ОЛЕГ ВАЛЬПА

bhv®

Borland C++ Builder

ЭКСПРЕСС-КУРС

- Простое и увлекательное изложение
- Детальное описание установки среды
- Множество иллюстраций и практических примеров
- Доступное объяснение специальных терминов

+видеокурс

Олег Вальпа

Borland C++ Builder

Экспресс-курс

Санкт-Петербург

«БХВ-Петербург»

2006

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
В16

Вальпа О. Д.

В16 Borland C++ Builder. Экспресс-курс. — СПб.: БХВ-Петербург, 2006. — 224 с.: ил.

ISBN 5-94157-855-5

Рассмотрены основы разработки приложений с использованием среды Borland C++ Builder, начиная с установки пакета и заканчивая отладкой приложений. Приведено большое количество практических примеров, позволяющих быстро научиться создавать собственные программы различного уровня сложности. Материал излагается просто и лаконично. Прилагаемый к книге компакт-диск содержит программы и проекты, описанные в книге, а также учебные видеоролики.

Для студентов и начинающих программистов

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Лапина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 21.02.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 18,06.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 5-94157-855-5

© Вальпа О. Д., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

От автора **1**

Введение **3**

ЧАСТЬ I. СРЕДА РАЗРАБОТКИ **5**

Глава 1. Установка **7**

Глава 2. Начало **26**

Глава 3. Интерфейс **30**

Глава 4. Первая программа **35**

Глава 5. Главное меню **37**

Группа команд *File* 37
Группа команд *Edit* 38
Группа команд *Search* 39
Группа команд *View* 39
Группа команд *Project* 40
Группа команд *Run* 40
Группа команд *Component* 41
Группа команд *Database* 41
Группа команд *Tools* 42
Группа команд *Windows* 42
Группа команд *Help* 42

Глава 6. Быстрые кнопки **44**

Кнопки *Standard* (Стандартные) 44
Кнопки *View* (Вид) 45
Кнопки *Debug* (Отладка) 46
Кнопки *Custom* (Заказ) 46

Глава 7. Палитра VCL-компонентов **47**

Глава 8. Дизайнер форм 50

Глава 9. Редактор кода 53

Глава 10. Инспектор объектов 58

Глава 11. Файлы проекта 63

Глава 12. Вывод на принтер 67

Глава 13. Отладка программ 70

Глава 14. Структура файлов 76

Глава 15. Инструменты отладчика 83

Глава 16. Окна отладчика 86

ЧАСТЬ II. ПРИМЕРЫ ПРОГРАММ 93

Глава 17. Цифровые часы — календарь 95

Глава 18. Визуальные компоненты 103

Компоненты *Standard* 105
Компоненты *Addition* 108
Компоненты *Win32* 109
Компоненты *System* 111
Компоненты *Dialogs* 112
Компоненты *Samples* 113
Компоненты *Office97* 114
Компоненты *Data Access* 115
Компоненты *Data Controls* 115
Компоненты *Internet Express* 115
Компоненты *Internet* 116
Итог 116

Глава 19. Программа ABC 117

Глава 20. Свойства компонентов 123

Свойство компонентов *Parent* 128
Расширенные свойства 128
Свойство *Color* 129
Применение компонента *MainMenu* 131

Применение компонента *Memo* ... 134
Применение компонента *ImageList* ... 136
Применение компонента *BitBtn* ... 137

Глава 21. Текстовый редактор ... 138

Глава 22. События компонентов ... 146

Событие *OnClick* ... 147
Событие *OnDblClick* ... 148
Событие *OnKeyDown* ... 149
Событие *OnKeyPress* ... 153
Событие *OnKeyUp* ... 153
Событие *OnEnter* ... 155
Событие *OnExit* ... 155
События *OnMouseDown* и *OnMouseUp* ... 155
Событие *OnMouseMove* ... 156
Событие *OnChange* ... 157
Событие *OnPaint* ... 157
Событие *OnProgress* ... 157
Событие *OnDragDrop* ... 158
Событие *OnDragOver* ... 158
Событие *OnStartDrag* ... 159
Событие *OnEndDrag* ... 159

Глава 23. Методы компонентов ... 160

Метод *Add* ... 166
Метод *Hide* ... 167
Метод *Show* ... 167
Метод *Delete* ... 167
Метод *CanFocus* ... 167
Метод *ChangeScale* ... 167
Метод *Ellipse* ... 169
Метод *Chord* ... 169
Метод *Pie* ... 170
Метод *Arc* ... 170
Метод *Rectangle* ... 170
Метод *RoundRect* ... 170
Метод *TextOut* ... 170
Метод *TextRect* ... 171
Метод *MoveTo* ... 171
Метод *LineTo* ... 171
Метод *FloodFill* ... 172
Метод *FillRect* ... 172
Пример программы *Function* ... 172

Глава 24. Программа *Square*.....178

Глава 25. Программа *MultiPlayer*.....183

Заключение.....195

ПРИЛОЖЕНИЯ.....199

Приложение 1. Горячие клавиши.....201

Приложение 2. Содержание компакт-диска.....203

Предметный указатель.....205

От автора

Я решил написать данную книгу, потому что сам прошел непростой путь обучения Borland C++ Builder и хотел бы значительно облегчить процесс освоения этого продукта новичками.

Сегодня я могу с уверенностью сказать, что процесс создания программ на Borland C++ Builder является не только простым и понятным, но самое главное — творческим, полезным и захватывающим занятием. Написав свою первую программу в этом пакете, вы, наверняка, не сможете остановиться и начнете творить. А что может быть прекраснее творчества?!

Я не претендую на звание программиста, так как в основном занимаюсь разработкой электронных устройств. Но поскольку в настоящее время электронная техника органично включает в себя микроконтроллеры и процессоры, которые необходимо программировать, а компьютеры переходят в разряд привычных инструментов, также требующих программирования, становится очевидным то, что программированием овладевать необходимо.

Эти знания позволят самостоятельно создавать тестовые программы, утилиты и несложные программно-инструментальные средства для отладки и тестирования схемотехнических устройств, а также различные электронные устройства с программной поддержкой.

В этой книге я попытался предельно просто описать среду разработки Borland C++ Builder 6 и объяснить процесс создания в ней программ, работающих в операционной системе Windows.

Эта книга позволит сделать очередной шаг к программированию. Она научит вас не только основной терминологии современного программирования, но и послужит практическим пособием при разработке собственных программ.

Введение

Данная книга не претендует на полное и детальное описание программного продукта Borland C++ Builder 6, но благодаря своей компактности она позволяет, не боясь большого объема материала для изучения, осваивать этот пакет и очень быстро начать писать свои программы.

Работать в среде Borland C++ Builder 6 может любой пользователь компьютера, имеющий хотя бы минимальные знания о программировании, например, из программы школьного курса. Постепенно этот человек может увлечься и стать настоящим программистом.

В своем изложении я хочу, не отягощая избыточностью материала, зажечь и увлечь в этот удивительный и прекрасный мир программирования новых, еще не познавших свой путь, людей. Хочу дать первый импульс к творчеству и ключ к простому пониманию замысловатых специальных понятий и терминов.

Программы разрабатываются в среде Borland C++ Builder 6 настолько просто и легко, что после первых упражнений с этим продуктом, с ним уже не хочется расставаться никогда.

Мы вместе с вами не только изучим основные инструменты пакета Borland C++ Builder 6, но и напишем свои первые программы с поразительной быстротой и легкостью. И я думаю, после этого вы сами начнете читать и изучать больше и больше об этом удивительном продукте, открывая путь к своей вершине совершенства.

И самое главное — все, что вы прочтете здесь, необходимо закреплять собственной практикой при работе за компьютером. Только так, с помощью практики, можно добиться хороших и прочных знаний в любом полезном деле. Не пытайтесь усвоить весь материал за один присест. Не ленитесь перечитать то, что не поняли или недопоняли, или забыли. Помните, "повторение — мать учения".

Новые понятия обычно усваиваются только после третьего прочтения. Придерживайтесь этих правил — и обязательно добьетесь успеха. Поверьте в себя!

Далее в тексте я буду использовать некоторые специальные термины, поясняя по ходу изложения материала, что они означают. Эти термины являются общеупотребительными для программистов, и к ним необходимо привыкнуть. В дальнейшем это очень поможет в изучении и освоении новых программных продуктов и специальной технической литературы.

Одним из первых таких терминов является *объектно-ориентированное программирование* (ООП). Данный термин означает, что при разработке программ используются объекты в виде окон программы, программных кнопок, групп флажков и т. п., знакомых пользователям компьютеров по программам, работающим в Windows. Данный метод разработки упрощает создание программы и не требует от программиста детально продумывать весь алгоритм работы программы от начала и до конца, как это делалось в *структурном* программировании. Разработку программы, используя этот метод, можно разбить на отдельные этапы, а затем соединить их вместе для получения конечной программы. Именно такой метод разработки имеет Borland C++ Builder 6 и ему подобные пакеты для создания программ под Windows.

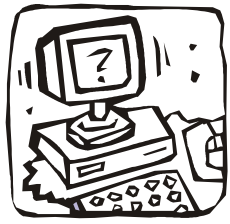
ООП дает возможность писать программу постепенно, шаг за шагом, создавая отдельные небольшие программы *(функции-методы)* для обработки действий *(событий)*, вызываемых *объектами* (кнопками и окнами программы, кнопками клавиатуры и т. п.). Скоро вы привыкнете к этим терминам и легко будете манипулировать ими, поверьте мне на слово.

Часть I

СРЕДА РАЗРАБОТКИ

В *части I* книги рассказывается об установке среды разработки Borland C++ Builder 6 и дается ее краткое описание. Читатель может сразу начать практическое знакомство со средой, производя ее установку и повторяя примеры программ, описанные в части I. Далее в этой части более детально описываются интерфейс среды разработки и средства для устранения ошибок и отладки программ.

Автор этой книги настоятельно рекомендует читать ее параллельно с практической работой на компьютере. Если читатель будет следовать рекомендациям автора, то уже через несколько часов он сможет почувствовать уверенность в своих силах и обретет навыки работы со средой разработки.



Глава 1

Установка

В этой главе говорится о том, как установить среду разработки Borland C++ Builder 6 на компьютер. В ней шаг за шагом описана вся последовательность установки и возникающие при этом сообщения и запросы. Если вы уже установили Borland C++ Builder 6 на свой компьютер или не нуждаетесь в такой помощи, можете пропустить эту главу и приступить к чтению следующей.

Несмотря на то, что установка какого-либо программного продукта на компьютер является довольно простой процедурой, многие начинающие пользователи не могут самостоятельно с ней справиться и прекращают изучение программы. Проблема установки состоит скорее не в сложности операций, сколько в непонимании вопросов, задаваемых программой инсталляции пользователю. Для того чтобы помочь начинающим пользователям в установке программного пакета Borland C++ Builder 6, процесс инсталляции излагается здесь в подробном варианте. В связи с этим рассматриваются всевозможные ситуации, которые могут и не возникнуть у большинства пользователей. Поэтому некоторые окна, рассматриваемые здесь, могут не появиться на экране компьютера при установке пакета Borland C++ Builder 6. Тем не менее лучше знать об их существовании и действиях, которые необходимо выполнить при появлении таких окон, чтобы не испытывать дополнительных трудностей.

Для установки Borland C++ Builder 6 на персональный компьютер не требуется особых знаний. Здесь подробно описывается весь процесс установки, шаг за шагом, с объяснением возникающих при этом сообщений. Желательно, чтобы компьютер, на который будет производиться установка пакета, имел процессор типа Pentium или Celeron с тактовой частотой не ниже 166 МГц, объем памяти компьютера был не менее 128 Мбайт, и на диске было свободное пространство объемом не менее 750 Мбайт. Естественно на компьютере уже должна быть установлена операционная система Windows 95/98/ME/NT/2000 или XP, и он должен иметь накопитель CD-ROM. После включения компьютера и завершения его загрузки установите

в привод CD-ROM компакт-диск, содержащий *инсталляционный* (установочный) пакет Borland C++ Builder 6. Если на компьютере разрешена автозагрузка CD-ROM, то через несколько секунд после установки в CD-ROM диска будет произведен автозапуск программы инсталляции или программной оболочки компакт-диска, в которой можно выбрать установку пакета. Если автозапуска не произошло, необходимо с помощью Проводника открыть содержимое компакт-диска и запустить в нем программу установки с именем setup, установив курсор мыши на имени этого файла и дважды щелкнув после этого левой кнопкой мыши.

Иногда установочный пакет хранится на компакт-диске в упакованном виде, для того чтобы разместиться на ограниченном объеме диска вместе с другими программами. При запуске программы установки с таких дисков вначале производится автоматическая распаковка установочного пакета на жесткий диск компьютера для последующего запуска установки среды разработки. Распаковка начинается с отображения на экране компьютера окна, подобного тому, что приведено на рис. 1.1.

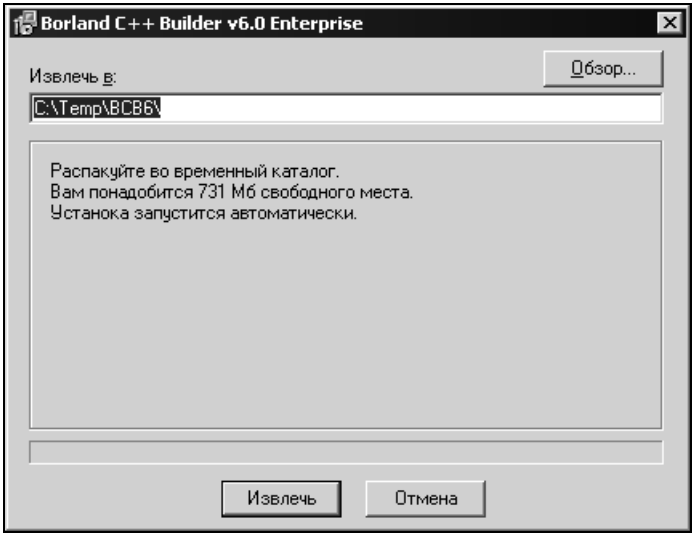


Рис. 1.1. Окно сообщения о подготовке к распаковке файлов установочного пакета

При выполнении такой процедуры необходимо указать путь для распаковки установочного пакета на тот раздел диска, где имеется свободное дисковое пространство с объемом не менее 731 Мбайт. При этом не забудьте, что для последующей установки пакета Borland C++ Builder 6 понадобится еще 750 Мбайт свободного пространства. Таким образом, при распаковке установочного пакета на диск C: нужно будет иметь на нем не менее 1481 Мбайт свободного дискового пространства. Либо укажите для распаковки инстал-

ляционного пакета другой раздел диска, например D или E, если такие разделы существуют на вашем компьютере и имеют необходимый объем свободного дискового пространства.

Процесс распаковки файлов из архива отображается в окне, подобном представленному на рис. 1.2.

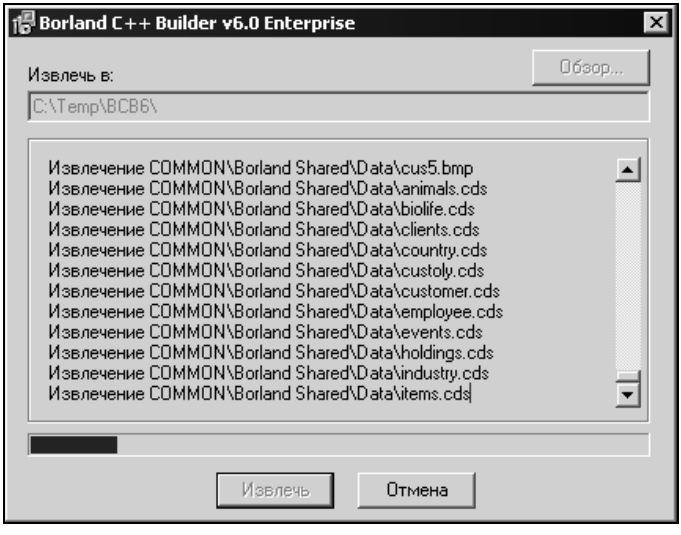


Рис. 1.2. Окно процесса распаковки файлов установочного пакета

После завершения установки пакета Borland C++ Builder 6 все распакованные из архива файлы можно будет удалить для освобождения пространства на диске.

Иногда компакт-диск содержит файлы помощи с названием типа readme.txt. В этих файлах содержится полезная справочная информация, помогающая установить необходимый программный пакет на компьютер, избегая при этом ошибок и напрасной потери времени. Прочтите содержимое этих файлов перед началом установки. Это избавит вас от ошибок и ускорит установку пакета.

Итак, после распаковки архива происходит автоматический запуск установки пакета Borland C++ Builder 6 на компьютер. Если этого не происходит, отыщите среди распакованных файлов файл с именем Borland C++ Builder 6.msi и дважды щелкните по нему левой кнопкой мыши.

При этом начнется установка пакета, сопровождающаяся отображением на экране окон с информацией, описывающей ход установки и предлагающей пользователю ввести некоторые данные или сделать определенный выбор.

Вначале появляется окно мастера установки **InstallShield Wizard**, показанное на рис. 1.3. При этом программа установки производит проверку типа установленной на компьютер операционной системы Windows.

Далее появится окно **Windows Installer** (рис. 1.4) с сообщением о подготовке к установке.

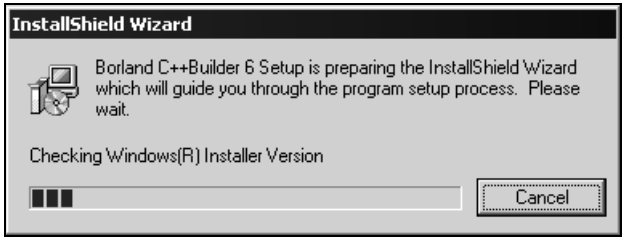


Рис. 1.3. Окно мастера установки **InstallShield Wizard**

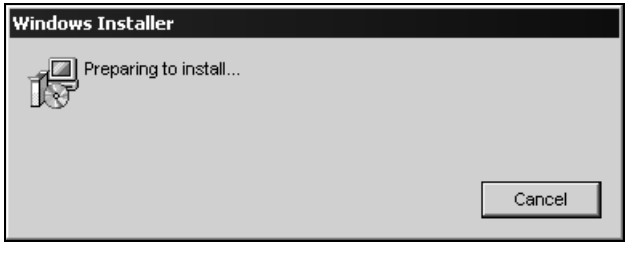


Рис. 1.4. Окно **Windows Installer**

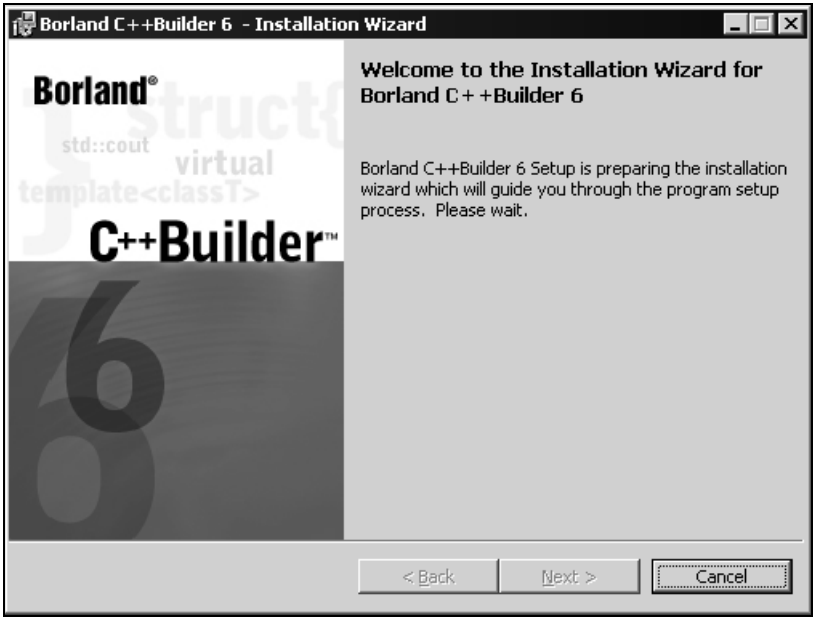


Рис. 1.5. Окно сообщения о подготовке мастера установки

После чего на экране отобразится окно, приведенное на рис. 1.5, в котором сообщается о том, что производится подготовка мастера установки, который проведет пользователя через всю установку пакета.

Это окно быстро сменяется на окно (рис. 1.6) с сообщением о том, что мастер установки готов к установке пакета Borland C++ Builder 6 на компьютер и для продолжения необходимо нажать кнопку **Next**. Здесь же можно прекратить установку, нажав кнопку **Cancel**. Если по какой-либо причине потребуется прекратить установку, это можно будет сделать и позже, на следующих этапах установки.

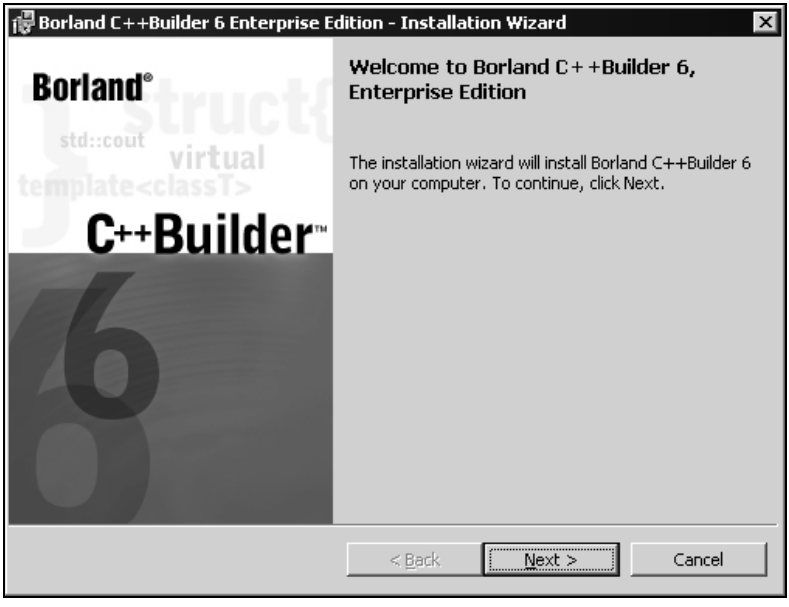


Рис. 1.6. Окно сообщения о готовности мастера установки

После нажатия кнопки **Next** будет отображено окно (рис. 1.7), в котором предлагается ввести серийный номер и код авторизации пакета. Эти номера находятся на установочном компакт-диске, и их следует безошибочно ввести в соответствующие поля отображенного окна. После чего нужно вновь нажать кнопку **Next**.

Далее будет выведено окно с лицензионным соглашением (рис. 1.8), в котором говорится о правах и обязанностях пользователя пакета. Ознакомившись с текстом соглашения, установите флажок в поле **I accept the terms in the license agreement**, принимая тем самым условия лицензионного соглашения.

Следующее окно (рис. 1.9) содержит дополнительную информацию по установке пакета. Эта информация предназначена в первую очередь для подготовленных специалистов и профессиональных программистов. Для начала работы с пакетом ее изучение не обязательно.

