнительных программных компонентов (библиотек) и могут быть запущены непосредственно с CD-ROM.

Некоторые программы, например программы работы с базами данных, требуют, чтобы в системе был установлен процессор баз данных Borland Database Engine (устанавливается автоматически в процессе установки на компьютер C++ Builder) и зарегистрирован псевдоним соответствующей базы данных. Создать псевдоним базы данных можно при помощи SQL Explorer или BDE Administrator, которые также автоматически устанавливаются на компьютер в процессе установки C++ Builder.

Для активной работы с примерами, для того чтобы иметь возможность вносить в них изменения, необходимо скопировать каталоги проектов на жесткий диск компьютера.

Таблица П2.1. *Содержимое компакт-диска*

Каталог/Проект	Краткое описание	Глава
Сила тока/Amper	Программа вычисляет силу тока в электрической цепи. Демонстрирует обработку события Click, ввод из поля редактирования, вывод в поле метки, использование функций strToFloat и FloatToStrF	2

Таблица П2.1 *(продолжение)*

Каталог/Проект	Краткое описание	Глава
Сила тока/Amper1	Программа вычисляет силу тока в электрической цепи. Демонстрирует: обработку события KeyPress; исключения EZerroDevide (деление на ноль) при помощи инструкции try ... catch	2
Вывод текста/ TextOutDemo	Демонстрирует вывод текста на поверхность формы при помощи метода TextOutA, обработку событий OnPaint и OnFormResize	3
График функции/grf	Строит график функции. Демонстрирует: использование методов Line, MoveTo, TextOutA; обработку события OnFormResize	3
Просмотр иллюстраций/ ShowPic	Программа позволяет просматривать иллюстрации. Демонстрирует: использование компонента Image; использование функций FindFirst и FindNext; доступ к стандартному окну **Обзор папок**	3
Битовый образ/BitMap	Формирует картинку из битовых образов, загруженных из файла. Демонстрирует: загрузку и вывод битовых образов; влияние свойства Transparent	3
Фоновый рисунок/Back	Формирует фоновый рисунок путем многократного вывода битового образа на поверхность формы	3
Кораблик/Ship	Простая мультипликация (изображение формируется из графических примитивов). Демонстрирует использование методов, обеспечивающих вычерчивание графических примитивов; использование компонента Timer	3
Полет над городом /Flight	Мультипликация, элементы которой загружаются из bmp-файла. Демонстрирует вывод на поверхность формы иллюстраций, загруженных из файла	3
Полет над городом /Flight1	Мультипликация, элементы которой загружаются из ресурса программы. Демонстрирует загрузку битовых образов из ресурса программы	3
Баннер/Baner	Выводит на поверхность формы простой "мультик" — (баннер), все кадры которого находятся в одном bmp-файле. Демонстрирует использование метода CopyRect	3

Таблица П2.1 (продолжение)

Каталог/Проект	Краткое описание	Глава
Просмотр анимации/ ShowAVI	Программа позволяет просмотреть, в том числе и по кадрам, простую (не сопровождаемую звуком) анимацию (содержимое avi-файла). Демонстрирует использование компонентов Animate И FileOpen	4
Звуки Windows/ WinSound	Программа позволяет прослушать звуковые фрагменты, находящиеся в wav-файлах. Демонстрирует использование компонента MediaPlayer, использование функций FindFirst и FindNext для формирования списка файлов	4
CDPlayer/CDPlayer	Полнофункциональный проигрыватель CD-дисков. Демонстрирует использование компонента MediaPlayer	4
VideoPlayer	Простой видеоплеер. Демонстрирует использование компонента MediaPlayer для воспроизведения небольших видеороликов (формат AVI) и использование компонента SpeedButton	4
Ежедневник/org	База данных "Ежедневник". Комплексный пример. Демонстрирует использование компонентов Table, DataSource, Query, DBGrid И DBNavigator	5
Компонент	Компонент программиста, предназначенный для ввода целых или дробных (в зависимости от настройки компонента) чисел в заданном диапазоне. В каталоге находится модуль компонента (NkEdit.cpp), программа тестирования модуля компонента (tk.bpr) и программа тестирования компонента (Amper4.bpr). Замечание. Перед тем как открыть проект Amper4, необходимо установить компонент NkEdit в пакет dclusr	6
Консольное приложение/ Game	Игра "Угадай число". Пример консольного приложения	7
Справочная система	Справочная система для программы "Сапер". Каталог содержит файлы справочной информации и файлы проектов, необходимые для создания справочной системы в форматах HLP и CHM	8
Сапер	Игра "Сапер". Демонстрирует работу с графикой, использование компонента Menu, использование рекурсии, вывод справочной информации, запуск внешних программ	10

Таблица П2.1 *(окончание)*

Каталог/Проект	Краткое описание	Глава
Проверка знаний	Система тестирования. Демонстрирует использование компонентов, создаваемых во время работы программы	10
Очистка диска	Утилита "Очистка диска". Демонстрирует использование рекурсии	10

Рекомендуемая литература

1. Вирт Н. Алгоритмы и структуры данных / Пер. с англ. -- М.: Мир, 1989. - 360 с., ил.
2. Гринзоу Лу. Философия программирования для Windows 95/NT / Пер. с англ. — СПб.: Символ-Плюс, 1997. — 640 с., ил.
3. Зелковиц М., Шоу А., Гэннон Дж. Принципы разработки программного обеспечения / Пер. с англ. — М.: Мир, 1982. — 386 с., ил.
4. Культин Н. Б. С/С++ в примерах и задачах. -- СПб.: БХВ-Петербург, 2001. - 288 с., ил.
5. Практическое руководство по профаммированию / Пер. с англ. Б. Мик, П. Хит, Н. Рашби и др.; под ред. Б. Мика, П. Хит, Н. Рашби. -- М.: Радио и связь, 1986. — 168 с., ил.
6. Страуструп Б. Язык профаммирования С++. — М.: Радио и связь, 1991.
7. Фокс Дж. Профаммное обеспечение и его разработка / Пер. с англ. – М.: Мир, 1985. — 368 с., ил.

Предметный указатель

И

К

Л

М

О

П

Р

С

Продолжение рубрики см. на с. 320