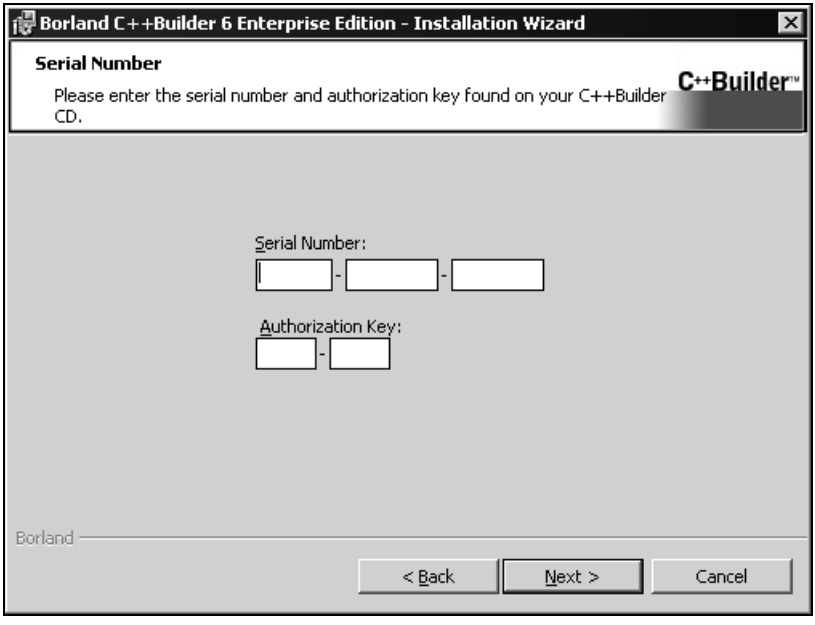


Рис. 1.7. Окно ввода серийного номера и кода активизации

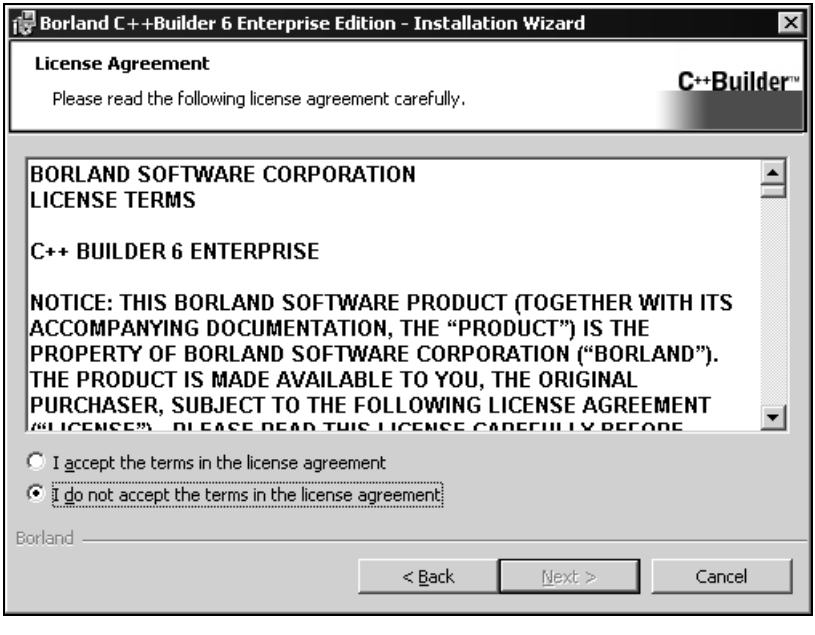


Рис. 1.8. Окно с лицензионным соглашением

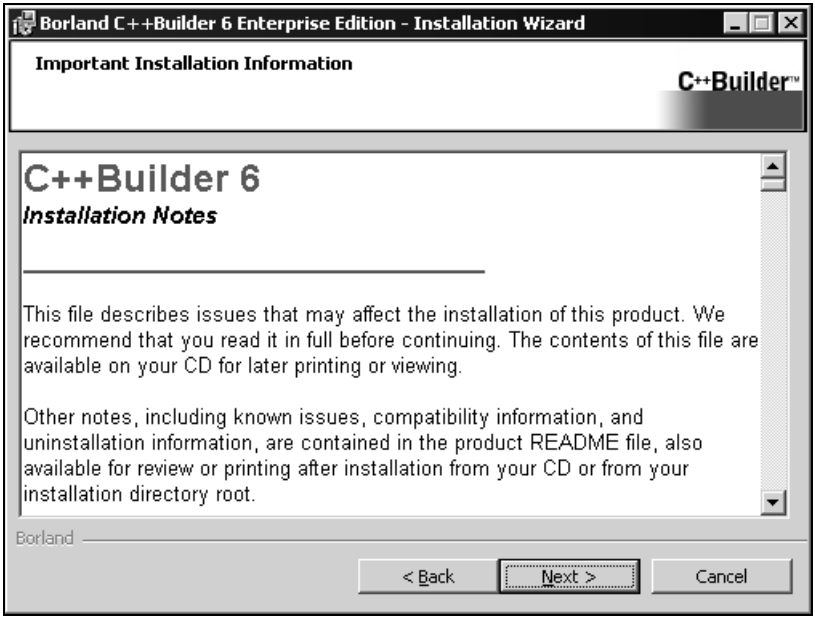


Рис. 1.9. Окно с дополнительной информацией по установке пакета

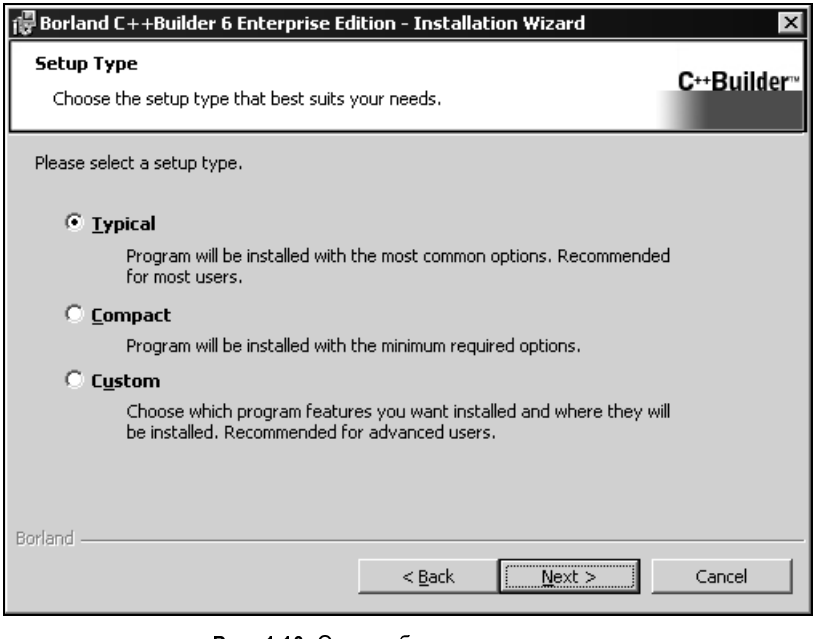


Рис. 1.10. Окно выбора типа установки

Нажав кнопку **Next**, можно продолжить установку и увидеть новое окно (рис. 1.10), в котором предлагается выбрать тип установки среды разработки.

Здесь предлагается сделать выбор между типовой (**Typical**), компактной (**Compact**) и выборочной (**Custom**) установкой пакета. Оставьте флажок установленным напротив типовой установки и нажмите кнопку **Next**. В дальнейшем вы сможете попробовать и другие варианты установки пакета, если это потребуется.

Некоторые окна, описываемые далее по тексту, могут не появляться при установке программного пакета. Их отображение на экране происходит только в том случае, если до установки Borland C++ Builder 6 были установлены другие среды разработки, офисные программы или базы данных. Кроме того, содержимое отображаемых окон также может незначительно отличаться от содержимого, приводимого в этой книге. Бояться этого не стоит, так как смысл сообщений не будет сильно отличаться, и при любом ответе установка пакета будет завершена нормально.

Если до установки пакета Borland C++ Builder 6 на компьютере была установлена другая среда разработки, например Microsoft Visual Studio, то на экране отобразится окно (рис. 1.11), в котором сообщается о том, что программа установки обнаружила на данном компьютере зарегистрированный отладчик, который будет обновлен при нажатии кнопки **Да**. Здесь пользователь вправе сам сделать выбор того отладчика программ, с которым будет работать в дальнейшем. Если ничего изменять не требуется, нужно просто нажать кнопку **Нет**. При любом выборе продолжение установки будет нормальным.

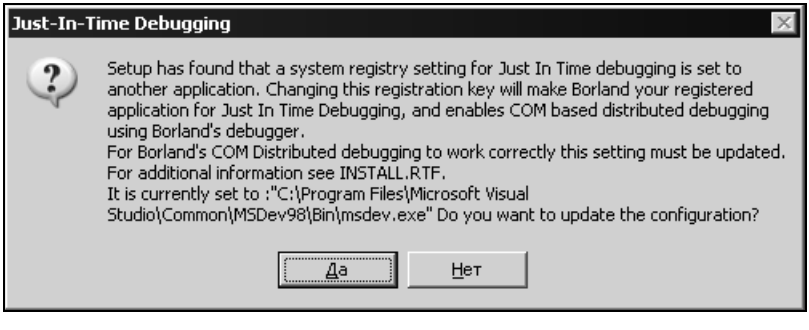


Рис. 1.11. Окно с сообщением об обнаружении отладчика

Далее установщик выведет окно (рис. 1.12) с запросом на установку компонента CORBA. Можно отключить этот флажок для облегчения установки пакета. В дальнейшем вы всегда сможете установить некоторые компоненты пакета по мере накопления знаний и необходимости в этих дополнительных компонентах.

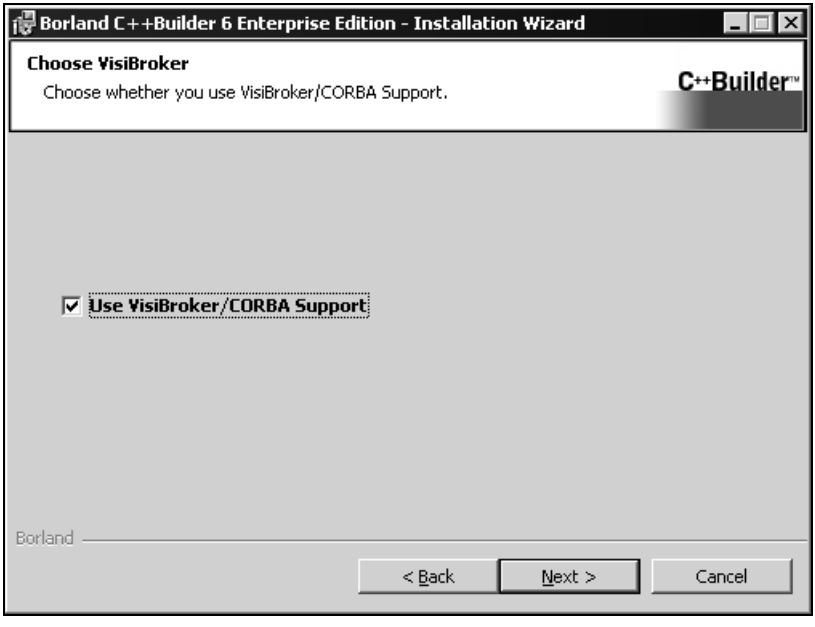


Рис. 1.12. Окно запроса на установку компонента CORBA

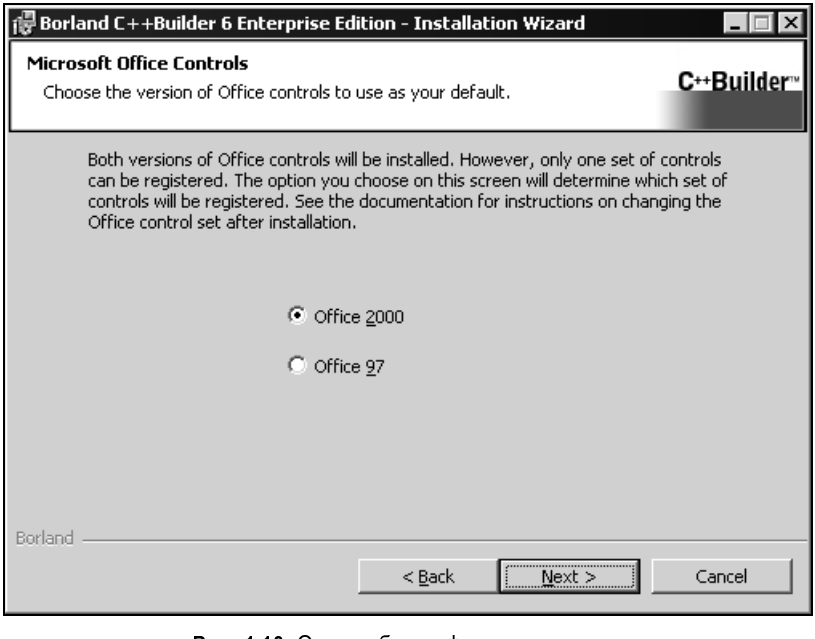


Рис. 1.13. Окно выбора офисного продукта

Очередное окно выбора офисного продукта Office 2000 или Office 97 (рис. 1.13) предлагает пользователю указать, какая из этих версий будет использоваться по умолчанию. Здесь необходимо установить флажок напротив того продукта, который установлен на вашем компьютере.

Если программа инсталляции обнаружит, что на компьютере имеется ассоциация файлов языка программирования C++ с расширением cpp какой-либо другой программе, она предложит изменить ее на ассоциацию для Borland C++ Builder 6. При этом будет отображено окно, показанное на рис. 1.14.

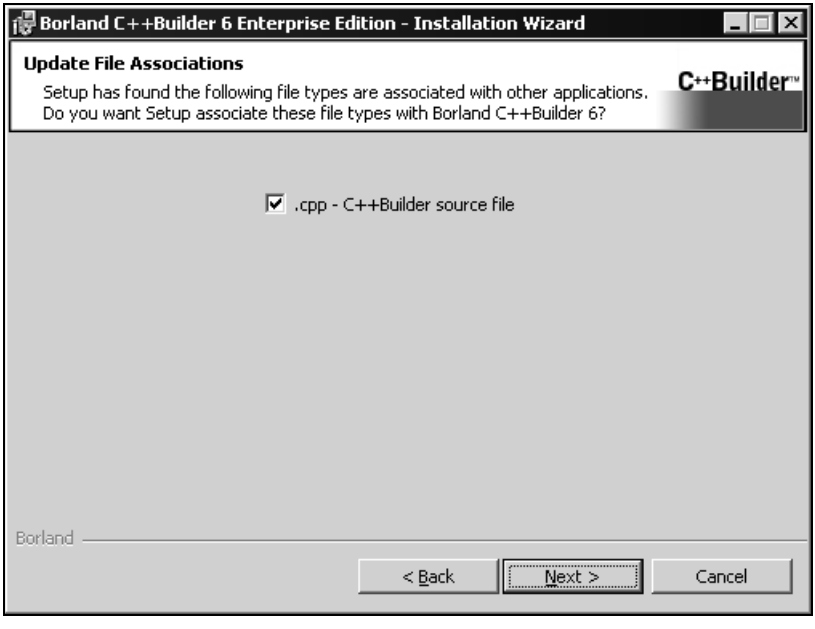


Рис. 1.14. Окно запроса на изменение ассоциации файлов

Ассоциацией называется свойство операционной системы при двойном щелчке по имени файла с некоторым расширением вызывать определенную заранее программу для обработки данного файла.

На следующем этапе инсталляции будет выведено окно (рис. 1.15) с запросом об установке дополнительных компонентов.

В этом окне сообщается о том, что пользователь может установить программу базы данных InterBase Client в настоящее время или выполнить ее позже с компакт-диска.

Здесь флажок установки базы данных можно оставить установленным и нажать кнопку **Next**.

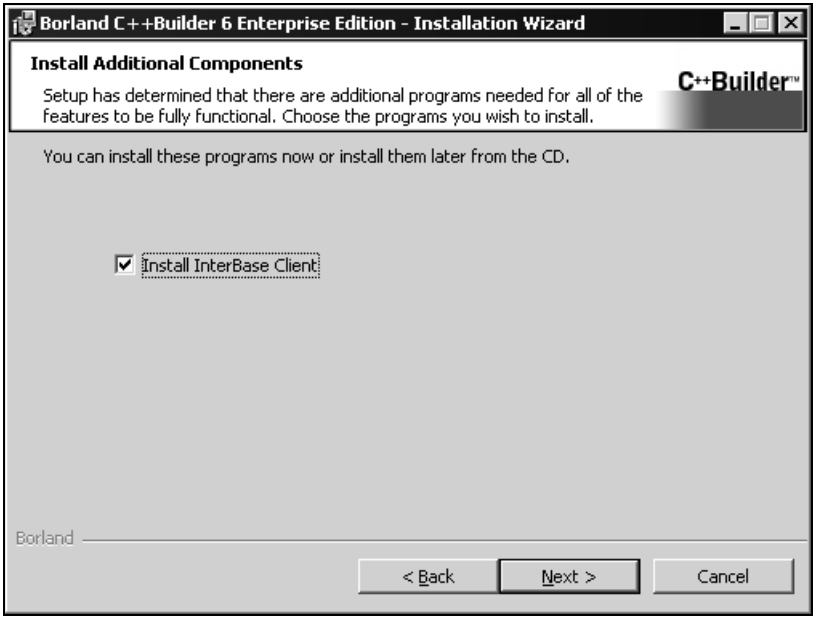


Рис. 1.15. Окно запроса об установке дополнительных компонентов

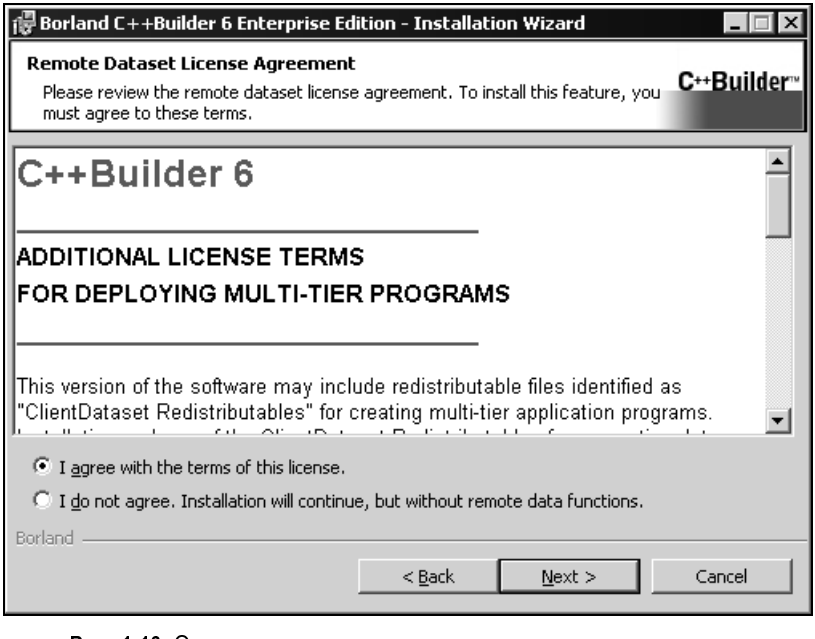


Рис. 1.16. Окно с дополнительными лицензионными соглашениями

Далее отобразится окно (рис. 1.16), в котором предлагается ознакомиться с дополнительными лицензионными соглашениями и согласиться с ними, установив флажок в поле **I agree with the terms of this license**. После этого следует продолжить установку, нажав кнопку **Next**.

Следующее отображаемое окно (рис. 1.17) предлагает нажать кнопку **Next** для установки пакета в указанные каталоги или кнопку **Change** — для изменения соответствующего каталога. Здесь можно согласиться с установкой в указанные каталоги и нажать кнопку **Next**.

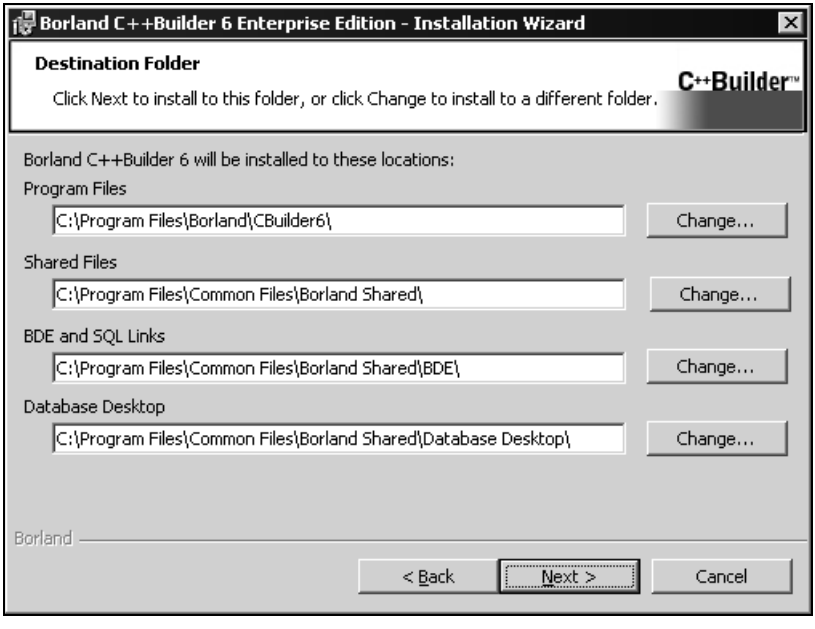


Рис. 1.17. Окно назначения каталогов для установки пакета

Если на диске будет недостаточно места для установки пакета, на экране появится окно, представленное на рис. 1.18. В нем сообщается о недостатке места, о размерах занятого и свободного пространства и предлагается установить сокращенную версию пакета или сменить диск, на который будет произведена установка. В этом окне следует нажать кнопку **OK** и при необходимости освободить место на диске или сменить диск для установки.

Следующее окно (рис. 1.19) предлагает выбрать из списка те программы, которые необходимо установить. Здесь можно согласиться с установкой всех программ и нажать кнопку **Next**.

Далее появится окно, позволяющее изменить конфигурацию типа используемых баз данных (рис. 1.20). Здесь можно, ничего не меняя, нажать кнопку **Next** для продолжения установки.

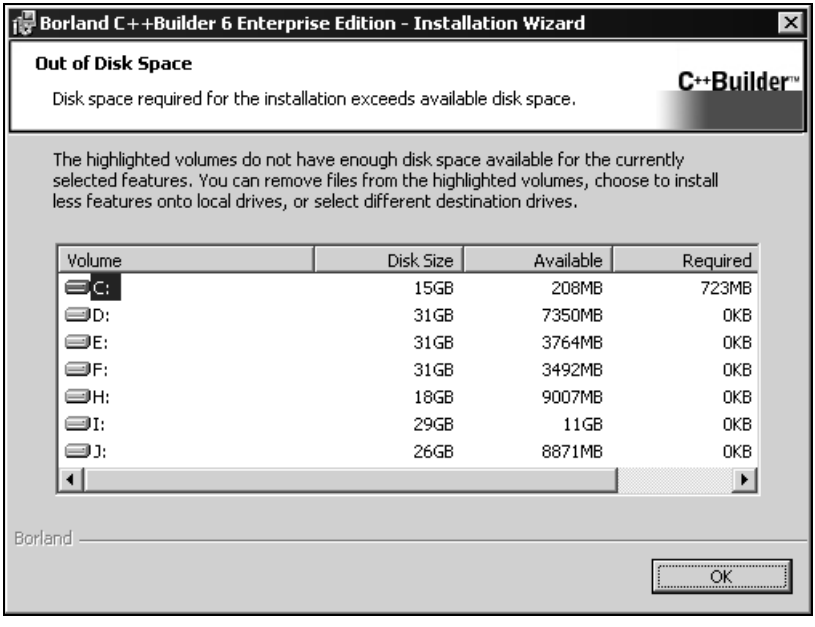


Рис. 1.18. Окно информации о дисковом пространстве

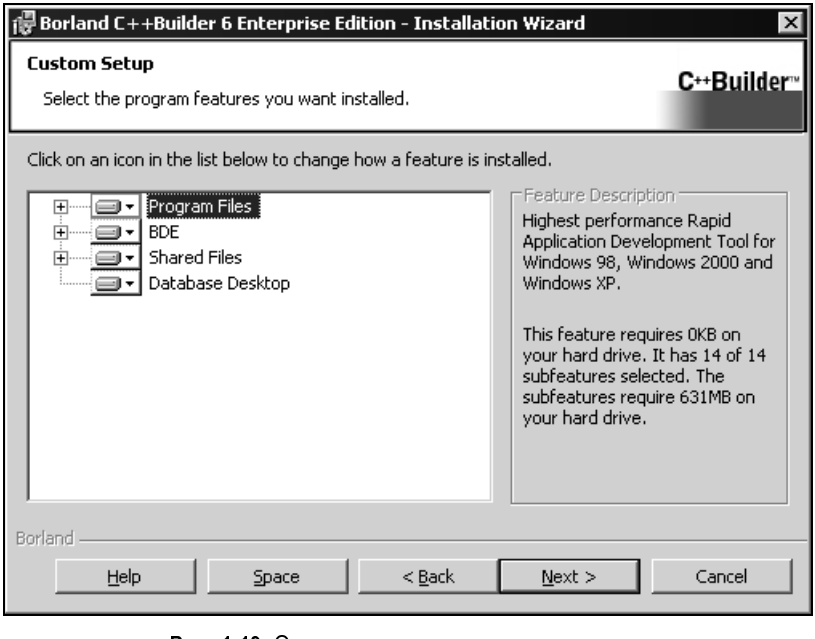


Рис. 1.19. Окно со списком программ установки

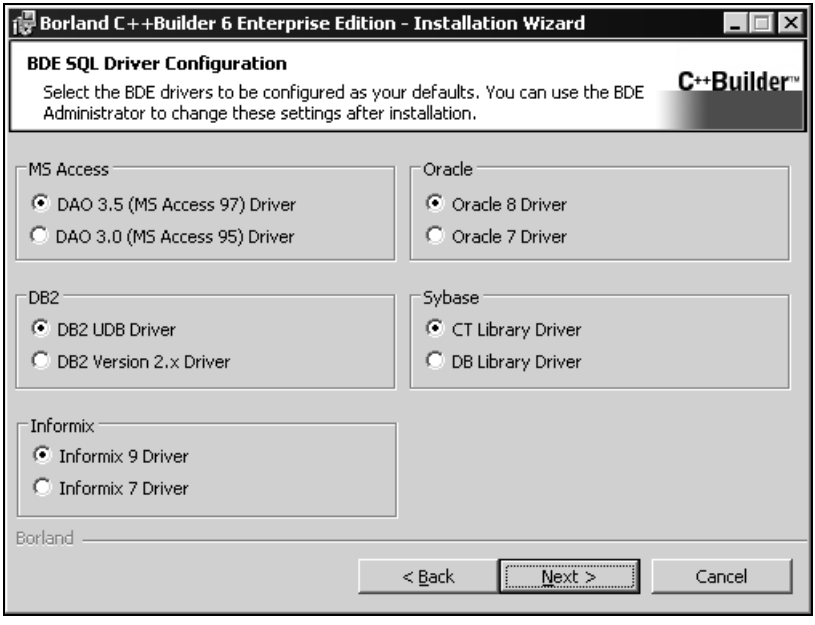


Рис. 1.20. Окно конфигурации типа баз данных

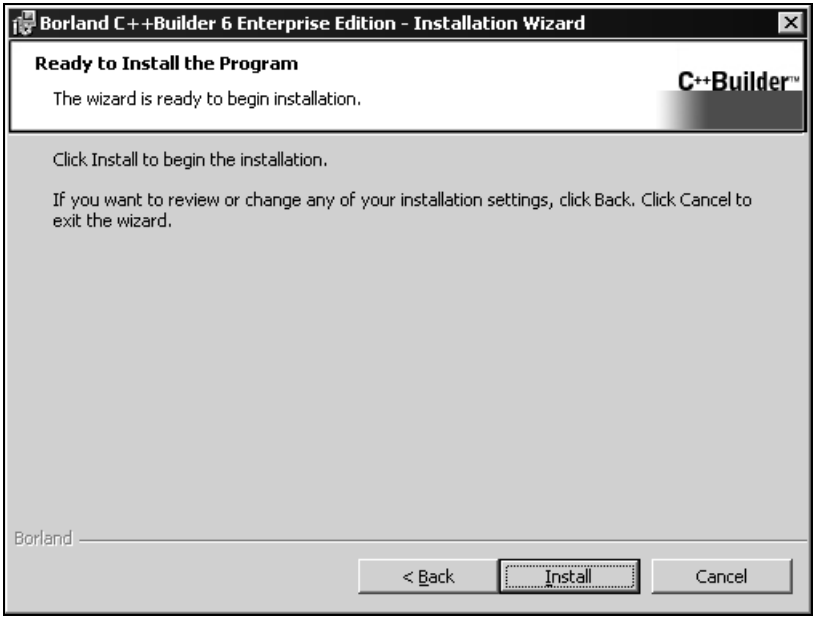


Рис. 1.21. Окно сообщения о готовности к завершающей стадии установки

Наконец программа инсталляции выведет окно (рис. 1.21), сообщающее о готовности к завершающей стадии установки, связанной с записью файлов на диск.

После нажатия в этом окне кнопки **Install** программа установки отобразит окно (рис. 1.22) с сообщением о том, что начался процесс инсталляции. В нижней части этого окна будет отображаться состояние процесса (status) и шкала выполнения процесса установки.

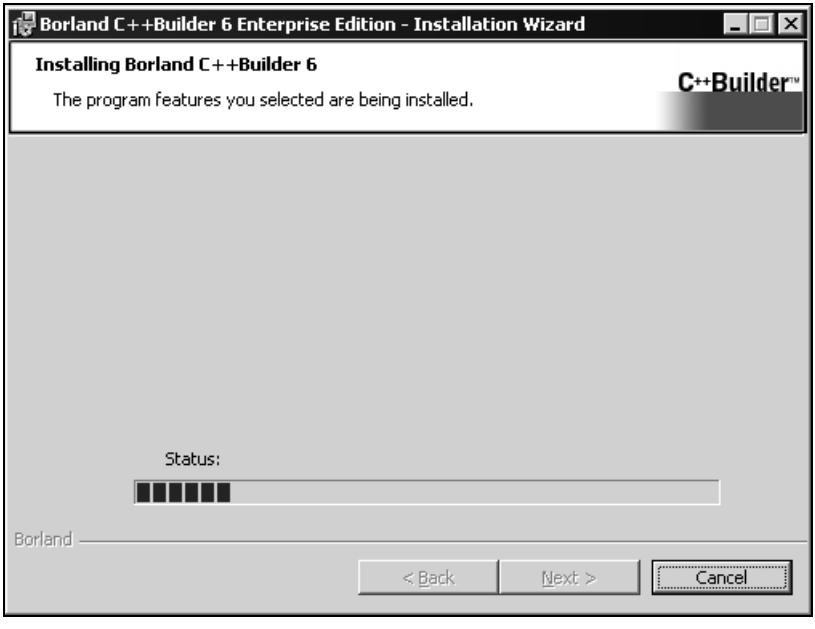


Рис. 1.22. Окно с сообщением о начале процесса инсталляции

Это окно в процессе установки будет сменяться поочередно окнами, показанными на рис. 1.23—1.27, в которых сообщается об устанавливаемых компонентах и этапах установки.

Наконец через несколько минут появится окно (рис. 1.28), в котором будет предложено вставить второй компакт-диск, на котором размещается программа InterBase.

Нажмите в нем кнопку отмены. После чего появится окно (рис. 1.29) с сообщением о том, что программу InterBase можно будет установить позже.

На этом работа программы установки среды разработки завершается, и на экран будет выведено окно (рис. 1.30) с предложением нажать кнопку **Finish** для выхода из мастера установки.

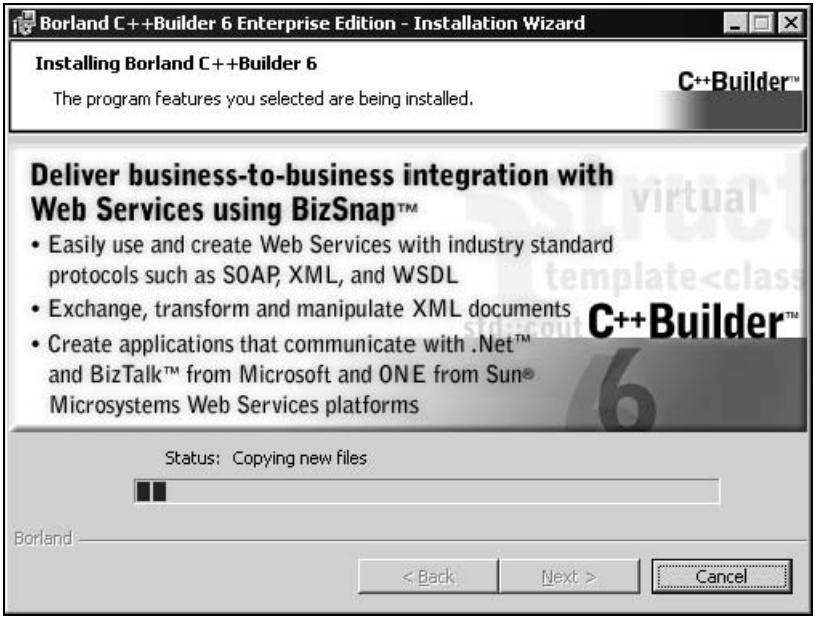


Рис. 1.23. Первое окно с сообщением об устанавливаемых компонентах

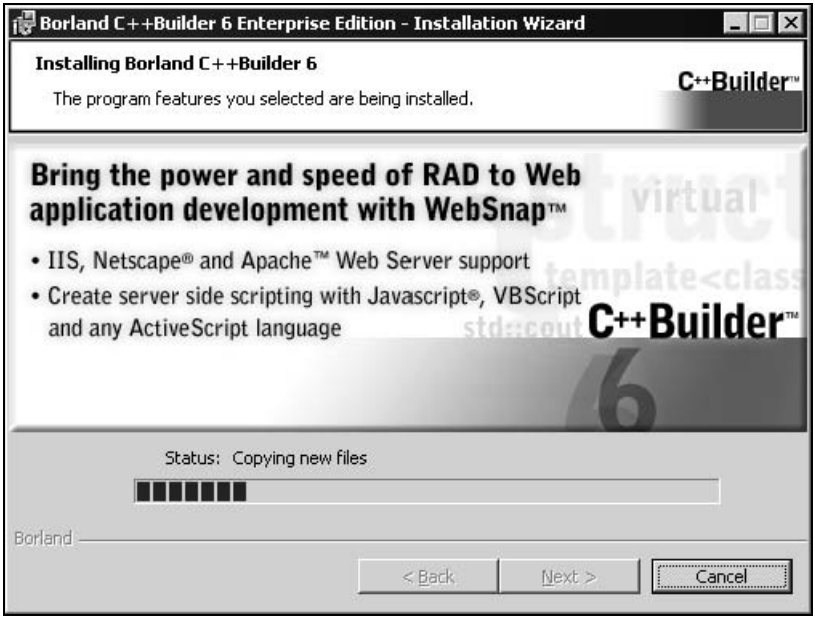


Рис. 1.24. Второе окно с сообщением об устанавливаемых компонентах

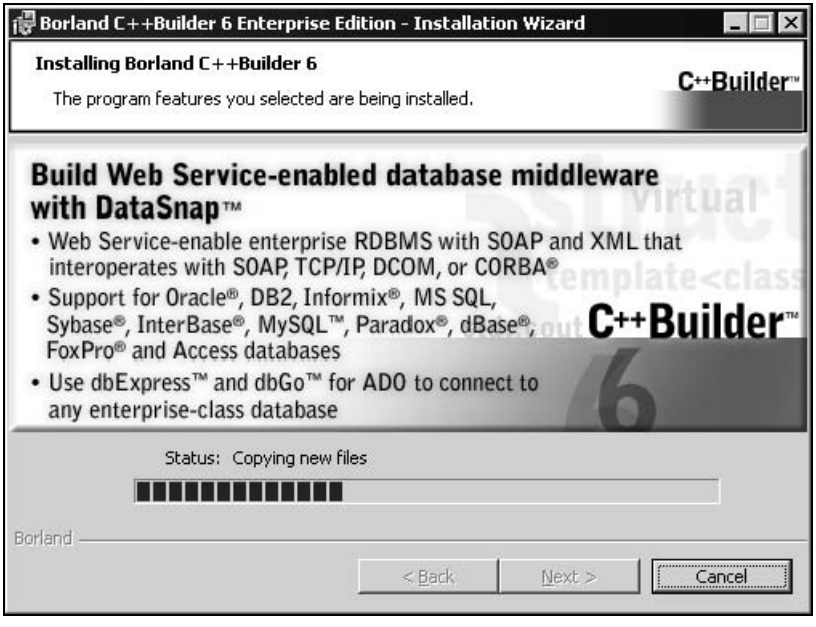


Рис. 1.25. Третье окно с сообщением об устанавливаемых компонентах

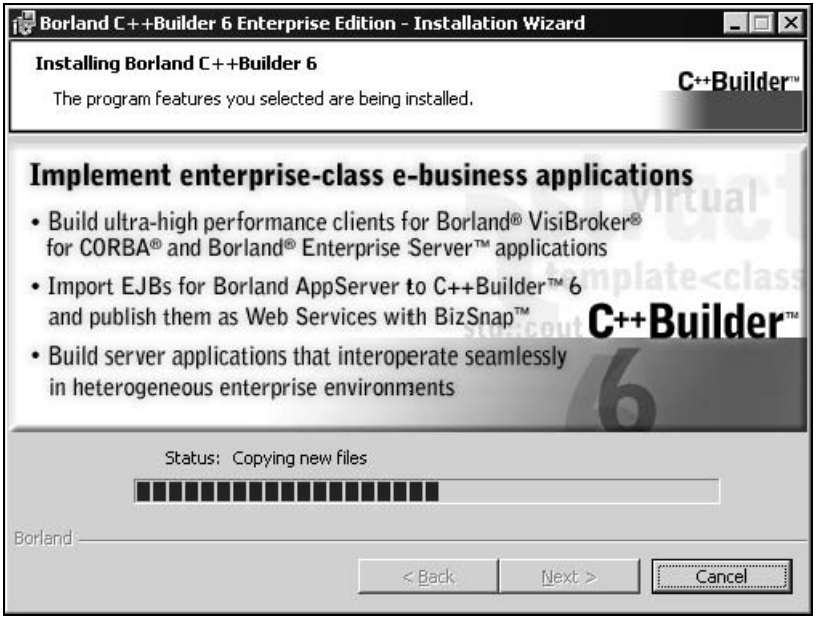


Рис. 1.26. Четвертое окно с сообщением об устанавливаемых компонентах

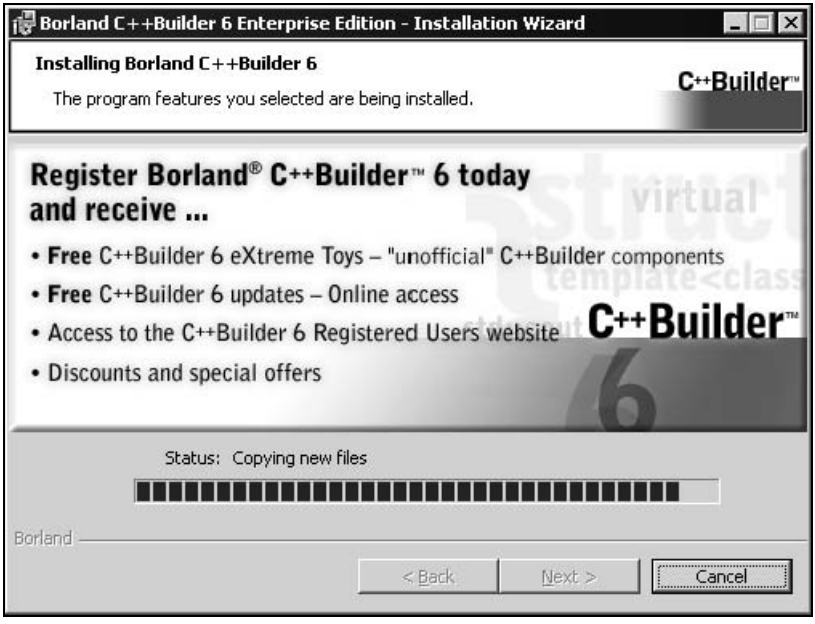


Рис. 1.27. Пятое окно с сообщением об устанавливаемых компонентах

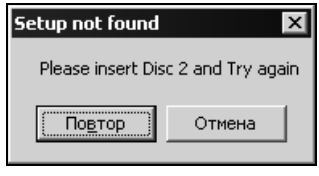


Рис. 1.28. Окно с предложением установки второго компакт-диска

Рис. 1.29. Окно сообщения о возможности установки программы InterBase

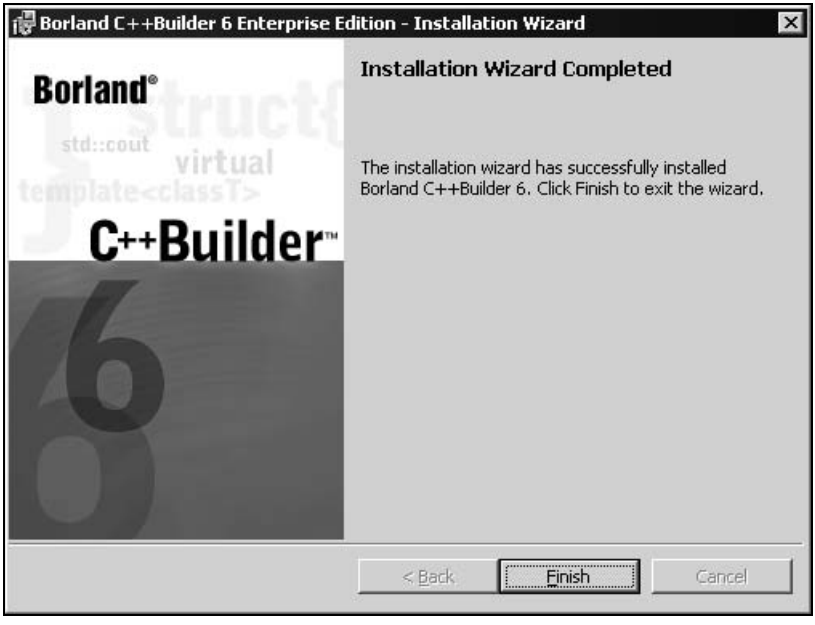


Рис. 1.30. Окно завершения установки

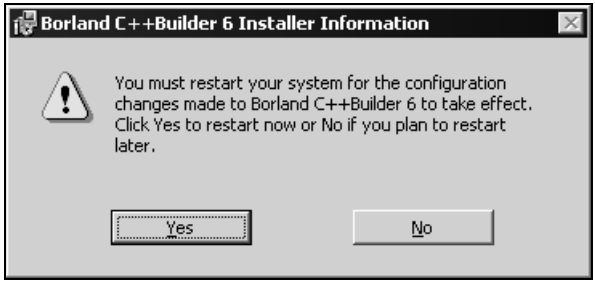
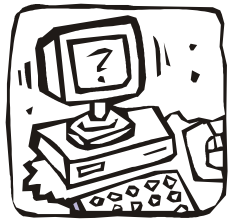


Рис. 1.31. Окно с предложением выполнить перезагрузку компьютера

После нажатия кнопки **Finish** будет выведено окно (рис. 1.31), в котором предлагается выполнить перезагрузку компьютера для того, чтобы установленная программа могла полноценно функционировать.

Нажмите в этом окне кнопку **Yes** и дождитесь перезагрузки компьютера.



Глава 2

Начало

В данной главе говорится о первом запуске программы Borland C++ Builder 6 и необходимой при этом регистрации программы. Кроме того, даются советы по быстрому запуску среды разработки.

После установки Borland C++ Builder 6 и перезагрузки компьютера, в меню программ компьютера появится новый раздел под названием "Borland C++ Builder 6", в который входит множество программ и справочных файлов на английском языке (рис. 2.1).

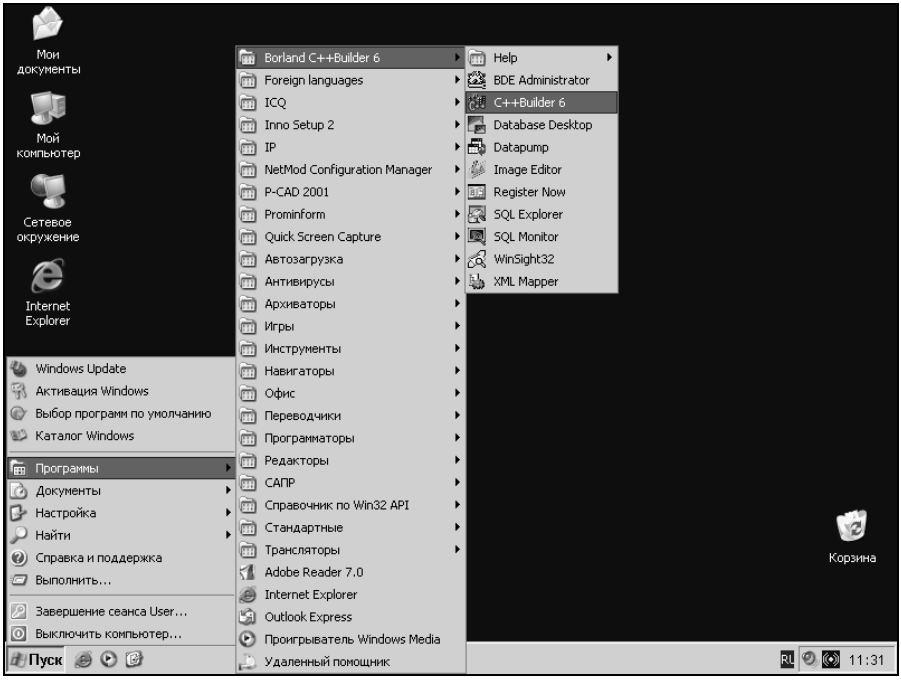


Рис. 2.1. Меню программ

Со временем эти программы станут понятными и привычными инструментами для вас. Поэтому не пугайтесь их изобилия и замысловатости названий.

Для начала работы в среде Borland C++ Builder 6 необходимо выполнить команду **Пуск | Программы | Borland C++ Builder 6 | C++ Builder**. При этом начнет работать главная программа, которая хранится в файле с названием *bcb.exe*, располагающемся в каталоге C:\Program Files\Borland\CBuilder6\Bin. После запуска этой программы вы сможете вызывать остальные программы среды разработки по мере необходимости.

При первом запуске среды разработки на экране отобразится окно, в котором предлагается выбрать способ регистрации программы (рис. 2.2).

Рис. 2.2. Окно выбора способа регистрации

В верхнем поле окна предлагается регистрация через Интернет, в среднем поле — с помощью телефона или через браузер, а в нижнем поле — выполнить регистрацию позже. Установите флажок в среднем поле этого окна и нажмите кнопку **Next**.

В следующем окне (рис. 2.3) будут отображены введенные ранее серийный номер и ключ авторизации. Кроме того, будет показан код регистрации. Эти данные необходимо будет передать по телефону, приведенному в верхней части окна, или ввести в регистрационную форму, которая находится

в Интернете, по ссылке, приведенной ниже номера телефона. Второй способ более удобен, поскольку быстрее, дешевле и не требует общения с представителями компании на английском языке. Кроме того, во втором случае проще выполнить регистрацию без ошибок. Целью этой регистрации является получение ключа активации программы. Этот ключ будет выслан на электронный адрес пользователя, зарегистрировавшего программу. Введите полученный код в поле **Activation Key** и нажмите кнопку **Next**.

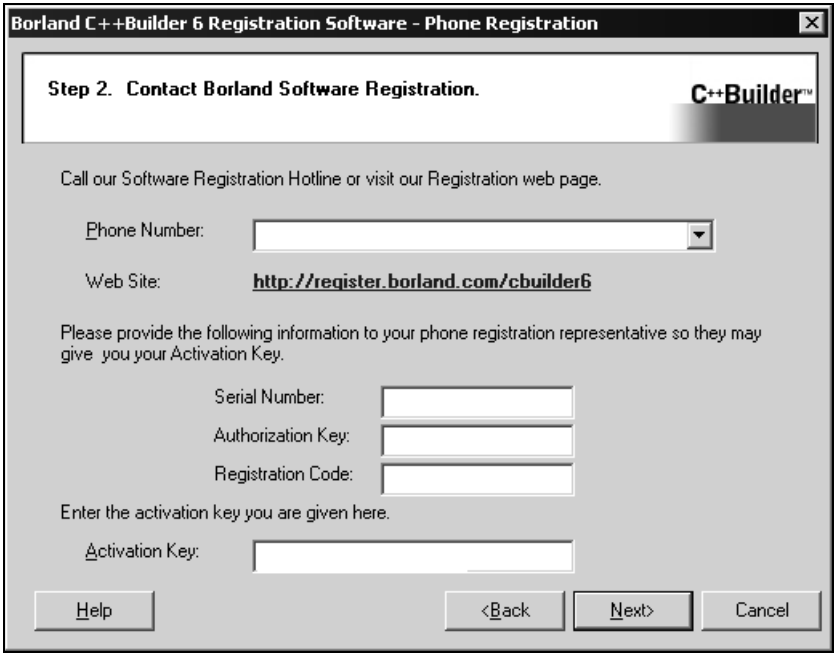


Рис. 2.3. Окно регистрации

После этого отобразится окно поздравления (рис. 2.4), в котором сообщается, что программный продукт был успешно зарегистрирован и пользователю предлагается сохранить регистрационную информацию для будущих потребностей.

Нажмите в этом окне кнопку **Exit** для завершения регистрации.

Последующие запуски программы среды разработки не потребуют повторной регистрации и будут приводить к немедленному ее запуску.

Советую поместить *ярлык* (значок) главной программы Borland C++ Builder 6 на *рабочий стол* (экран) компьютера для последующего быстрого и простого поиска и запуска среды разработки. Для этого необходимо свернуть все окна запущенных на компьютере программ или закрыть их. После чего необходимо найти через путь **Пуск | Программы | Borland C++ Builder** имя про-

граммы **C++ Builder 6**, нажать на ней левую кнопку мыши и, удерживая нажатыми ее и клавишу <Ctrl> (для копирования), перенести ярлык программы на свободное место рабочего стола. После этого можно запускать среду разработки двойным щелчком левой кнопки мыши по значку **C++ Builder 6**, помещенному на рабочем столе.

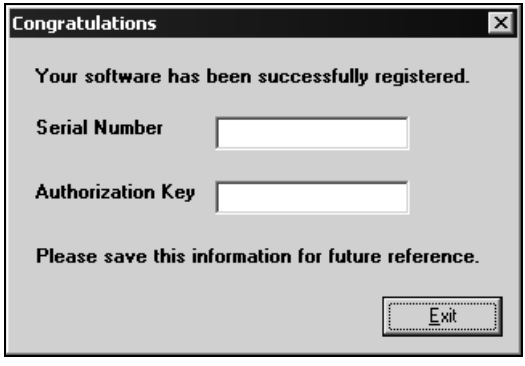


Рис. 2.4. Окно поздравления

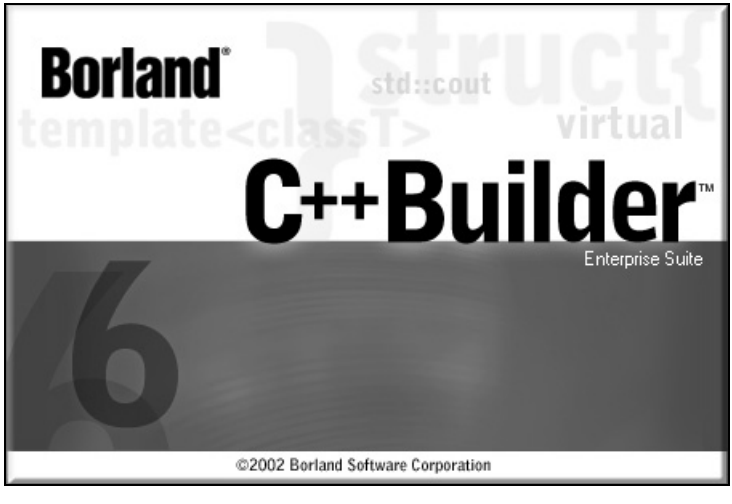
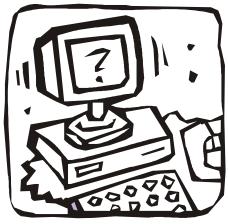


Рис. 2.5. Окно процесса загрузки

После запуска программы на экране монитора появится окно, изображенное на рис. 2.5, которое демонстрирует процесс загрузки программы Borland C++ Builder 6.

Это окно может отображаться несколько секунд, пока загружается программа.



Глава 3

Интерфейс

В этой главе говорится о внешнем виде среды разработки и назначении окон, меню команд, кнопок управления и других органов управления.

После запуска среды разработки окно загрузки сменяется несколькими новыми окнами, за которыми просматривается рабочий стол компьютера. Все эти окна представляют собой интерфейс программы Borland C++ Builder 6, который изображен на рис. 3.1.

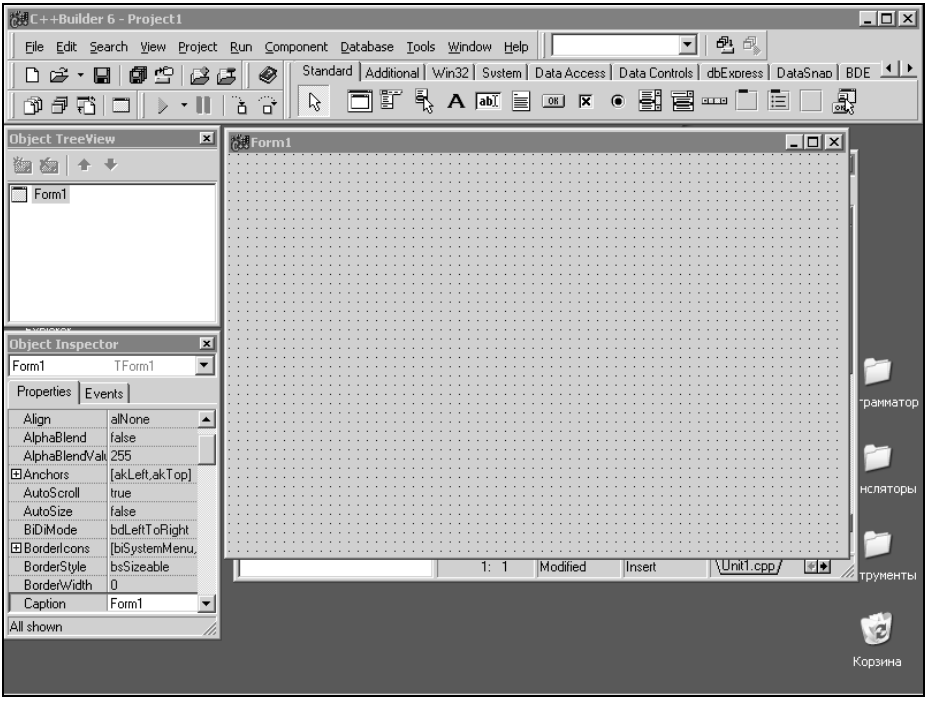


Рис. 3.1. Интерфейс программы Borland C++ Builder 6

В терминологии программистов этот интерфейс называется средой быстрой разработки приложений RAD (Rapid Application Development). Такое название он получил за то, что создание программы в нем сводится к простому конструированию внешнего вида будущей программы из готовых кубиков, а большую часть стандартных и рутинных операций за человека выполняет компьютер. Например, компьютер автоматически создает заготовку текста программы для функций обработки событий. Но об этом чуть позже.

Самое верхнее главное окно интерфейса (рис. 3.2) имеет заголовок **C++Builder 6 - Project1**, который отражает название среды разработки и имя нового проекта, из которого будет получена работающая программа.

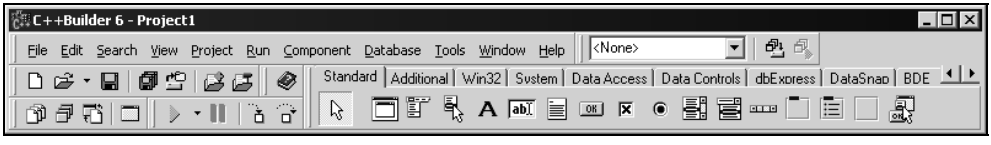


Рис. 3.2. Главное окно интерфейса

Проектом называется вся группа программных файлов, которые необходимы для создания конечной исполняемой программы. Так, например, в состав проекта могут включаться файлы с текстами программ, файл ресурсов с рисунками курсоров и иконок (значков), звуковые файлы и т. п. Первоначально проект хранится в памяти компьютера, и для того чтобы сохранить его на диске, необходимо будет выполнить стандартные операции сохранения, создав при этом отдельную папку. Кроме того, интерфейс сам предложит сохранить проект, если вы решите выйти из программы или попытаетесь создать новый проект.

Рассмотрим главное окно интерфейса. На строке заголовка проекта находятся кнопки свертывания, восстановления и закрытия окна. Под заголовком размещается строка главного меню, которая предоставляет доступ ко всем функциям и командам среды разработки. Под главным меню располагаются быстрые кнопки, объединенные в группы по назначению. Они позволяют получить быстрый доступ к наиболее часто используемым командам.

Справа от быстрых кнопок расположена палитра визуальных компонентов VCL (Visual Component Library, библиотека визуальных компонентов). Это те самые объекты или программные компоненты, предназначенные для быстрого создания визуальных программ для Windows. Компоненты позволяют быстро создавать в программе различные программные кнопки, рисунки, надписи, таймеры, календари и т. п. Палитра визуальных компонентов состоит из нескольких закладок, на которых располагаются компоненты, распределенные по группам. Именно с помощью этих компонентов вы будете быстро создавать новые программы.

При небольших разрешениях экрана все эти закладки не вмещаются в пределы рабочей области экрана и становятся невидимыми. Для получения доступа к их использованию необходимо воспользоваться кнопками прокрутки (стрелками), которые расположены в правой верхней части палитры компонентов. Подведите курсор к одной из кнопок и нажмите левую кнопку мыши. При этом закладки сдвинутся и появится группа, которая первоначально не отображалась (была недоступна).

В центре экрана располагается окно дизайнера форм (рис. 3.3). Это окно будущей программы. Оно содержит строку заголовка, в котором отображается название формы Form1 (созданное по умолчанию) и кнопки управления окном. На поле этого окна будут помещаться компоненты VCL в виде программных кнопок, надписей и других элементов будущей программы.

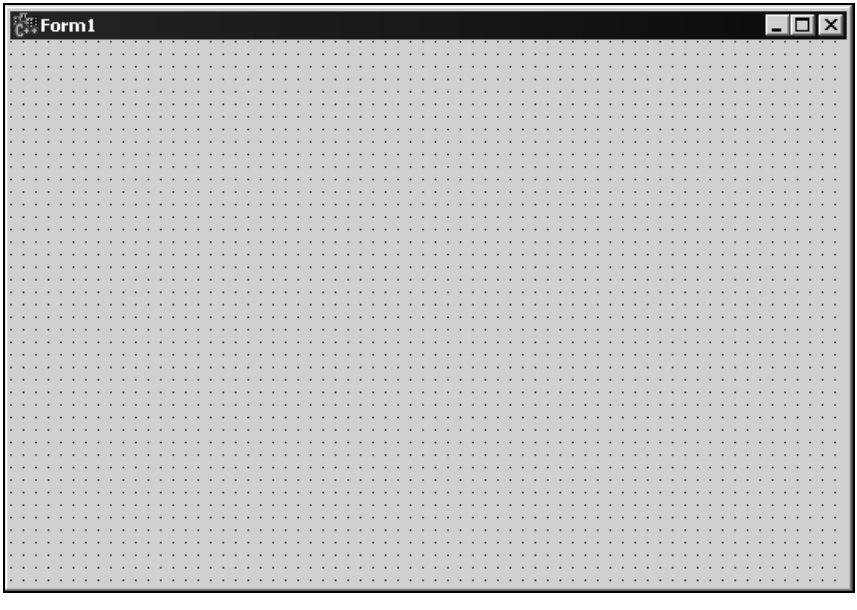


Рис. 3.3. Окно дизайнера форм

Под окном дизайнера форм располагается окно редактора кода (рис. 3.4) с заголовком Unit1.cpp (также созданным по умолчанию), в котором производится набор и редактирование кода (текста) программы.

Следующее окно интерфейса располагается в левой нижней части экрана. Это окно инспектора объектов **Object Inspector** (рис. 3.5). В этом окне производится настройка основных свойств визуальных компонентов. Расположение окна инспектора объектов в программе не фиксированное и при желании его можно переместить в ту часть рабочей области программы, которая для вас наиболее удобна. Для этого необходимо нажать левую кнопку мыши на строке заголовка окна и, удерживая ее, переместить окно.

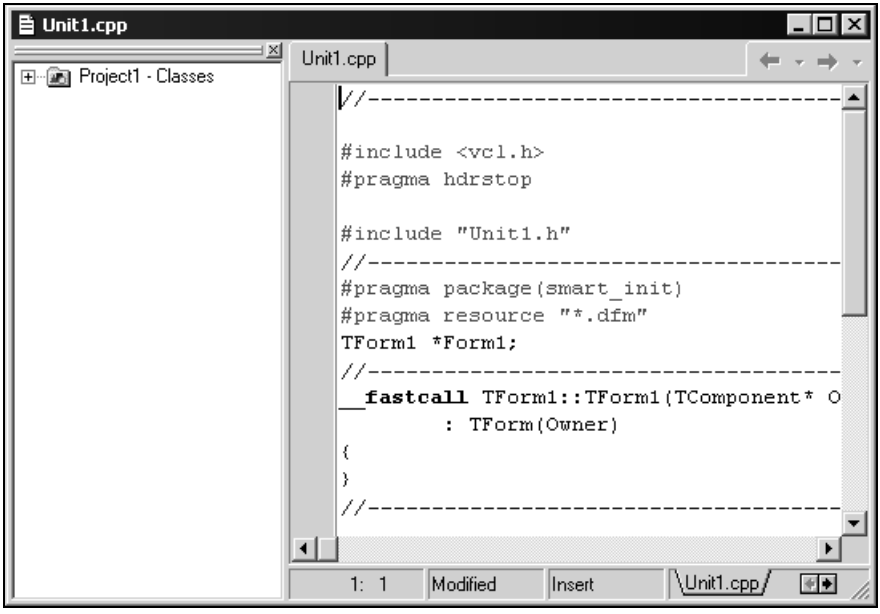


Рис. 3.4. Окно редактора кода

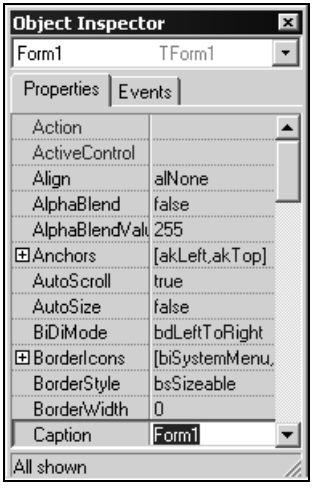


Рис. 3.5. Окно инспектора объектов

Над окном инспектора объектов расположено окно просмотра объектов **Object TreeView** (рис. 3.6). Оно отображает в виде дерева всю структуру проекта, состоящего из форм, кодов (текстов) программ и других ресурсов (файлов).

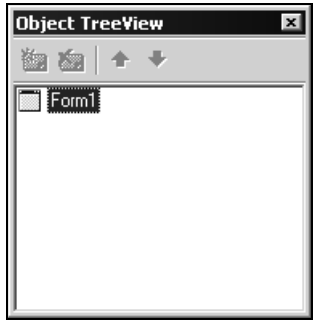
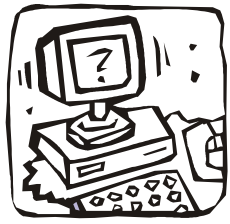


Рис. 3.6. Окно просмотра объектов

Советую читателям некоторое время самостоятельно знакомиться с интерфейсом среды разработки на компьютере, для того чтобы в памяти закрепился его образ. Ведь только желание и живой интерес украшают процесс изучения.

Данный интерфейс будет вашим самым главным инструментом при создании программ.



Глава 4

Первая программа

В этой главе читатели смогут самостоятельно очень быстро создать и запустить свою первую программу, работающую в операционной системе Windows.

Думаю, многим читателям уже не терпится написать первую программу под Windows. Пожалуй, нет ничего проще. Фактически она уже написана. И сделал это за вас RAD-интерфейс. После того как среда разработки запущена, выполните команду **Run** из главного меню команд, щелкните левой кнопкой мыши по зеленой кнопке-стрелке на панели быстрых кнопок или нажмите клавишу <F9> на клавиатуре. После этого произойдет компиляция проекта (преобразование программы в машинный код, понятный компьютеру) с отображением на экране окна, подобного представленному на рис. 4.1.

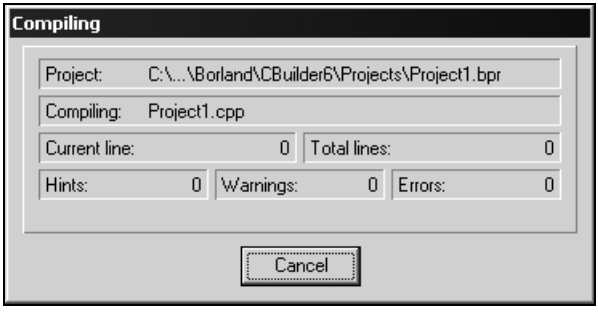


Рис. 4.1. Окно компиляции проекта

После того как это окно автоматически закроется, программа запустится, и на экране появится изображение окна вашей первой программы (рис. 4.2).

В верхней части этой программы будет виден фирменный значок пакета Borland C++ Builder и название окна программы **Form1**. Итак, мы создали новую программу, не написав при этом ни одной строчки кода! Впечатляет, не правда ли?! Какая же польза от такой программы? Что она умеет делать? Пока не очень много, но кое-что все же умеет.

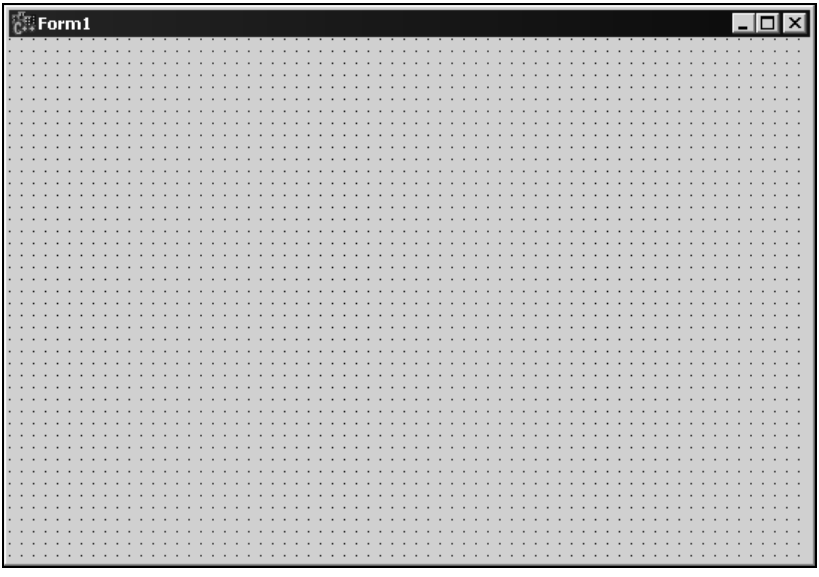


Рис. 4.2. Окно первой программы

Во-первых, мы получили полноценное оконное приложение, которое работает в операционной системе Windows.

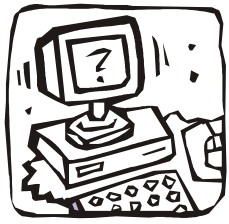
Во-вторых, это приложение имеет стильный интерфейс Windows и привычные стандартные органы управления, расположенные справа вверху. Вы можете сворачивать данное приложение и разворачивать его на весь экран, изменять размеры окна программы и перемещать его с помощью мыши.

Раньше для создания программы с подобными функциями программист затрачивал целые дни и недели. А мы с вами создали такую программу в считанные секунды. Конечно же, это заслуга RAD-интерфейса. Именно благодаря такому интерфейсу, нам удалось создать приложение так быстро и легко. Это одно из замечательных свойств данного интерфейса, потому он и получил очень большое распространение и признание во всем мире.

Обратите внимание, что при сворачивании приложение будет иметь имя Project1. Теперь перед нами встают вопросы о том, как сделать наше приложение полезным, как изменить его название, значок и т. д. Очень хорошо, что эти вопросы появились. Они пробуждают в нас интерес и жажду к знаниям — основным движущим силам прогресса и творчества.

Остановим работу нашей программы, щелкнув левой кнопкой мыши по крестику в верхней правой части ее окна, и продолжим обучение.

В следующих главах мы более подробно рассмотрим каждое из описанных ранее окон интерфейса и других органов управления пакета Borland C++ Builder 6. Это позволит понять, как изменять программу, для того чтобы она выполняла нужные действия.



Глава 5

Главное меню

В этой главе рассматривается главное меню команд интерфейса среды разработки Borland C++ Builder 6.

Главное меню предоставляет доступ к основным командам программы Borland C++ Builder 6, объединенным в группы. Рассмотрим состав и назначение этих групп.

Группа команд *File*

Первая группа меню команд под названием **File** (Файл) изображена на рис. 5.1.

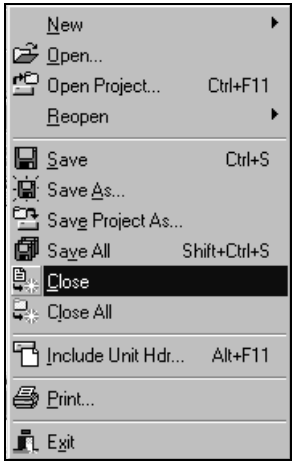


Рис. 5.1. Группа команд File

Команды этого меню осуществляют работу с файлами и предоставляют доступ к операциям создания (**New**) новых форм (окон) и приложений, открытия (**Open**), сохранения (**Save**) и закрытия (**Close**) файлов и проектов, печати

текстов программ (**Print**) и добавления заголовочных модулей (**Include**), то есть файлов с расширением h.

Обратите внимание на то, что перед некоторыми командами меню находятся значки (иконки). Эти же значки изображены на кнопках быстрого запуска. Благодаря такому соответствию можно без труда понять назначение быстрых кнопок. Кроме того, справа от некоторых команд меню написано соответствие команд горячим клавишам, то есть клавишам, с помощью которых вызывается данная команда. Еще правее этих надписей кое-где имеются треугольные стрелки, указывающие на то, что данная команда имеет расширенные возможности, которые становятся доступными после наведения на эту стрелку курсора мыши.

Группа команд *Edit*

Вторая по порядку группа команд меню называется **Edit** (Редактор), она изображена на рис. 5.2.

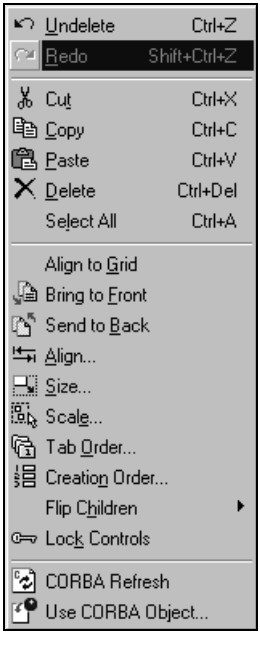


Рис. 5.2. Группа команд **Edit**

В этом меню собраны команды редактирования, такие как отмена (**Undelete**) и повторение (**Redo**) операций, вырезание (**Cut**), копирование (**Copy**), вставка (**Paste**) и удаление (**Delete**) фрагментов текста. А также команды выделения всего текста (**Select All**), выравнивания компонентов (**Align**) и настройки редактора кода (текста программы).

Группа команд *Search*

Следующая группа команд **Search** (Поиск) изображена на рис. 5.3.

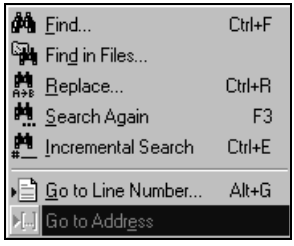


Рис. 5.3. Группа команд **Search**

Команды данного меню позволяют осуществить поиск текста в файле, продолжить поиск после первого вхождения, произвести автоматическую замену, а также быстро перейти к строке кода, задав ее номер.

Группа команд *View*

Группа команд **View** (Вид) изображена на рис. 5.4.

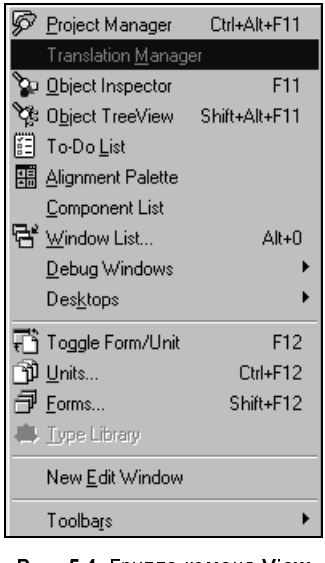


Рис. 5.4. Группа команд **View**

Из этого пункта меню вызываются основные диалоговые окна управления проектом и компонентами, такие как менеджер проектов (**Project Manager**),

список компонентов (**Component List**) и список окон (**Window List**). Также из этого пункта меню открываются все окна отладки программ (**Debug Windows**), работу с которыми мы рассмотрим позже.

Группа команд *Project*

Группа команд **Project** (Проект) приведена на рис. 5.5.

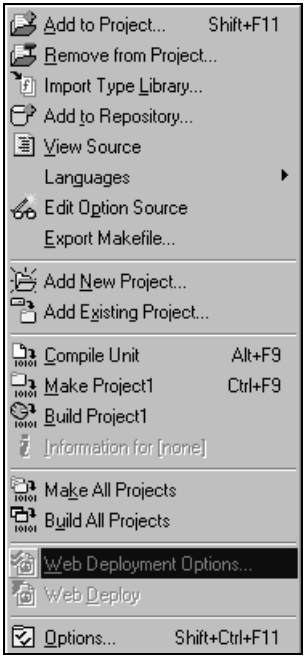


Рис. 5.5. Группа команд **Project**

В этой группе меню собраны команды управления проектом. С их помощью можно добавлять и удалять модули (файлы с текстами программ), добавлять библиотеку компонентов VCL, откомпилировать проект и т. д.

Группа команд *Run*

Группа команд **Run** (Выполнить) изображена на рис. 5.6.

С помощью команд этого меню выполняется запуск и останов программ, запуск программ в непрерывном и пошаговом режимах, добавление переменных для просмотра, установка точек останова и другие действия по отладке программы.

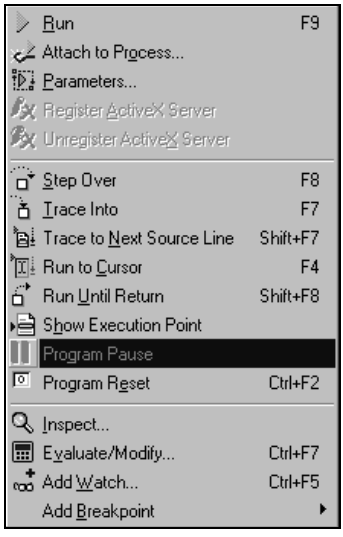


Рис. 5.6. Группа команд **Run**

Группа команд *Component*

Группа команд **Component** (Компонент) представлена на рис. 5.7.

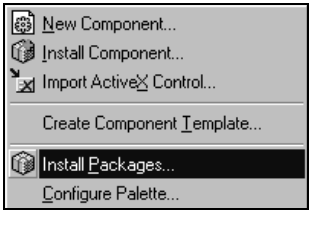


Рис. 5.7. Группа команд **Component**

Из этого меню вызываются команды добавления в систему новых компонентов и конфигурации их палитры.

Группа команд *Database*

Группа **Database** (База данных) (рис. 5.8) содержит команды для работы с базами данных. Изучение компонентов данной группы не входит в задачи данной книги ввиду большого объема материала.

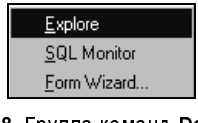


Рис. 5.8. Группа команд **Database**

Группа команд *Tools*

Группа **Tools** (Инструменты) изображена на рис. 5.9.

Рис. 5.9. Группа команд **Tools**

В этом меню собраны команды как по настройке параметров программы, так и команды вызова различных дополнительных утилит.

Группа команд *Windows*

Группа **Windows** (Окна) приведена на рис. 5.10.

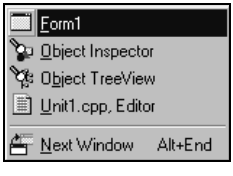


Рис. 5.10. Группа команд **Windows**

С помощью команд данного меню производится управление окнами интерфейса.

Группа команд *Help*

Замыкает главное меню группа команд **Help** (Помощь), представленная на рис. 5.11.

В нее включены команды, с помощью которых можно получить справку как о языке C++, так и о работе с пакетом Borland C++ Builder 6 и всеми его компонентами. Выбрав в данном меню команду **About**, вы увидите сведения о данном пакете и о вашем компьютере. Это окно изображено на рис. 5.12.

Рис. 5.11. Группа команд **Help**

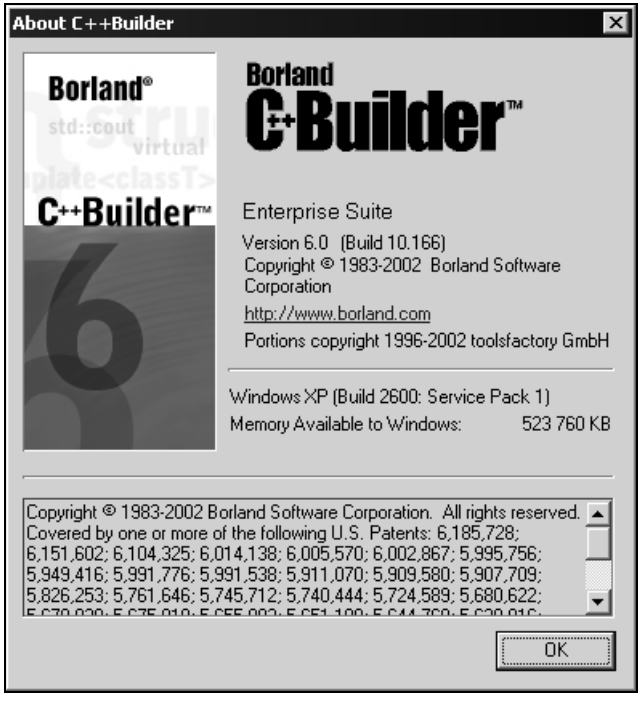
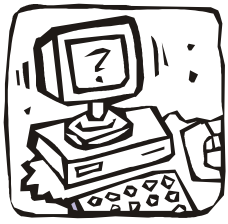


Рис. 5.12. Окно команды **About**



Глава 6

Быстрые кнопки

В этой главе описываются состав и назначение кнопок для быстрого выполнения некоторых часто применяемых команд.

Быстрые кнопки расположены рядом с главным меню (рис. 6.1).

Рис. 6.1. Быстрые кнопки

С их помощью осуществляется быстрый доступ к основным и часто используемым командам главного меню. Все эти кнопки имеют всплывающее меню, появляющееся при наведении на них курсора мыши, и горячие клавиши, показываемые в скобках. Все быстрые кнопки так же, как и главное меню, разделены на группы.

Кнопки *Standard* (Стандартные)

Первая из этих кнопок позволяет создавать новые объекты программы. Такими объектами могут быть программы, новые формы, файлы, библиотеки и т. п. Для создания нового объекта необходимо нажать кнопку **New**. При этом откроется диалоговое окно **New Items** (рис. 6.2), в котором можно выбрать необходимый для создания программы объект. Это окно имеет множество вкладок, в каждой из которых находятся объекты определенного типа. Для начала работы с Borland C++ Builder 6 достаточно будет пользоваться лишь несколькими объектами из вкладки **New**.

Другие кнопки этой группы позволяют сохранять (**Save**) и открывать (**Open**) как отдельные файлы, так и целые проекты. Замыкают группу две кнопки управления проектами, с помощью которых можно добавлять файлы к проекту (**Add File To Project**) и удалять их из него (**Remove File From Project**).

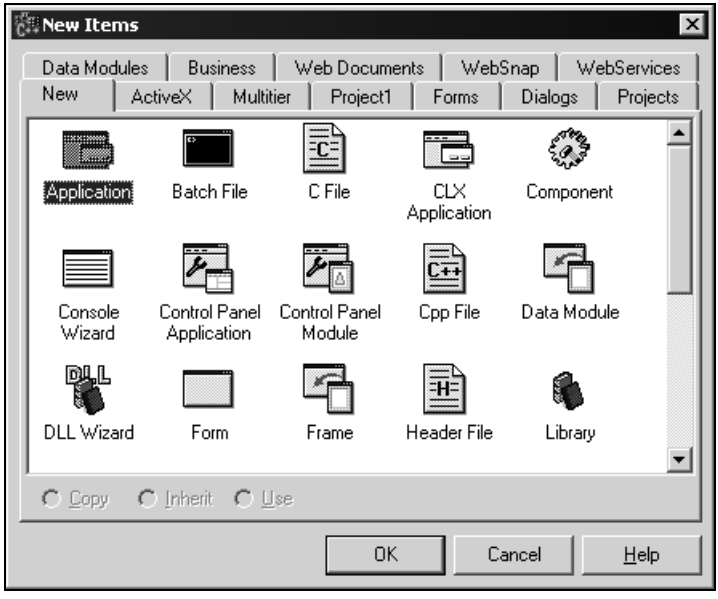


Рис. 6.2. Диалоговое окно **New Items**

Обратите внимание на то, что при подведении курсора к любой из кнопок появляется контекстная справка, в которой приведено название кнопки и клавиши для ее активации с клавиатуры. Например, для кнопки сохранения **Save** такая справка будет содержать слова **Save (Ctrl+S)**. Это значит, что кнопка сохранения файла может быть активирована с клавиатуры одновременным нажатием клавиш <Ctrl> и <S>. Подобные комбинации клавиш существуют и для других органов управления средой разработки. В дальнейшем мы будем прибегать к помощи этих средств, поскольку при разработке программы встречаются ситуации, когда все окна среды разработки закрыты формой программы, и нажать быструю кнопку или выполнить команду меню невозможно с помощью мыши. Тогда приходит на помощь клавиатура. Кроме того, некоторые действия с ее помощью можно выполнить гораздо быстрее.

Кнопки *View* (Вид)

В эту группу собраны кнопки, управляющие просмотром модулей (**View Unit**) и форм (**View Form**), кнопка для переключения окон форм и модулей (**Toggle Form/Unit**), а также кнопка создания новой формы (**New Form**).

Для переключения между формой и редактором кода служит кнопка **Toggle Form Unit**. Если в вашей программе используется несколько форм, то с помощью кнопки **View Form (Shift+F12)** вы можете выбрать форму для просмотра.

Для создания новой формы необходимо нажать кнопку **New Form**. При этом появится новое окно дизайнера форм, в заголовке которого будет отображаться название новой формы **Form2**. Новые формы нужны для создания программ, в которых будет возможность отображения дополнительных окон.

Кнопки *Debug* (Отладка)

В этой группе собраны кнопки для отладки программы. Кнопка **Run (F9)** запускает программу на выполнение. Кнопка **Pause** позволяет приостановить выполнение программы. С помощью кнопок **Trace Into (F7)** и **Step over (F8)** происходит выполнение программы в пошаговом режиме для отладки программы.

Кнопки *Custom* (Заказ)

В группе кнопок **Custom** присутствует одна кнопка **Help contents** для вызова справочной системы среды разработки.

Положение кнопок быстрого доступа в среде разработки нефиксированное, их можно разместить на главной панели по своему усмотрению. Для этого необходимо подвести курсор к левому краю группы кнопок. Нажав и удерживая левую кнопку мыши, нужно переместить группу на новое место. Если группа вынесена за пределы главной панели, ей отводится отдельное окно.

Кроме того, нажав правую кнопку мыши на панели быстрых кнопок, можно открыть окно (рис. 6.3) для настройки отображения любой группы кнопок и других элементов на главной панели системы.

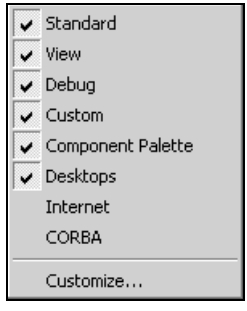
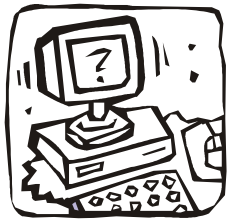


Рис. 6.3. Окно настройки отображения

Устанавливая или сбрасывая флажки в этом окне, можно задавать отображение той или иной группы кнопок или других органов управления среды разработки.



Глава 7

Палитра VCL-компонентов

В этой главе описываются состав и назначение библиотеки визуальных компонентов VCL среды разработки Borland C++ Builder 6.

Разработка программ на Borland C++ Builder 6 построена на основе выбора необходимых визуальных компонентов VCL и расположения их на поле форм (окон будущей программы). Компоненты, таким образом, служат кирпичиками, из которых строятся программы. Кстати, Builder в переводе на русский язык означает "строитель". Все компоненты VCL располагаются на палитре, расположенной ниже и правее главного меню. Палитра компонентов состоит из вкладок. Вкладки позволяют разделить большое число компонентов на группы, близкие по назначению. Щелкая левой кнопкой мыши по вкладкам, можно выбрать необходимую группу компонентов, которая будет отображаться на экране. Например, при щелчке по вкладке **Standard** на экране появятся компоненты, изображенные на рис. 7.1.

Рис. 7.1. Окно с компонентами

Среди них легко найти компонент **Button** (Кнопка) с надписью **OK** и компонент **Label** (Этикетка) с надписью **A**. С назначением компонентов мы познакомимся позже.

Иногда часть компонентов не умещается на экране. Для их просмотра существуют кнопки прокрутки в виде треугольных стрелок слева и справа от самих компонентов. Щелкая по ним кнопкой мыши, можно сдвигать компоненты группы влево или вправо. Все компоненты видны на экране в виде иконок. При наведении на них курсора мыши эти иконки приподнимаются, как кнопки, а под курсором появляется строка с надписью, отображающей название данного компонента.

Любой из компонентов можно поместить на форму двумя способами. Первый способ заключается в двойном щелчке левой кнопкой мыши по самому компоненту. При этом компонент появится на форме строго по центру. После чего его можно переместить с помощью мыши в любую часть формы, а также изменить размеры так, как это делается с окнами программ. Второй способ заключается в том, что сначала производится один щелчок левой кнопкой мыши по компоненту, а затем — в нужном месте формы. При этом можно сразу задать расположение компонента на форме и его размеры. Для этого необходимо перед щелчком на форме выбрать место расположения левого верхнего угла компонента и щелкнуть в этом месте, не отпуская кнопки мыши. Далее необходимо продвинуть указатель мыши на место, выбранное для правого нижнего угла компонента, и отпустить кнопку. Во время такой операции ниже указателя мыши будет высвечиваться окно с координатами курсора для справки о его местоположении в сетке координат формы. Удаление компонента с формы производится клавишей <Delete> после его выделения щелчком левой кнопки мыши.

Палитру компонентов можно совсем убрать с экрана, как и другие элементы главной панели интерфейса. Для этого необходимо нажать правую кнопку мыши на панели главного меню или быстрых кнопок, и в открывшемся контекстном меню (рис. 7.2) убрать галочку напротив названия **Component Palette** (Палитра компонентов) щелчком левой кнопки мыши.

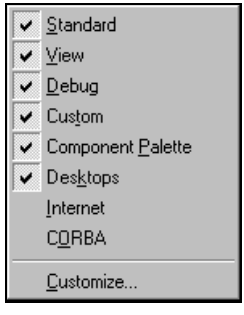


Рис. 7.2. Окно контекстного меню

Положение палитры компонентов на главной панели интерфейса Borland C++ Builder 6 не фиксировано, ее можно разместить по своему усмотрению. Для этого необходимо подвести курсор к левому краю палитры. Нажав и удерживая левую кнопку мыши, переместить палитру на новое место. Если группа вынесена за пределы главной панели, ей отводится отдельное окно.

Для доступа к дополнительным командам и настройки палитры компонентов необходимо нажать правую кнопку мыши на поле палитры. При этом на экране появится окно контекстного меню команд палитры (рис. 7.3).

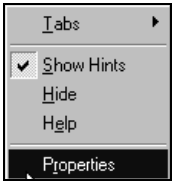


Рис. 7.3. Контекстное меню команд палитры

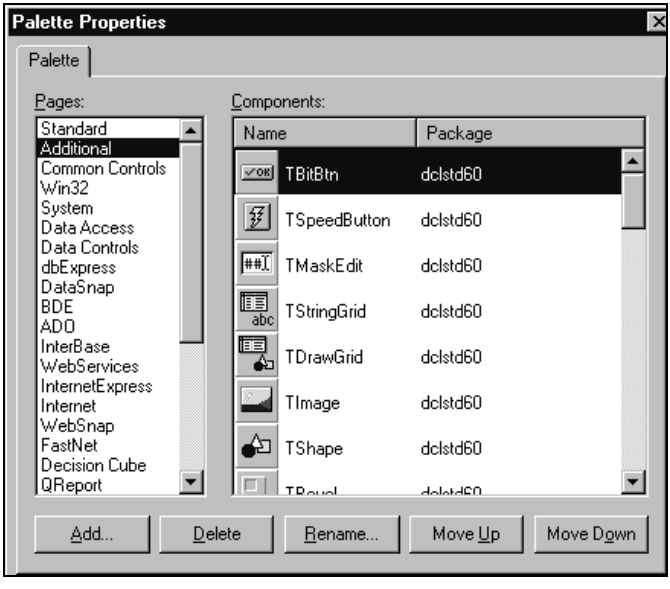
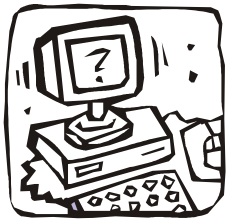


Рис. 7.4. Окно настроек палитры компонентов

В этом окне содержатся команды быстрого перехода к любой вкладке **Tabs**, показа подсказки **Show Hints**, скрытия палитры **Hide**, вызова справки **Help** и изменения настроек **Properties**. После щелчка левой кнопкой мыши по команде настроек **Properties** откроется окно настроек палитры компонентов (рис. 7.4).

В этом окне можно настроить каждую из вкладок палитры и всю палитру в целом. С помощью команды **Rename** можно переименовать любую из вкладок. Команда **Add** позволяет добавить новую вкладку, а команда **Delete** удалить любую вкладку. Для того чтобы, например, вкладка **System** следовала сразу за вкладкой **Standard**, необходимо выделить строку **System**, щелкнув по ней левой кнопкой мыши, и с помощью кнопки **Move Up** (Двигать вверх) переместить строку вверх до строки **Standard**. После чего необходимо нажать кнопку **OK**. Теперь вкладка **System** на палитре компонентов будет располагаться сразу же за вкладкой **Standard**. Аналогично можно перемещать сами компоненты внутри вкладки с помощью кнопок **Move Up** и **Move Down** (Двигать вниз), или скрыть компонент с помощью кнопки **Hide**. Перемещаемый компонент предварительно необходимо выделить щелчком левой кнопки мыши.



Глава 8

Дизайнер форм

В этой главе дается описание окна, в котором производится редактирование форм разрабатываемой программы.

Для размещения визуальных компонентов в окне будущей программы и изменения размеров самого окна служит Дизайнер форм (рис. 8.1).

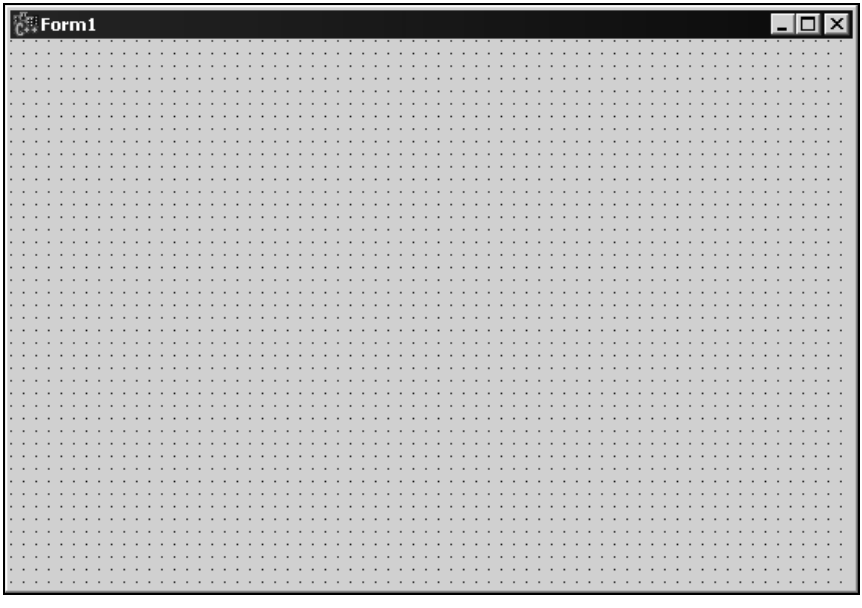


Рис. 8.1. Дизайнер форм

Строка заголовка будущего окна находится в верхней части Дизайнера форм. Правее заголовка находятся стандартные кнопки свертывания, разворачивания и закрытия окна. Остальная область Дизайнера форм покрыта сеткой привязки в виде точек, которая служит для облегчения визуального

выравнивания компонентов. При перемещении компонентов на окне Дизайнера форм с помощью мыши, изменение положения компонентов происходит дискретно с шагом сетки. Этот шаг можно изменить, вызвав из главного меню **Tools** команду **Environment Options** (рис. 8.2).

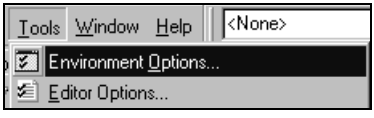


Рис. 8.2. Вызов команды

В открывшемся окне необходимо нажать кнопку вкладки **Designer**, после чего появится окно, приведенное на рис. 8.3.

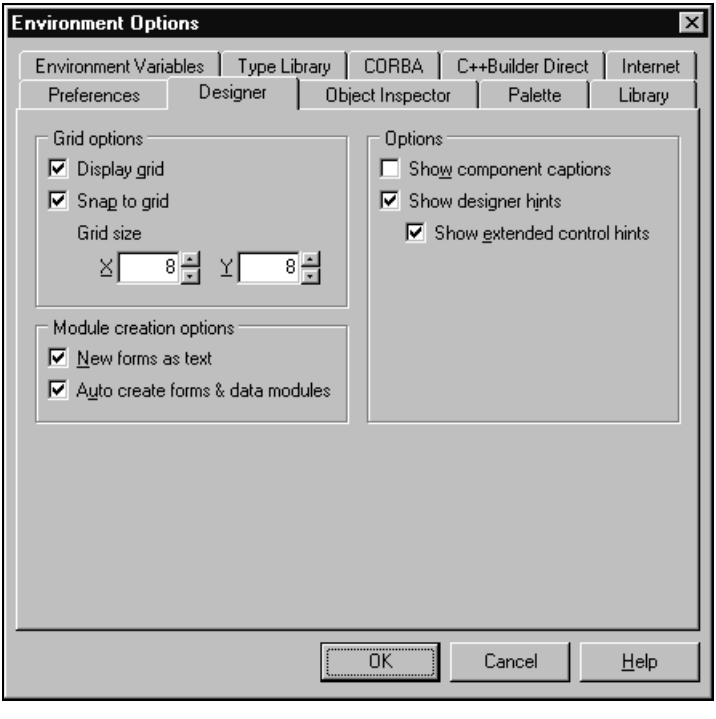


Рис. 8.3. Окно **Environment Options**, вкладка **Designer**

Шаг сетки **Grid size** в пикселах по горизонтали и вертикали задается с помощью изменения величины параметров сетки **X** и **Y** соответственно. Кроме того, сетку можно совсем отключить, сняв галочку в поле **Display grid**. Для перемещения компонентов без привязки к сетке необходимо снять галочку в строчке **Snap to grid**. Все эти элементы размещены в группе **Grid options**. Соседняя группа **Options** объединяет в себе элементы управления показом

заголовка компонента **Show component captions**, подсказки **Show designer hints** и расширенного управления подсказок **Show extended control hints**. Ниже расположена группа **Module creation options**. Она содержит элемент управления **New forms as text**, который определяет, как сохранять файл описания формы: как текст или как двоичный файл. Второй элемент управления этой группы **Auto create forms & data modules** определяет, что новые формы будут создаваться автоматически.

Перемещать компоненты по окну формы без привязки к сетке можно также с помощью клавиш со стрелками, удерживая при этом нажатой клавишу <Ctrl>. С помощью клавиш со стрелками можно изменить размеры компонента, удерживая нажатой клавишу <Shift>. Предварительно компонент необходимо выделить, щелкнув по нему левой кнопкой мыши.

Для практики разместите на поле формы два компонента кнопок **Button** и один компонент этикетки **Label**. После чего измените положения и размеры компонентов и самой формы аналогично рис. 8.4.

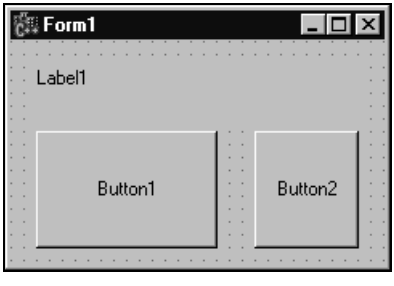
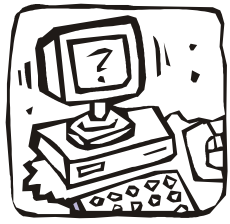


Рис. 8.4. Пример формы

Затем удалите эти компоненты с формы посредством клавиши <Delete>, предварительно выделив каждую из них.



Глава 9

Редактор кода

В этой главе дается описание окна, в котором производится редактирование текста программ, именуемого в среде разработки программным кодом.

Для создания, просмотра и редактирования текста программ служит редактор кода. Внешний вид окна редактора кода представлен на рис. 9.1.

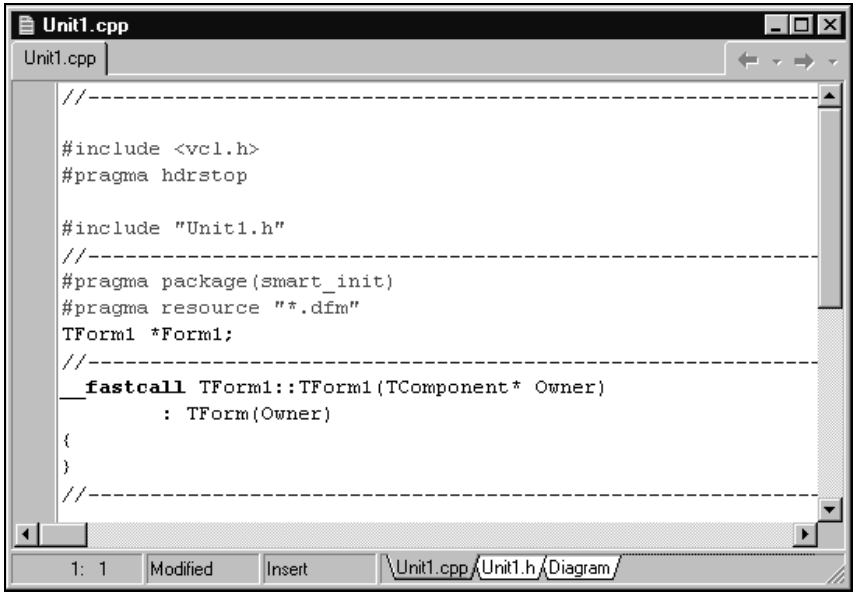


Рис. 9.1. Окно редактора кода

В среде разработки Borland C++ Builder файл с текстами программ (кодовая часть) и файл заголовков с описаниями функций и переменных (интерфейсная часть) принято называть модулями. Каждый модуль состоит из двух частей. Кодовая часть хранится в файле с расширением cpp (сокращение

от C++), а интерфейсная часть в файле с расширением h (сокращение от head, голова). Таких модулей в программе может быть несколько, в зависимости от количества форм (окон) в программе и ее сложности. По умолчанию файлам модулей даются названия **Unit1**, **Unit2** и т. д., которые могут быть изменены разработчиком программы при создании и сохранении проекта. Названия этих файлов видны на соответствующих вкладках Дизайнера форм внизу и вверху его окна. Щелкая левой кнопкой мыши по этим вкладкам, можно выбирать файл для редактирования. Кроме того, на вкладке находится страница **Diagram**, которая позволяет строить алгоритмы или диаграммы, показывающие взаимосвязи компонентов всей программы. Такая диаграмма служит в качестве шпаргалки для программиста и хранится вместе с другими файлами проекта. К тому же она избавляет разработчика от необходимости использовать и хранить записи и рисунки на бумаге.

Набор текста программ, их просмотр и редактирование производится в центральном окне редактора кода. Левее этого окна находится поле для установки вкладок и точек останова при отладке программы.

Редактор кода можно настроить по своему вкусу. Для этого необходимо щелкнуть правой кнопкой мыши правее названия файла в верхней части окна. После чего откроется окно команд, приведенное на рис. 9.2.

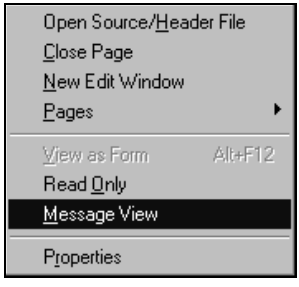


Рис. 9.2. Окно команд редактора кода

В этом окне появляется доступ к командам открытия файла исходных кодов или файла заголовков **Open Source/Header File**, закрытия страницы **Close Page**, открытия нового окна редактирования **New Edit Window**, выбора редактируемой страницы **Pages**, установки атрибута только чтения **Read Only**, открытия окна сообщений **Message View** и настройки свойств **Properties** редактора кода. Выбрав команду **Properties** щелчком левой кнопки мыши, открывается окно настройки редактора кодов **Editor Properties** (рис. 9.3).

Это окно содержит несколько вкладок. В первой из них (**General**) производится настройка режимов форматирования текста и схема раскладки клавиатуры.

При выборе вкладки **Display** откроется окно, приведенное на рис. 9.4.

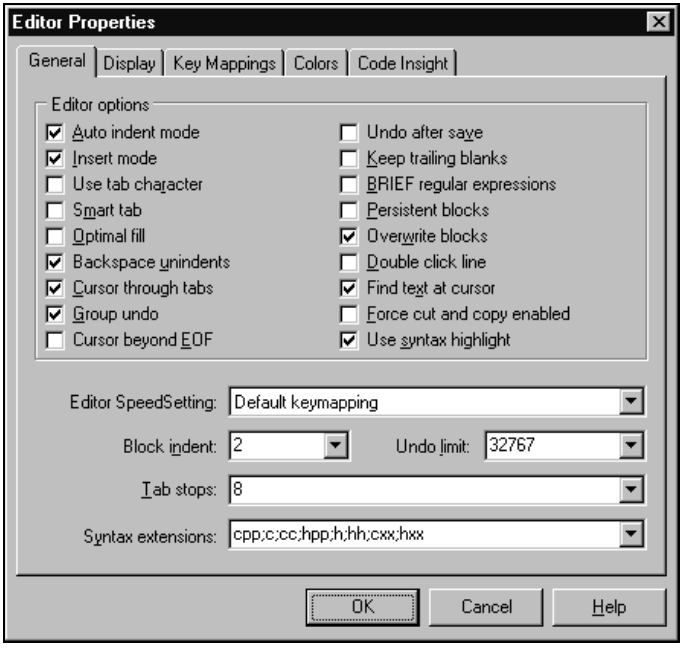


Рис. 9.3. Диалоговое окно **Editor Properties**, вкладка **General**

Рис. 9.4. Диалоговое окно **Editor Properties**, вкладка **Display**

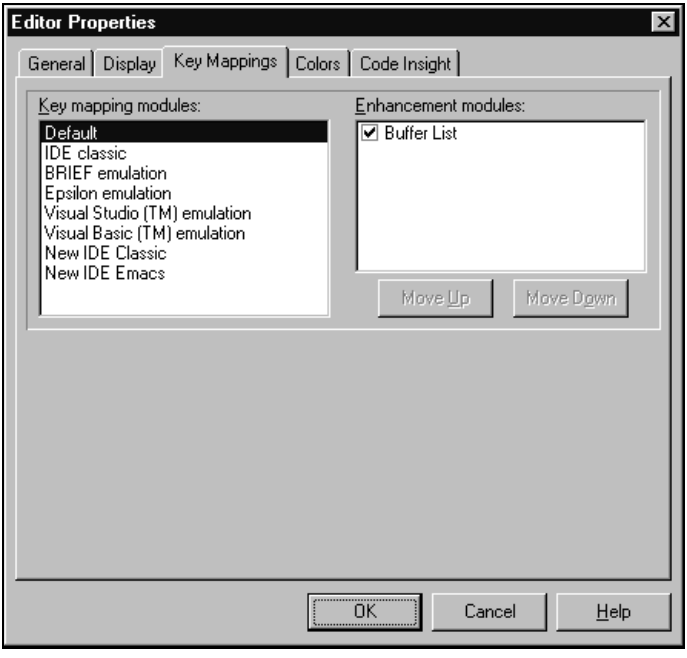


Рис. 9.5. Диалоговое окно **Editor Properties**, вкладка **Key Mappings**

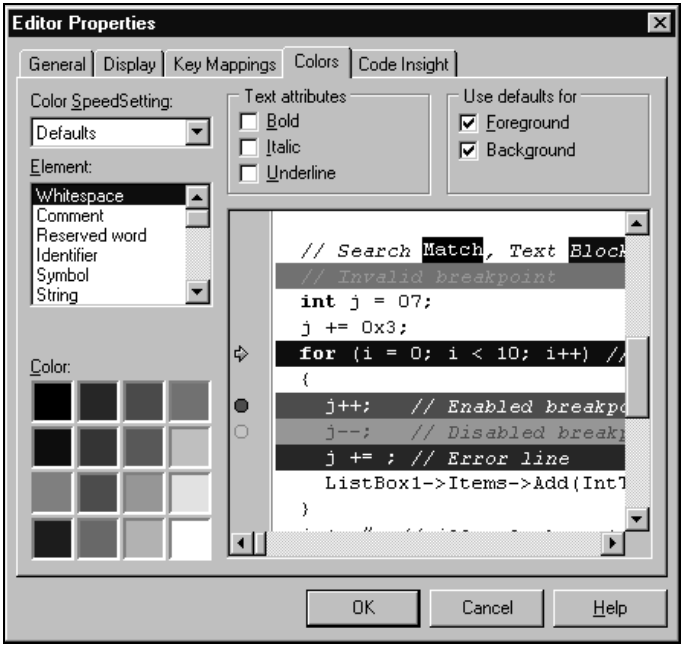


Рис. 9.6. Диалоговое окно **Editor Properties**, вкладка **Colors**

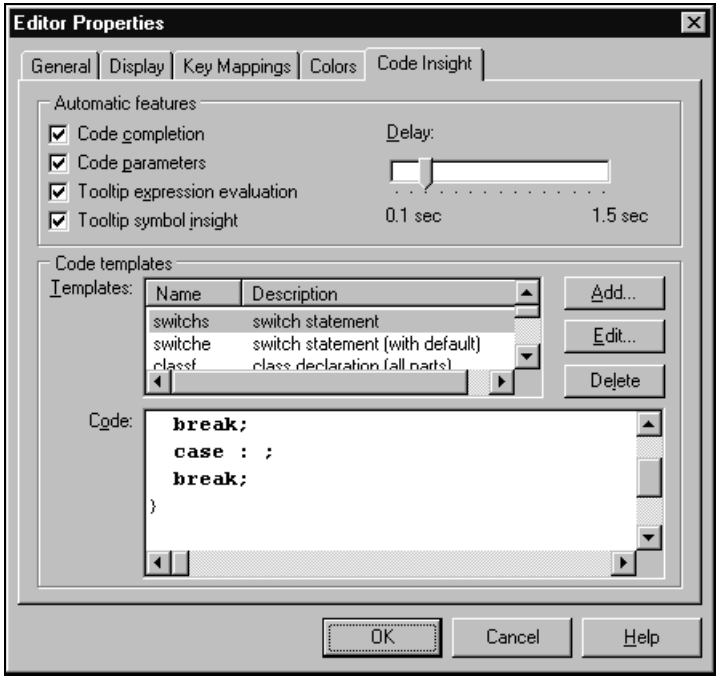


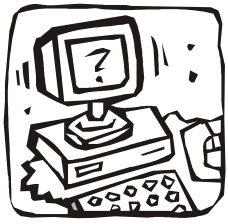
Рис. 9.7. Диалоговое окно **Editor Properties**, вкладка **Code Insight**

В этой вкладке задается форма курсора, ширина полей отступа и подшивки, а также шрифт редактора.

Вкладка **Key Mappings** (см. рис. 9.5) позволяет определить действующие наборы раскладок клавиатуры.

Вкладка **Colors** (см. рис. 9.6) содержит параметры, управляющие цветом и типом шрифта для выделения синтаксических конструкций языка программирования C++.

Вкладка **Code Insight** (рис. 9.7) содержит параметры, управляющие суфлером (подсказчиком) кода и настройкой шаблонов кода.



Глава 10

Инспектор объектов

В этой главе дается описание окна, в котором производится редактирование размеров окон, шрифт и цвет текста, вид кнопок и т. п., именуемых в среде разработки свойствами объектов.

Инспектор объектов (Inspector Object) предназначен для редактирования свойств объектов (форм и визуальных компонентов) во время разработки программы. С его помощью можно установить начальные свойства объектов и назначить обрабатываемые события. Рассмотрим окно инспектора объектов (рис. 10.1).

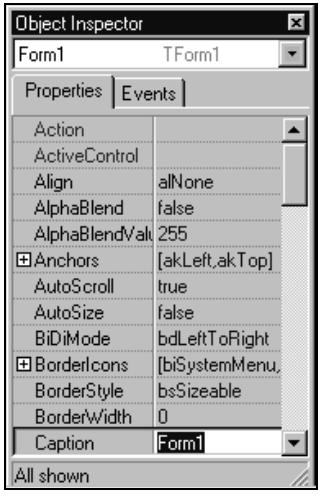


Рис. 10.1. Окно инспектора объектов

В верхней части окна инспектора находится поле выбора объектов, в котором отображается название выбранного объекта. Для выбора нового объекта необходимо открыть выпадающий (раскрывающийся) список щелчком

левой кнопки мыши по кнопке со стрелкой. Затем выбрать необходимый объект из списка. После этого можно редактировать свойства выбранного объекта. Второй способ выбора объекта заключается в том, что производится щелчок левой кнопкой мыши по самому объекту. Третий способ выбора объекта состоит в том, что производится щелчок левой кнопкой мыши по названию объекта в окне просмотра дерева объектов **Object TreeView** (рис. 10.2).

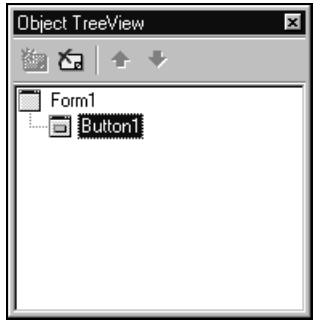


Рис. 10.2. Окно просмотра дерева объектов

Ниже поля выбора объектов находятся кнопки вкладок свойств **Properties** и событий **Events**. На вкладке **Properties** располагаются все не скрытые свойства выбранного объекта. Вкладка состоит из двух полей. В левом поле отображается название свойств, а в правом — их значение. Свойства могут быть вложенными с обозначением плюса в квадратике, расположенном левее названия свойства. Для их раскрытия и редактирования необходимо щелкнуть левой кнопкой мыши по этому значку, а затем по самому свойству. Если свойств много, их можно просматривать с помощью кнопки прокрутки. Например, свойство **Caption** отражает заголовок объекта, а свойство **Name** — имя объекта.

На вкладке **Events** располагаются события, на которые может реагировать объект. Она позволяет связать объект с событиями, происходящими в момент выполнения программы. Вкладка **Events** разделена на два поля. В левом поле отображаются названия событий, на которые может реагировать объект, а в правом — название методов их обработки, или проще, функций-подпрограмм обработки. Для создания обработчика необходимо выполнить двойной щелчок левой кнопкой мыши в правом поле напротив названия выбранного события. При этом появится окно редактора кода, в котором автоматически создастся заготовка текста программы обработчика события. Между фигурными скобками заготовки разработчиком вводится код программы обработки события. В результате при возникновении описанного события для выбранного объекта будет выполняться данная программа. Например, событие **OnClick** означает щелчок левой кнопкой мыши по объекту.

Рассмотрим пример простой программы. Пусть эта программа будет иметь всего одно окно, содержащее одну-единственную кнопку, щелчок по которой будет закрывать приложение. Запустите Borland C++ Builder 6.0 и поместите на форму **Form1** одну кнопку **Button1**. Уменьшите размеры формы и переместите кнопку в центр формы. Выберите в инспекторе объектов форму **Form1** и выделите ее свойство **Caption**. Замените слово **Form1** на слово **Программа 1**. Вы увидите, что заголовок самой формы тоже изменился. Теперь выберите в инспекторе объектов **Button1** и также измените его свойство **Caption** с **Button1** на **Выход**. Обратите внимание, что кнопка на форме стала называться **Выход**. Далее переключитесь в инспекторе объектов на закладку событий **Events** и щелкните дважды в поле напротив события **OnClick**. При этом откроется окно редактора кодов с заготовкой текста программы обработчика события. Впишите между фигурными скобками текста программы строку `Close();`. Это команда закрытия приложения. В результате должен получиться код (текст) программы, приведенный ниже.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------
```

Здесь в первой строке записан заголовок функции-подпрограммы обработки события щелчка **Click** по кнопке **Button1** на форме **Form1**. Тип `void` означает, что функция не возвращает параметров в основную программу. Между фигурными скобками могут записываться операторы, команды и функции. Последняя строка текста является комментарием-разделителем. Если вы допустите ошибку при написании команды, в инспекторе кода откроется окно сообщений, в котором будет выведен текст сообщения об ошибке. Для вызова справки о команде `Close()` поместите курсор на текст этой команды и нажмите комбинацию клавиш <Ctrl>+<F1>. Аналогично вы можете вызывать справку и по любым другим командам и операторам.

Теперь запустите программу, выполнив команду **Run** из главного меню или из панели быстрых кнопок или нажав клавишу <F9> на клавиатуре. После компиляции программа запустится и появится окно, подобное приведенному на рис. 10.3.

Щелкните левой кнопкой мыши на кнопке **Выход**, и приложение закроется. Программа работает и уже выполняет команды, пусть даже такие простые. Если в программе имеются ошибки и она не будет закрываться, выполните команду **Program reset** из главного меню или нажмите комбинацию клавиш <Ctrl>+<F9> на клавиатуре. После чего проверьте программу и, устранив ошибки, запустите снова. Исполняемый файл программы вместе с файлами проекта находится на диске, прилагаемом к этой книге. Вы можете сравнить файлы из каталога \PROJECT\PROG1 компакт-диска с теми, что получились у вас.

Рис. 10.3. Окно программы

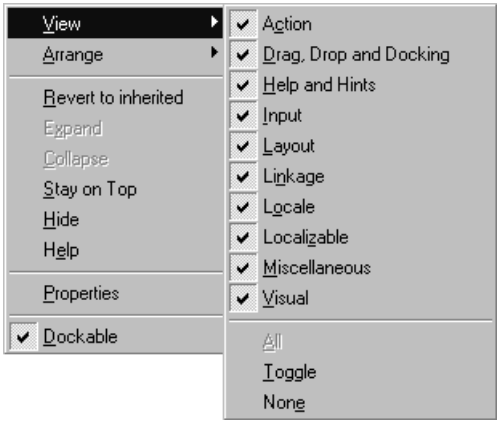


Рис. 10.4. Окно контекстного меню команд

Для доступа к дополнительным командам и настройкам инспектора объектов необходимо нажать правую кнопку мыши на окне инспектора. При этом на экране появится окно контекстного меню команд (рис. 10.4).

В этом окне содержатся команды просмотра **View** свойств и событий по группам размещения (упорядочивания) **Arrange** записей полей по категории или имени, размещения окна поверх всех других окон **Stay on Top**, скрытия окна **Hide**, вызова справки **Help**, изменения настроек **Properties** и пр. После щелчка левой кнопкой мыши по команде настроек **Properties** откроется окно настроек инспектора объектов (рис. 10.5). Это же окно можно открыть, вызвав команду главного меню **Tools Environment Options** и затем открыв вкладку **Object Inspector**.

В данном окне можно настроить цвета записей свойств и событий на полях инспектора объектов, а также осуществить выбор **Options** отображения некоторых свойств и событий, в том числе и тех, на которые имеются ссылки **References**.

При изучении визуальных компонентов и разборе новых примеров программ мы еще не раз вернемся к инспектору объектов. А пока для закреп-

ления материала на практике самостоятельно попробуйте изменить некоторые свойства формы и кнопки и посмотрите, что при этом получается при выполнении программы.

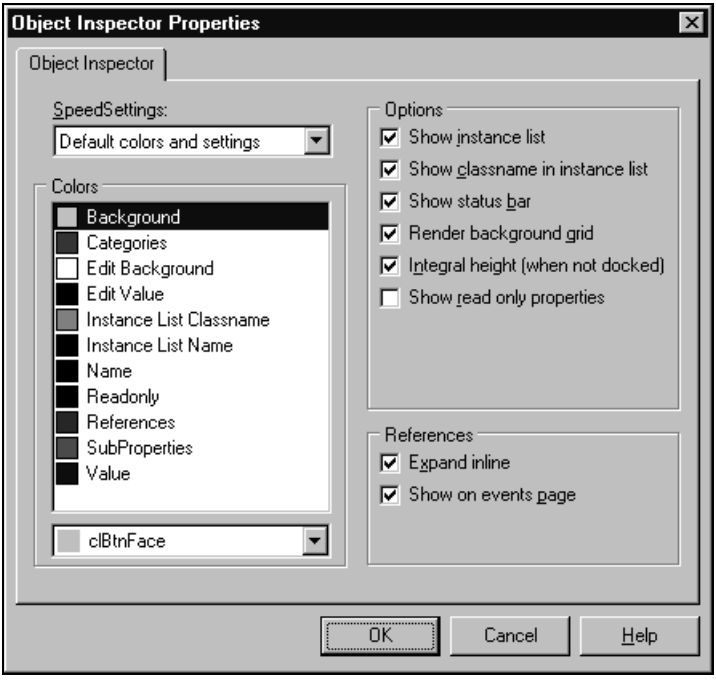
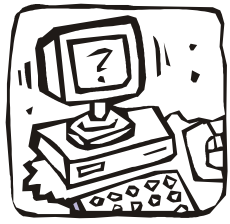


Рис. 10.5. Окно настроек инспектора объектов



Глава 11

Файлы проекта

Здесь дается описание файлов, из которых состоит проект, и советы по работе с проектом.

Теперь пора познакомиться подробнее с файлами проекта и операциями, производимыми над ними. При работе в среде разработки Borland C++ Builder 6 довольно часто приходится открывать и сохранять файлы. Все изменения в проекте перед запуском программы необходимо сохранять в файлах проекта во избежание их потери в случае возникновения ошибок или зависания компьютера. При этом можно производить сохранение всего проекта целиком или сохранять отдельно необходимые файлы. Для сохранения всего проекта целиком вызовите из главного меню команду **Save Project As** (Сохранить проект как) или **Save All** (Сохранить все) или нажмите комбинацию клавиш <Shift>+<Ctrl>+<S>. На экране отобразится стандартное диалоговое окно сохранения файлов (рис. 11.1).

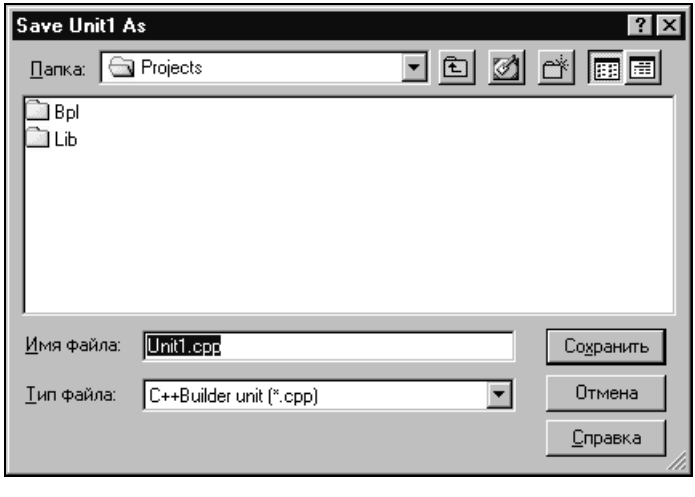


Рис. 11.1. Диалоговое окно сохранения файлов

В этом окне необходимо выбрать место расположения проекта в имеющихся каталогах или создать для этого отдельный каталог (папку). Последний вариант предпочтительнее, поскольку позволяет систематизировать все проекты и устраняет возможность наложения файлов с одинаковыми именами друг на друга. После чего следует последовательно произвести сохранение всех файлов проекта.

Среда разработки предлагает по умолчанию свои имена для сохраняемых файлов. Например, для сохранения файлов с текстами программ предлагается имя Unit1.cpp, Unit2.cpp и т. д. Для сохранения файлов проектов будут предлагаться имена Project1.bpr, Project2.bpr и т. д. Вы вправе менять эти имена на собственные. Это даже полезно делать, поскольку по окончании разработки программы ее исполняемый файл с расширением exe будет иметь имя Project1.exe. Согласитесь, что это имя ничего не говорит о программе, и лучше давать файлам имена, которые дают представление о том, какая программа находится в файле. Например, Edit, Timer, Bloknot и т. п. При переименовании файлов рекомендуется использовать названия, состоящие из букв латинского алфавита. Рассмотрим сохранение проекта на примере простой программы, описанной ранее. Нажмите комбинацию клавиш <Shift>+<Ctrl>+<S> на клавиатуре или соответствующий данной команде значок на панели быстрых кнопок. В открывшемся окне замените имя сохраняемого файла Unit1.cpp на uprog1.cpp и нажмите кнопку **Сохранить**. Далее замените предлагаемое для сохранения имя файла проекта Project1.bpr на Prog1.bpr и вновь нажмите кнопку **Сохранить**. Теперь запустите программу на выполнение с помощью команды **Run** из главного меню. После того как на экране появится окно программы, сверните его. Вы увидите, что в свернутом виде программа имеет название Prog1, а не Project1, как раньше. Таким образом можно давать программе осмысленные имена. Завершите работу приложения (программы). Теперь, если вы просмотрите содержимое каталога, в котором сохраняли файлы проекта, с помощью, например, Проводника (**Explorer**) из среды Windows, то обнаружите в этом каталоге файлы, в том числе и с именами, которые вы задали.

Ценными файлами в этой группе являются файлы с текстами программ с расширением cpp, заголовочные файлы с расширениями h и hpp, файл установок формы проекта с расширением dfm, файл ресурсов с расширением res и файл проекта с расширением bpr, bpg или bpk. Файлы с расширениями ~bpr, ~dfm, ~cpp, ~h, obj, tds и exe создаются в момент компиляции проекта и могут быть восстановлены после удаления в любое время.

Наибольшим объемом обладает файл с расширением tds, предназначенный для отладки программы. Со временем у вас будет расти количество проектов, поэтому для их хранения может понадобиться значительный объем места на жестком диске. Для рационального использования места на диске не рекомендуется хранить вспомогательные файлы проекта, а обходиться

хранением только кодов программ, файлов проектов, исполняемого файла и файлов рисунков, иконок, фонов, курсоров, звуковых файлов и пр.

Для облегчения данной задачи я предлагаю вам использовать командный файл очистки проекта от всех вспомогательных файлов. Листинг (текст) данного командного файла приведен ниже.

Листинг 11.1. Командный файл очистки проекта

```
del *.~*
del *.obj
del *.tds
```

Как видите, он состоит всего из трех строчек с командами удаления вспомогательных файлов с некоторыми расширениями. Создайте с помощью любого текстового редактора файл с именем bcbclr.bat и занесите в него данные строки команд. После чего поместите этот файл в каталог с файлами проекта, закройте среду разработки Borland C++ Builder 6 и запустите файл bcbclr.bat на выполнение. В результате каталог будет очищен от лишних файлов и освободит тем самым немало места на жестком диске.

Теперь рассмотрим, как можно открыть сохраненный проект для продолжения работы с ним или для его корректировки. Запустите среду разработки Borland C++ Builder 6. Вызовите из главного меню команду **Open Project** (Открыть проект) или нажмите комбинацию клавиш <Ctrl>+<F11> на клавиатуре. На экране отобразится стандартное диалоговое окно открытия файлов (рис. 11.2).

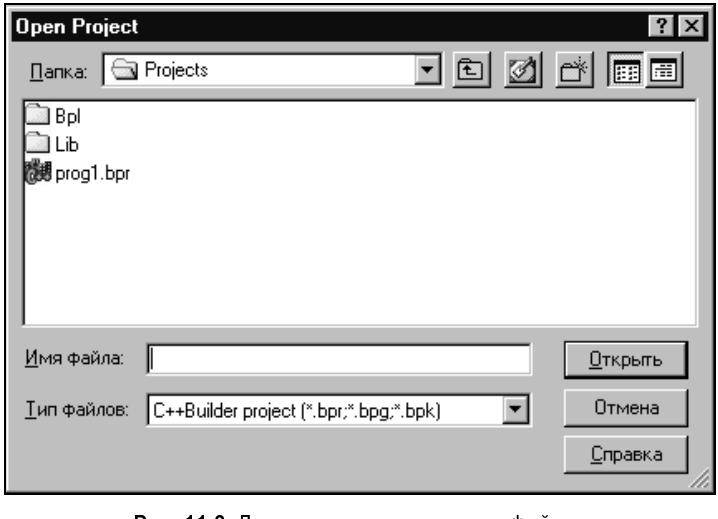


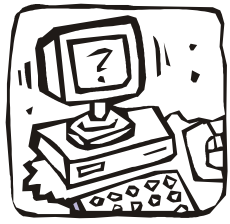
Рис. 11.2. Диалоговое окно открытия файлов

В данном окне необходимо выбрать место расположения сохраненного проекта в имеющихся каталогах и выделить имя проекта. После чего следует нажать кнопку **Открыть**. Перед вами появится проект в том виде, в котором вы закончили с ним работу перед сохранением. Аналогично можно открыть отдельно любой файл проекта, используя команду **Open** главного меню.

Для закрытия всего проекта или файла без сохранения необходимо использовать команду **Close All** (Закрыть все) или **Close** (Закрыть) главного меню, соответственно.

Кроме того, интерфейс среды разработки Borland C++ Builder 6 запоминает пути хранения нескольких проектов и позволяет быстро загрузить их без поиска по каталогам с помощью команды **Reopen** главного меню.

Для практики сохраните, а затем откройте проект описанными выше способами. Создайте файл очистки bcbclr.bat и запустите его для удаления вспомогательных файлов.



Глава 12

Вывод на принтер

Здесь говорится о том, как в среде разработки подготовить и распечатать на принтере текстовый файл или графический образ.

При разработке программ вам может потребоваться распечатать какой-либо файл. Для этого необходимо предварительно выделить этот файл и выполнить настройки для печати. Выделите небольшой блок в файле инспектора кода и выполните команду **File Print** главного меню. При этом откроется окно (рис. 12.1), в котором производится настройка параметров печати.

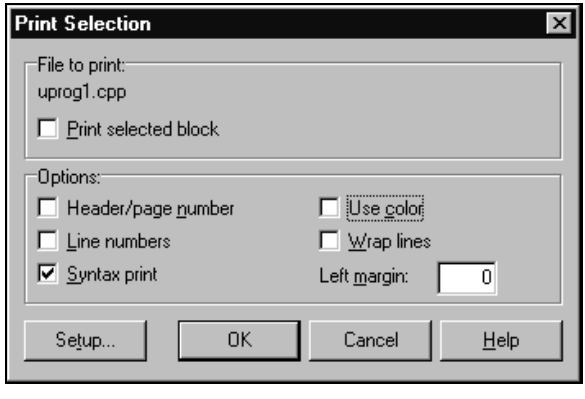


Рис. 12.1. Окно настройки параметров печати

В верхней части окна находится группа под названием **File to print**, в которой отображается имя выделенного файла для печати. Если в файле предварительно был выделен блок, то становится доступной печать только данного блока путем установки флажка перед строкой **Print selected block**.

Под группой выбора модулей для печати **File to print** располагается группа установок параметров печати специальных полей **Options**. При установке флажка перед строкой **Header/page number** вы обеспечиваете автоматическую

нумерацию страниц текста при печати. Флажок, установленный перед строкой **Line numbers**, обеспечивает нумерацию строк печатаемого текста. Установка флажка перед строкой **Syntax print** позволяет печатать жирный шрифт, курсив, символы подчеркивания и другие синтаксические выделения шрифтом. Для использования цвета необходимо установить флажок перед строкой **Use color**. Для переноса строк, выходящих за пределы правой границы листа, необходимо установить флажок перед строкой **Wrap lines**. Поле **Left margin** позволяет задать отступ слева.

Для установки дополнительных параметров печати необходимо нажать кнопку **Setup**. При этом откроется окно настройки принтера (рис. 12.2). Здесь можно осуществить стандартные настройки принтера и выбора формата бумаги для печати.

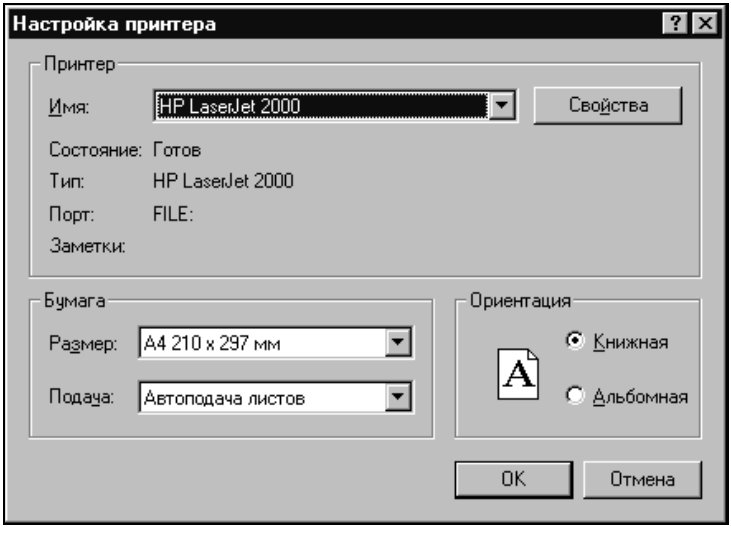


Рис. 12.2. Окно настройки принтера и выбора формата бумаги для печати

Кроме текстовых файлов в среде разработки Borland C++ Builder 6 можно распечатать и графические образы. Например, форму приложения. Для этого необходимо открыть форму в инспекторе форм и выполнить команду **File Print** главного меню. При этом откроется окно, подобное приведенному на рис. 12.3.

Это диалоговое окно позволяет задать масштабирование формы при печати. Масштабирование зависит от размера бумаги. Окно имеет три опции масштабирования:

- **Proportional** — позволяет осуществить пропорциональное масштабирование, то есть распечатать форму в том масштабе, как она видна на экране;

- **Print to fit page** — позволяет растянуть печатаемую форму на всю ширину страницы;
- **No scaling** — никак не масштабирует форму и при печати использует настройки режима экрана. Если вы выберете эту опцию, форма может напечататься на нескольких страницах. При распечатке формы ее заголовок не печатается.

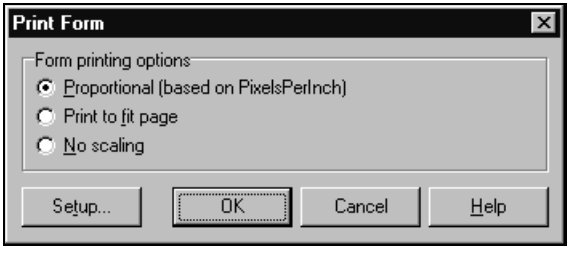
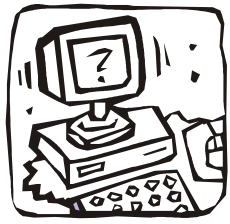


Рис. 12.3. Окно масштабирования формы при печати

Попробуйте распечатать одну и ту же форму при трех различных установках масштабирования, и вы сразу же поймете, что это за режимы.



Глава 13

Отладка программ

При разработке программ, особенно в первое время, возможны ошибки как синтаксические, так и логические. Чем сложнее программа, тем труднее их обнаружить. Для облегчения обнаружения и устранения ошибок в среде разработки Borland C++ Builder 6 существует встроенный отладчик программ. Он позволяет выполнять программу по шагам, устанавливать точки останова, просматривать переменные и т. п.

Несмотря на то, что созданные нами программы очень просты, я намеренно хочу рассказать об отладке программ. Дело в том, что рано или поздно в создаваемой программе могут появиться ошибки: или синтаксические, или более сложные для обнаружения логические ошибки. И тогда потребуется инструмент для их устранения. Чем быстрее мы познакомимся с таким инструментом, тем легче нам будет создавать более сложные программы.

После разработки любого приложения необходимо выполнить его компиляцию, компоновку и тестирование. Эти операции выполняются автоматически каждый раз при выполнении команды **Run** из главного меню среды разработки Borland C++ Builder 6. Однако данные процедуры можно выполнять по отдельности с целью сокращения времени на отладку всего проекта в целом. В нашем очередном примере мы будем пользоваться различными командами трансляции (компиляции и компоновки) проекта. Рассмотрим, в чем состоит отличие между этими командами. Например, компиляцию отдельного модуля программы можно выполнить с помощью команды главного меню **Project Compile**. При этом не будет затрачено время на компиляцию и компоновку всего проекта в целом. Команда главного меню **Project Make** позволяет выполнить компиляцию только тех модулей, которые были отредактированы, после чего выполняет компоновку проекта с созданием исполняемого модуля с расширением exe. Таким образом, эта команда также экономит время на трансляцию проекта в целом. И только команда **Project Build** компилирует все модули независимо от того, редактировались они или нет. А затем компонует проект и готовит исполняемый

файл с расширением exe. Данная команда необходима для переработки всего проекта после изменения настроек компилятора или среды разработки в целом. Команда **Project Run** в отличие от **Project Build** автоматически запускает исполняемый файл.

На этапе компиляции происходит автоматический поиск синтаксических ошибок, неправильного объявления или использования переменных и т. п. При обнаружении подобных ошибок в инспекторе кодов будет выведено соответствующее сообщение об ошибке или предупреждение. Предупреждение в отличие от ошибки не блокирует выполнение программы, но не гарантирует корректность ее работы. Если ошибок в программе нет, компилятор создает объектный (машинный) код программы в модулях (файлах) с расширением obj.

При компоновке программы также выявляются некоторые виды ошибок, связанные с отсутствием объявления переменных, функций, библиотек и т. п., и выполняется объединение всех объектных модулей и библиотек при необходимости в один исполняемый файл с расширением exe. Вместо исполняемого файла может создаваться библиотечный файл с расширением lib или dll при соответствующих установках среды разработки. Рассмотрим выполнение описанных процедур на конкретном примере. Создадим небольшое приложение на форме **Form1** с двумя кнопками **Button1**, **Button2** и одним элементом надписи **Label1**. О том, как размещаются эти элементы на форме, рассказывалось в *главе 7*. Это приложение должно сообщать нам о количестве нажатий на первую кнопку. При нажатии на вторую кнопку приложение должно закрываться. Разместите эти элементы на форме **Form1** и измените размер формы в соответствии с рис. 13.1.

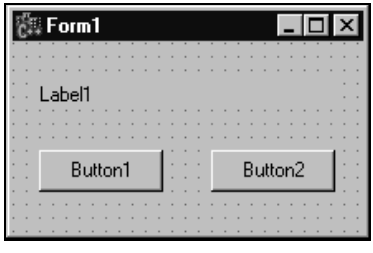


Рис. 13.1. Внешний вид формы **Form1**

Теперь замените свойство **Caption** всех объектов приложения в инспекторе объектов на заголовки **Программа 2**, **Кнопка 1**, **Выход** и **пустую строку** соответственно. В результате у вас должно получиться окно, показанное на рис. 13.2. Обратите внимание, что элемент **Label1**, располагающийся выше кнопок, не виден, поскольку мы изменили его свойство **Caption** на **пустую строку**, то есть очистили это свойство.

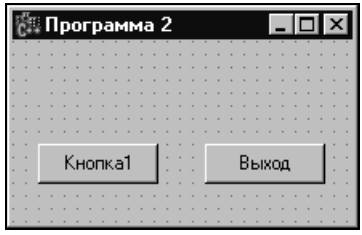


Рис. 13.2. Внешний вид измененной формы **Form1**

Щелкните дважды левой кнопкой мыши по созданной кнопке **Выход** и в отрывшемся окне инспектора кодов впишите между фигурными скобками заготовки обработчика события, команду закрытия приложения `Close();`. Вернитесь к окну формы и щелкните дважды левой кнопкой мыши по кнопке с названием **Кнопка1**. В отрывшемся окне инспектора кодов впишите между фигурными скобками следующую строку:

```
Label1->Caption="Кнопка 1 нажата " + IntToStr(++i) +" раз";
```

В этой строке команд выполняется присвоение (знак "равно") свойству `Caption`, элемента надписи `Label1`, текстовой строчки, состоящей из трех слагаемых частей. Поскольку свойство `Caption` элемента надписи `Label1` предназначено для отображения текста, мы должны присваивать этому свойству только текстовые (строковые) значения. В языке C++ такие строковые значения заключаются в кавычки. Первая и последняя части присваиваемого значения таковыми и являются. Счетчиком количества нажатий на кнопку в программе будет служить переменная `i`, которая должна автоматически увеличиваться на единицу перед выводом. Для этой цели перед ней записаны два знака `+`. Данная операция в языке C++ называется автоинкрементом. Для превращения числовой переменной `i` в строковую переменную используется встроенная функция C++ преобразования целых чисел в строки `IntToStr()`. Итак в одной строке команд мы осуществили целый ряд операций. Разве это не изящный язык программирования?!

Сохраните проект под именем butct.bpr, а программный модуль под именем Ubutct.cpp. Впрочем, имена вы можете дать любые другие. Не допускается лишь использование в этих именах русских букв. И не изменяйте расширения файлов.

Теперь попробуем скомпилировать, скомпоновать и выполнить данное приложение, а заодно проверить, нет ли в нем ошибок. Выполните команду **Compile** из группы **Project** главного меню или нажмите комбинацию клавиш для компиляции программы <Alt>+<F9>. Перед вами откроется окно, приведенное на рис. 13.3.

В верхней строке этого окна отображается путь размещения и имя проекта программы. В следующей строке вначале отображается процесс компиляции, сменяющийся записью **Done**, говорящей о завершении данной операции,

и сообщением об обнаруженных ошибках **There are errors** (Здесь есть ошибки). Ниже отображается номер текущей строки программы **Current line** и общее количество строк программы **Total lines**. В нижней строке отображается обнаруженное на данный момент число замечаний **Hints**, предупреждений **Warnings** и ошибок **Errors**. Как видим, в программе есть одна ошибка. Нажмите кнопку **OK** — и перед вами окажется окно инспектора кода с выделенной строкой, имеющей ошибку, и сообщением о типе ошибки (рис. 13.4).

Рис. 13.3. Окно компиляции программы

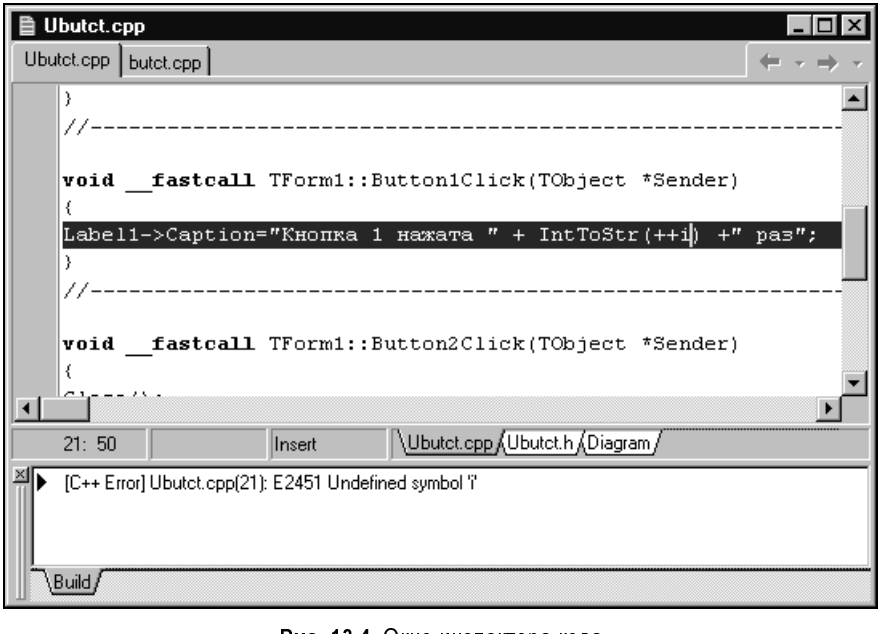


Рис. 13.4. Окно инспектора кода

Строка *[C++ Error] Ubutct.cpp(21): E2451 Undefined symbol 'i'* сообщает о том, что компилятор C++ обнаружил ошибку в 21-й строке модуля Ubutct.cpp. Код ошибки E2451 означает, что обнаружен необъявленный символ. В нашем случае это переменная с именем `i`. Действительно, мы не объявили ее в программе, хотя и сделали это намеренно, с целью получения сообщения об ошибке в качестве примера. В языке C++, как впрочем и в других языках программирования, все переменные, используемые в программе, необходимо объявлять. Это делается для того, чтобы компилятор знал, сколько ячеек памяти необходимо резервировать в памяти компьютера для ее хранения. Поскольку различные типы переменных требуют для своего хранения различное количество ячеек памяти (байт), рационально резервировать для них лишь необходимый объем памяти. В нашем случае переменная `i` является целой, то есть принимающей только целочисленные значения. Для ее объявления необходимо использовать директиву **int** (от integer, целый). Можно сделать такое объявление непосредственно в тексте программы обработчика событий, но тогда эта переменная будет недоступна в других функциях обработчика. Поэтому мы объявим ее в блоке **public** файла описания заголовков Ubutct.h. Для этого необходимо щелкнуть левой кнопкой мыши по вкладке **Ubutct.h** инспектора кодов и вписать строку объявления переменной с комментариями сразу же после строчки `public`. Ниже, в листинге 12.1, эта запись приведена полностью.

Листинг 12.1. Строки объявления переменной `i`

```
public: // User declarations
int i; // Переменная - счетчик
```

Теперь снова выполним компиляцию и обнаружим, что ошибок нет. Но не спешите радоваться. Дело в том, что мы не присвоили переменной `i` начального значения. Компилятор не обращает на это внимания и не выдает никаких сообщений по этому поводу. Но в программе это может привести к недоразумениям. Поэтому выполним присвоение начального значения переменой `i`, сделав запись `i=0;` (ведь при запуске программы **Кнопка 1** не была нажата ни разу) в тексте программы обработчика создания формы. Для этого щелкните дважды левой кнопкой мыши по форме приложения и в открывшемся окне инспектора кода впишите между фигурными скобками текста программы строку `i=0;`.

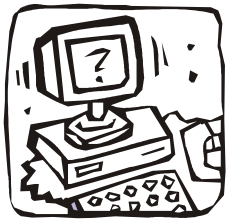
Выполним компоновку проекта с помощью команды **Make** из группы **Project** главного меню или нажав комбинацию клавиш для компоновки программы <Ctrl>+<F9>. В результате мы должны получить безошибочную компоновку приложения. Теперь приложение можно сохранить и запустить на выполнение с помощью команды **Run** или воспользоваться горячей кнопкой запуска программы <F9>.

На экране появится окно нашей программы. Щелкая левой кнопкой мыши по кнопке с названием **Кнопка 1**, вы увидите, что каждый раз в окне будет появляться сообщение о количестве нажатий этой кнопки.

Внимание

Каждый раз перед запуском приложения после редактирования обязательно сохраняйте проект. Это позволит избежать потери информации при зависании программы, вызванном наличием в ней ошибок.

Исполняемый файл программы вместе с файлами проекта находится на прилагаемом к книге диске. Потренируйтесь, специально внося ошибки в программу и изучая сообщения об ошибках, возникающие при компиляции и компоновке проекта.



Глава 14

Структура файлов

В этой главе описывается структура файлов проекта. На первый взгляд этот материал может показаться очень трудным. Но со временем описанные директивы станут привычными.

Теперь рассмотрим более подробно структуру основных файлов программы, для того чтобы не теряться при возникновении сложных ошибок и иметь представление о том, как работает программа. Вернемся к нашему примеру и рассмотрим состав файлов проекта. Для лучшего понимания текста программ тем, кто не знаком с языками программирования, я рекомендую познакомиться с основами языка программирования C++, например, по книге[1]. В языке программирования C++ главной функцией проекта является функция `main`. В программах, написанных для операционной системы Windows, главной функцией является `WinMain`. Данная функция содержится в программном модуле butct.cpp нашего примера. В листинге 14.1 приведено его содержимое.

Листинг 14.1. Программный модуль butct.cpp

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------
USEFORM("Ubutct.cpp", Form1);
//---------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
```

[1] Подбельский В. В. Язык СИ++. — М.: Финансы и статистика, 2003.

```
                Application->Initialize();
                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;
}
//---------------------------------------------------------------------------
```

В этих нескольких строках содержатся директивы и команды, с помощью которых формируется программа. Строки, начинающиеся с символов `//`, являются строками комментариев для разделения или снабжения текстовым описанием частей программы. В языке Си допускается также заключать комментарии между символами `/*` и `*/`. Строка `#include <vcl.h>` является строкой препроцессора, то есть программы, которая будет подключать к нашим модулям дополнительные файлы библиотек и описаний. В данном случае, строка `#include <vcl.h>` сообщает препроцессору о том, что ему необходимо подключить к нашей программе заголовочный файл vcl.h, в котором содержатся описания визуальных компонентов Borland C++ Builder 6. Директива `#pragma hdrstop` завершает список файлов заголовка для препроцессора. Строка макрокоманды `USEFORM("Ubutct.cpp", Form1);` подключает к проекту модуль Ubutct.cpp и форму `Form1`. Далее следует заголовок программы главной функции проекта `WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)`. Перед названием главной функции находится дескриптор `WINAPI`, который позволяет использовать множество готовых, полезных функций Windows в нашем приложении. В скобках заключены параметры главной функции, необходимые для опознания нашего приложения среди множества других работающих приложений, для передачи параметров в программу в командной строке и для определения конфигурации окна нашего прило-

жения. После заголовка главной функции следует блок кода, в котором может произойти сбой программы — аварийная ситуация (останов, ошибка программы и пр.). Такой блок должен начинаться с ключевого слова `try`. В этом блоке заключены операторы инициализации компонентов приложения `Application->Initialize();`, создания формы приложения `Application->CreateForm(__classid(TForm1), &Form1);` и выполнения приложения `Application->Run();`.

Блок `catch (Exception &exception)` предназначен для включения операторов, вызываемых при возникновении аварийной ситуации. В нашем примере программы таким оператором является стандартный обработчик исключений `Application->ShowException(&exception);`. В следующих примерах мы рассмотрим, как его использовать. Аналогичная конструкция блоков повторяется для копии приложения, запущенного вторично. Последним оператором тела функции `WinMain` является `return 0;` — оператор, завершающий приложение с кодом 0.

Данный файл можно редактировать, изучив предварительно команды и функции Windows и получив некоторый опыт разработки программ. Поскольку этот файл создается средой разработки автоматически, мы в нашем курсе обучения не будем его изменять намеренно, чтобы не отягощать себя лишними проблемами. Тем не менее, при возникновении ошибок компилятора или компоновщика, не лишним будет посмотреть содержимое данного файла и отредактировать его в случае необходимости. Например, при создании новой формы и последующем ее удалении в этом файле могут сохраниться записи о несуществующей форме, приводящие к ошибкам во время трансляции. Для устранения подобных ошибок необходимо удалить лишние записи из этого файла.

Рассмотрим структуру файла butct.cpp. Содержимое этого файла целиком приведено в листинге 14.2.

Листинг 14.2. Содержимое файла butct.cpp

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

#include "Ubutct.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
```

```
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Label1->Caption="Кнопка 1 нажата " + IntToStr(++i) +" раз";
}
//---------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
i=0;
}
//---------------------------------------------------------------------
```

Большинство строк этого файла нам уже известно из предыдущего описания и самого примера программы. Рассмотрим лишь неизвестные строки.

Строка `#pragma package(smart_init)` определяет последовательность инициализации составных частей программы. Строка `#pragma resource "*.dfm"` сообщает препроцессору о том, что для формы необходимо использовать файл `*.dfm` с именем данного файла. В строке `TForm1 *Form1;` объявляется класс `TForm1` для формы `Form1`. Далее следуют подпрограммы создания формы приложения и обработчики событий, рассмотренные ранее.

Обратите внимание на строку `#include "Ubutct.h"`, которая подключает к модулю заголовочный файл Ubutct.h, представленный в листинге 14.3.

Листинг 14.3. Заголовочный файл Ubutct.h

```
//---------------------------------------------------------------------
#ifndef UbutctH
#define UbutctH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
```

```
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TButton *Button1;
        TButton *Button2;
        TLabel *Label1;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
private:        // User declarations
public:         // User declarations
        int i;  // Переменная - счетчик
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Ранее мы уже редактировали его в примере, вставляя строку инициализации переменной `i`. Рассмотрим этот файл детальнее. Строка `#ifndef UbutctH` является директивой условной компиляции. Эта директива проверяет, не был ли ранее определен идентификатор `UbutctH` директивой `#define`. Если не был, то предваряемый директивой фрагмент кода компилируется. В противном случае компиляция фрагмента не производится. Затем строка `#define UbutctH` определяет идентификатор `UbutctH`. Далее следуют четыре строки `#include` включения файлов, где описаны компоненты, переменные, константы и функции, которые будут использованы в данном модуле. Сюда можно добавлять дополнительные директивы, не создаваемые средой разработки автоматически. После этих строк следует описание класса формы с тремя разделами:

- `__published:` — содержит объявления размещенных на форме компонентов и обработчиков событий, которые среда разработки записывает в файл автоматически в процессе проектирования формы;
- `private:` — предназначен для объявления закрытых переменных, то есть доступных только для данного модуля;
- `public:` — предназначен для объявления открытых переменных, то есть доступных не только для данного модуля, но и для других классов и модулей. Именно в этот раздел мы включили (записали) переменную — счетчик `i`.

Строка `extern PACKAGE TForm1 *Form1;` объявляет класс `TForm1` как доступный из других модулей. И завершает файл строка `#endif` окончания условной директивы `#ifndef`.

Следующие два файла `butct.bpr` и `Ubutct.dfm` представляют собой текстовые файлы установок проекта и формы данного проекта соответственно. Они создаются и редактируются автоматически в процессе разработки и изменения проекта в среде разработки. Файл `butct.bpr` содержит директивы и макрокоманды настроек среды разработки и его не следует корректировать вручную для избежания внесения ошибок, хотя просмотреть содержимое можно с помощью любого текстового редактора. Файл `Ubutct.dfm` представляет собой текстовые записи содержания и свойств формы. Его можно просмотреть и даже отредактировать в самой среде разработки. Для этого необходимо щелкнуть левой кнопкой мыши на форме для ее выделения, а затем щелчком правой кнопки открыть контекстное меню (рис. 14.1) и выбрать в нем команду **View as Text**.

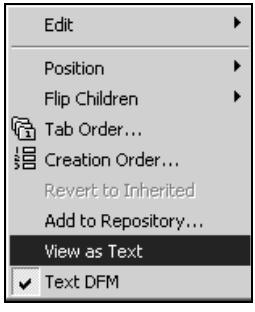


Рис. 14.1. Контекстное меню

Перед вами должно открыться окно (рис. 14.2), в котором представлено содержимое файла формы в текстовом виде.

Здесь можно просмотреть и отредактировать при необходимости любую запись. Например, в строках `Left = 192`, `Top = 107`, `Width = 220` и `Height = 140` можно изменить левый и верхний отступ, а также ширину и высоту формы соответственно. В строчках `Caption = 'Программа 2'` и `Color = clBtnFace` можно изменить, соответственно, заголовок формы и цвет ее фона. Вернуться к исходному виду формы можно с помощью комбинации клавиш <Alt>+<F12> или через контекстное меню по команде **View as Form**.

Файлы `butct.res` и `Ubutct.ddp` являются двоичными файлами, хранящими ресурсы и диаграммы проекта соответственно, и не подлежат ручному редактированию. Исполняемый файл проекта имеет имя `butct.exe`. Наконец, файл с расширениями tds является отладочным файлом проекта, который можно всегда создать в среде разработки повторно, имея исходные файлы

проекта. Помимо описанных файлов среда разработки позволяет создавать дополнительные файлы, которые являются вспомогательными и не подлежат ручной корректировке.

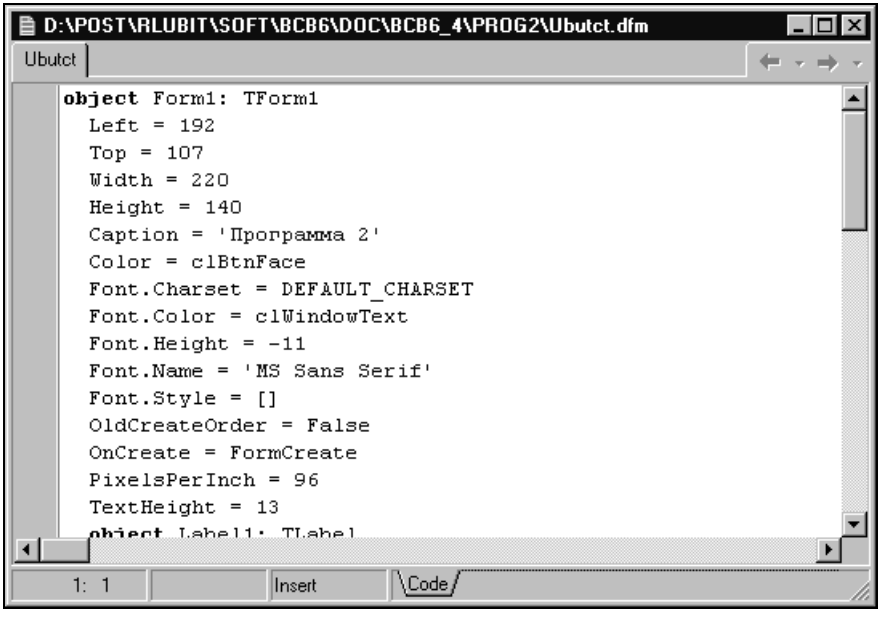
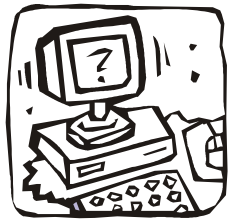


Рис. 14.2. Окно формы



Глава 15

Инструменты отладчика

В этой главе мы рассмотрим инструменты отладчика, которые очень помогут нам при отладке программ.

Основными инструментами отладки любой программы являются команды выполнения программы по шагам и установки точек останова. Естественно, что такие команды имеются и в среде разработки Borland C++ Builder 6. Команды пошагового выполнения программы вызываются из главного меню в группе **Run** или с помощью горячих клавиш. Список этих команд, их соответствие горячим клавишам и описание назначения приведен в табл. 15.1.

Таблица 15.1. *Список пошагового выполнения программы, соответствие команд горячим клавишам и описание назначения*

Команда	Горячие клавиши	Назначение
Step Over	<F8>	Пошаговое выполнение программы без захода в вызываемые функции и подпрограммы
Trace Into	<F7>	Пошаговое выполнение программы с заходом в вызываемые функции и подпрограммы
Trace to Next Source Line	<Shift>+<F7>	Перейти к следующей исполняемой строке программы
Run to Cursor	<F4>	Выполнить команды до места расположения курсора
Run Until Return	<Shift>+<F8>	Выполнить команды до выхода из выполняемой функции и остановиться на операторе, следующем за этой функцией

Команды установки точек останова и некоторые другие вызываются из контекстного меню инспектора кода через раскрывающийся список команды **Debug** (рис. 15.1), а также с помощью горячих клавиш. Контекстное меню вызывается щелчком правой кнопки мыши, когда курсор находится на окне инспектора кода.

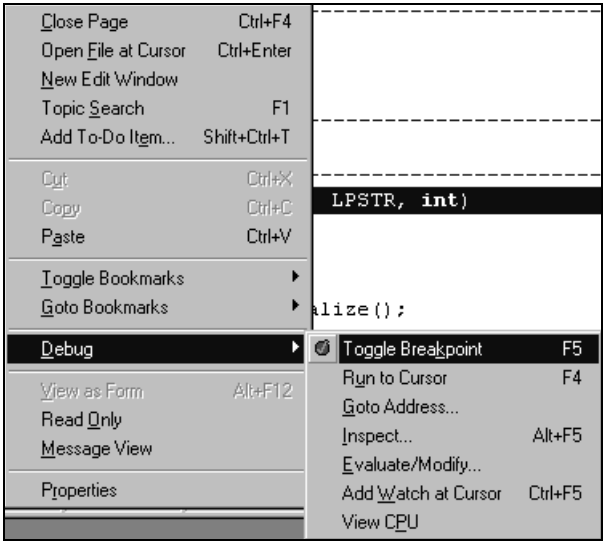


Рис. 15.1. Установка точки останова

Попробуйте применить эти команды на практике, используя программу, приведенную в листинге 14.2. Поместите курсор в инспекторе объектов на строку:

```
Label1->Caption="Кнопка 1 нажата " + IntToStr(++i) +" раз";
```

и нажмите клавишу <F5>. При этом строка окрасится в красный цвет, что свидетельствует о том, что на ней установили точку останова. Теперь при выполнении программы произойдет автоматический останов на этой строке. Проверьте это, выполнив команду **Run** с помощью клавиши <F9>. Программа запустится и на экране появится ее форма. Но стоит щелкнуть левой кнопкой мыши по кнопке с названием **Кнопка 1**, программа остановится и откроется окно (рис. 15.2) Инспектора кода с зеленой стрелкой напротив строки, на которой установлена точка останова.

Теперь, нажимая клавишу <F8>, можно продолжить выполнение программы по шагам. Для прерывания работы программы необходимо нажать комбинацию клавиш <Ctrl>+<F2>. Можно продолжить выполнение программы с помощью клавиши <F9>, а затем щелкнуть кнопку **Выход** в окне про-

граммы для ее завершения. Попробуйте на практике выполнить и другие команды отладчика, для того чтобы понять их назначение.

В *главе 16* мы познакомимся с другими возможностями отладчика.

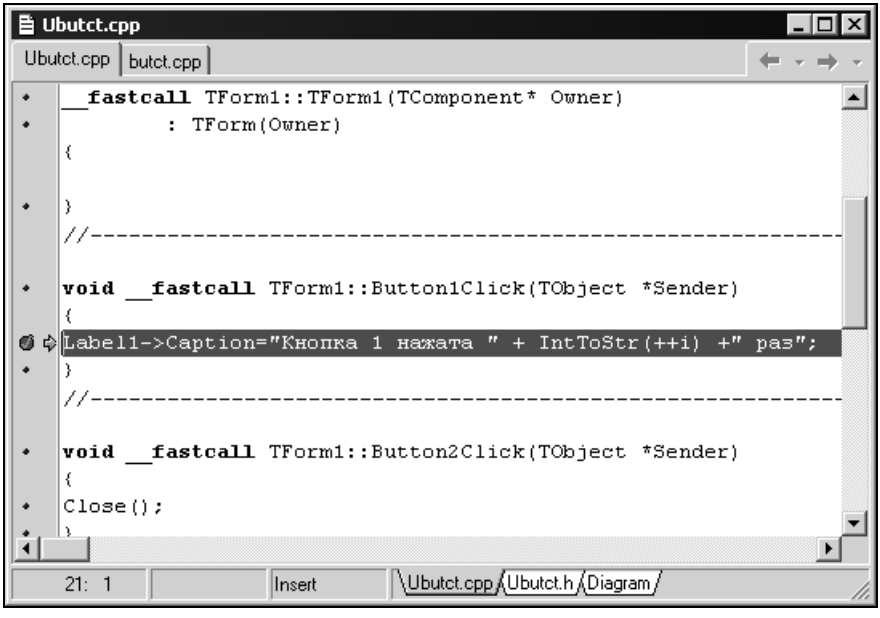
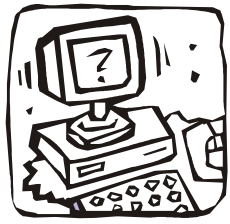


Рис. 15.2. Окно инспектора кода



Глава 16

Окна отладчика

Помимо команд отладки среда разработки позволяет просмотреть переменные, используемые в программе, состояние регистров процессора, процессы, происходящие в программе, и т. п. Для этих целей используются окна отладчика.

Доступ к окнам отладчика производится через пункт главного меню **View | Debug Windows**. При этом откроется окно (рис. 16.1), в котором расположены команды отладчика.

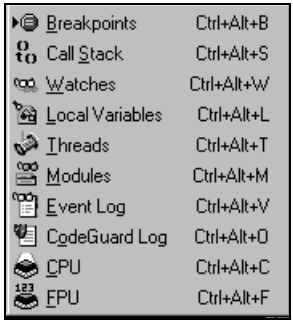


Рис. 16.1. Окно команд отладчика

Команда **Breakpoints** открывает окно (рис. 16.2), в котором отображаются и редактируются точки останова.

Из этого окна через контекстное меню, вызываемое щелчком правой кнопки мыши, можно осуществлять добавление, удаление и редактирование точек останова (рис. 16.3).

Рис. 16.2. Окно точек останова

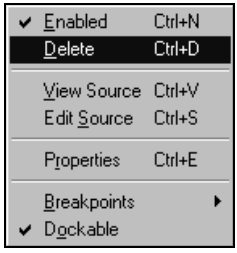


Рис. 16.3. Контекстное меню

Для вызова команды **Breakpoints** отладчика можно так же использовать горячие клавиши <Ctrl>+<Alt>+<B>. Остальные команды можно вызывать аналогично. Комбинация горячих клавиш для каждой из команд изображена правее самой команды (см. рис. 16.1).

Команда **Call Stack** вызывает окно просмотра стека программы (рис. 16.4). Данное окно позволяет увидеть вызовы функций, производимые в программе в момент ее отладки.

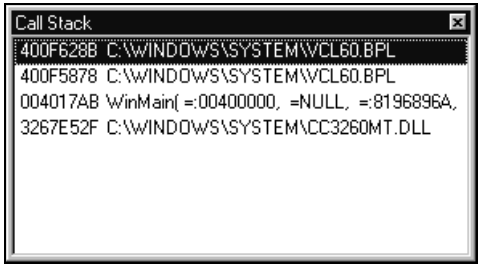


Рис. 16.4. Окно просмотра стека программы

Команда **Watches** открывает окно переменных (рис. 16.5), в котором можно просматривать переменные, используемые в программе.

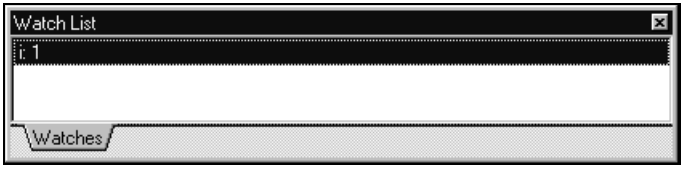


Рис. 16.5. Окно переменных

Из окна переменных щелчком правой кнопки мыши можно вызвать контекстное меню (рис. 16.6), с помощью которого появляется возможность редактировать, добавлять, активизировать, делать невидимыми и удалять переменные из окна отображения переменных.

Рис. 16.6. Контекстное меню окна переменных

Окно редактирования свойств переменных **Watch Properties** (рис. 16.7) вызывается из контекстного меню командой **Edit Watch** или с помощью комбинации клавиш <Ctrl>+<E>. В этом окне можно выбрать (**Expression**) отображаемые переменные, разбить переменные на группы (**Group name**), задать количество элементов массива (**Repeat count**) для отображения, ограничить разрядность (**Digits**) отображаемых значений, разрешить (**Enabled**) отображение переменной, разрешить (**Allow Side Effects**) отображение выражений и выбрать один из девяти форматов отображения переменных.

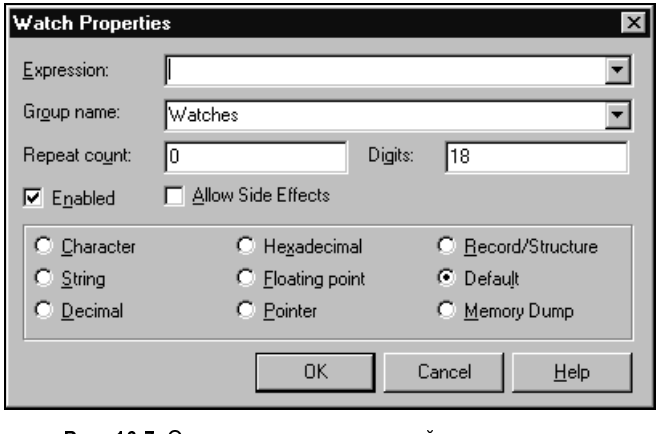


Рис. 16.7. Окно редактирования свойств переменных

Команда **Local Variables** показывает текущие локальные переменные в отладочном режиме работы (рис. 16.8).

Рис. 16.8. Окно локальных переменных

Команда **Threads** отображает окно процессов, происходящих в режиме отладки программы (рис. 16.9). В частности, в нем можно увидеть имя исполняемой программы, состояние процесса отладки и адрес точки останова.

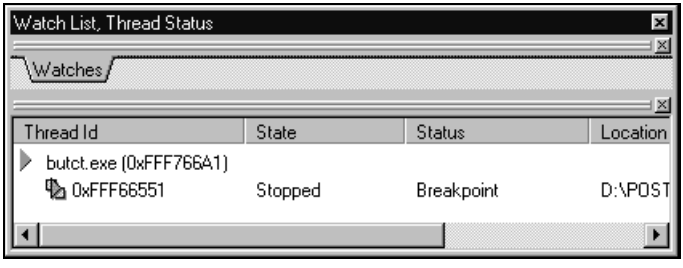


Рис. 16.9. Окно процессов отладки

Команда **Modules** вызывает окно (рис. 16.10), в котором можно просмотреть все программные модули, их адреса и точки входа.

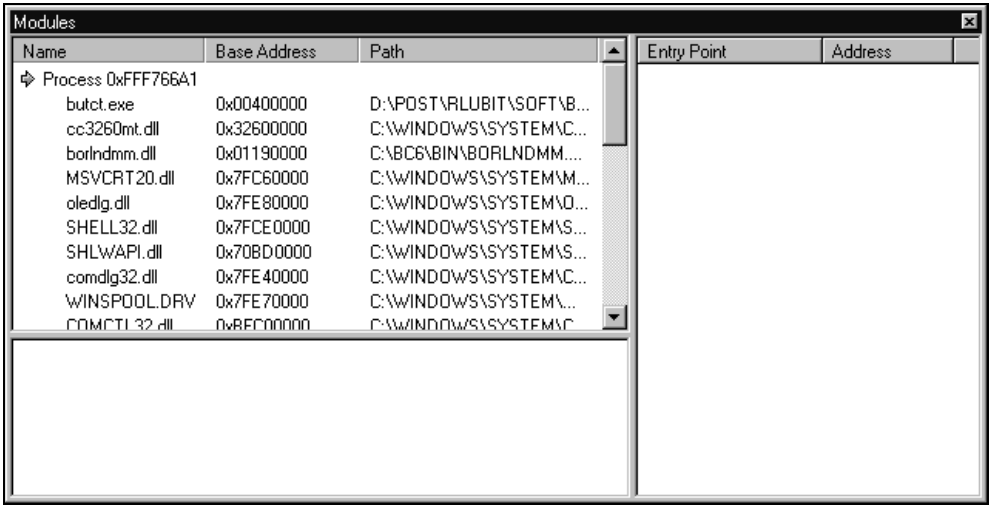


Рис. 16.10. Окно программных модулей

Команда **Event Log** отображает окно регистрации событий программы (рис. 16.11).

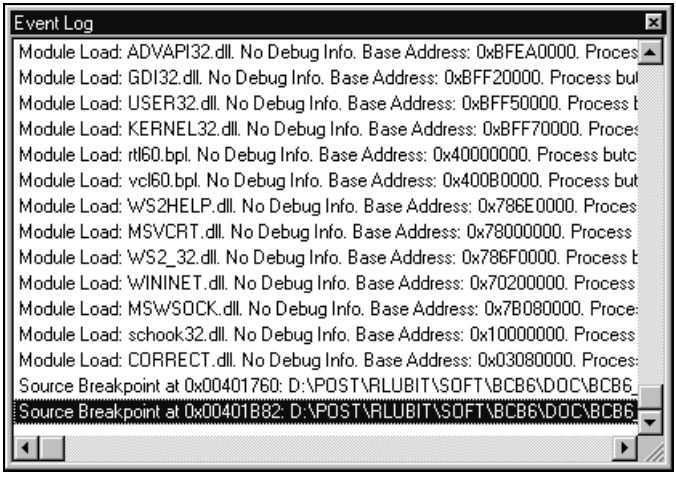


Рис. 16.11. Окно регистрации событий программы

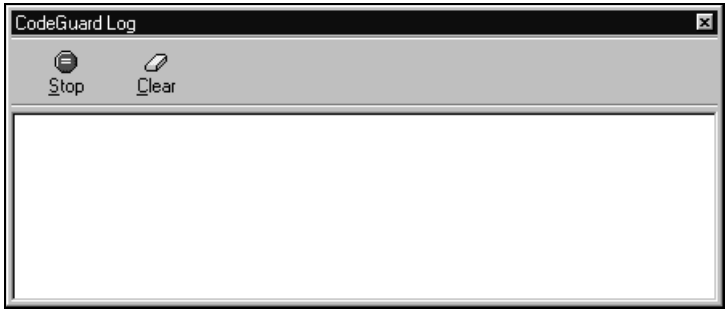


Рис. 16.12. Окно сообщений **CodeGuard Log**

Команда **CodeGuard Log** вызывает окно сообщений (рис. 16.12), сгенерированных специальной утилитой отладки **CodeGuard**.

С помощью команды **CPU** вызывается окно (рис. 16.13), в котором отображается работа процессора в машинных кодах и содержимое регистров процессора. Если программа не запущена, данная и следующая команды недоступны.

Наконец, команда **FPU** вызывает окно (рис. 16.14) для просмотра состояния математического сопроцессора компьютера.

Рекомендую читателям для тренировки и запоминания с помощью горячих клавиш поочередно открыть эти окна во время пошагового выполнения программы.

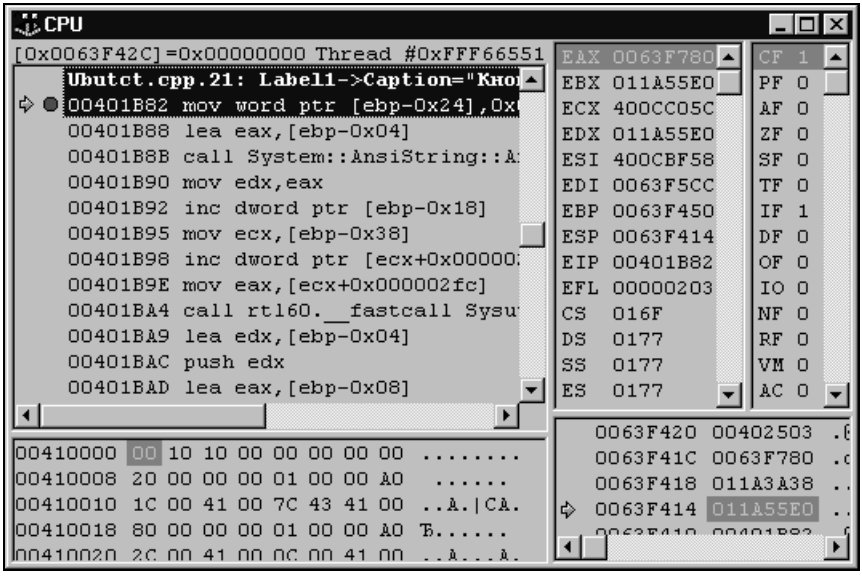


Рис. 16.13. Окно работы процессора

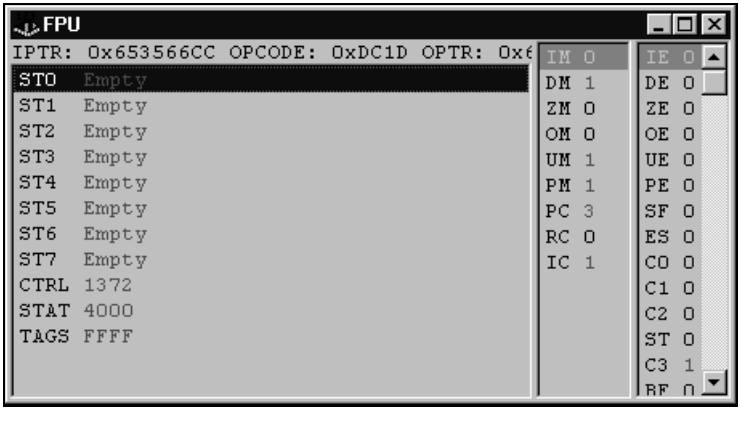


Рис. 16.14. Окно работы математического сопроцессора

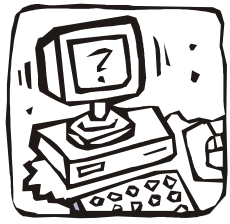
Часть II

Примеры программ

В *части II* этой книги приводятся практические примеры разработки программ различного назначения, их настройки, отладки и выполнения.

Примеры позволяют проследить путь создания программы от идеи до исполняемого файла. Кроме того, эти примеры призваны вызвать у читателя желание создать свою программу.

Между описанием примеров приводится описание свойств, событий и методов компонентов, которые можно использовать при разработке новых программ.



Глава 17

Цифровые часы — календарь

В этой главе рассказывается о том, как с помощью Borland C++ Builder 6 за считанные минуты можно создать полноценную программу цифровых часов — календаря.

Прежде чем продолжать теоретическое изучение Borland C++ Builder 6, создадим на практике полезную программу, которую можно будет использовать для отображения текущей даты и времени. Назовем эту программу "Цифровые часы — календарь".

Запустите Borland C++ Builder 6 и начните создание нового проекта. Поместите на форму **Form1** компонент **Static Text** из вкладки **Additional** визуальных компонентов, щелкнув по нему дважды левой кнопкой мыши. Этот элемент будет служить нам циферблатом часов. В качестве такого элемента можно было бы взять и компонент **Label**, но в отличие от него, **Static Text** обладает более богатыми возможностями оформления (то есть расширенным набором свойств). Изменим некоторые свойства этого компонента. Вначале заменим свойство **Alignment** (Выравнивание) на **taCenter** (по центру), для того чтобы показания часов были в центре окна. Затем установим свойство **BorderStyle** (Стиль границы) в **sbsSunken** (Утопленный) для придания циферблату часов элегантного внешнего вида. Свойство **AutoSize** (Автоматический размер) должно иметь значение **true** (Истина). Теперь раскроем свойство **Font** (Шрифт) щелчком левой кнопки мыши по символу + (плюс) перед ним. Аналогично щелчком левой кнопкой мыши по троеточию напротив свойства **Font** откроем окно для изменения шрифта текста, который будет служить для отображения цифровых показаний времени нашей программы часов (рис. 17.1).

Здесь необходимо выбрать полужирный шрифт **Times New Roman** или любой другой понравившийся вам, установить синий цвет и размер шрифта, равный 30 единицам. Заметьте, что в списке размера шрифта отсутствует размер номер 30. Для того чтобы его установить, необходимо выделить текущий размер шрифта с помощью левой кнопки мыши и ввести с клавиатуры

новый размер. В дальнейшем вы всегда сможете изменить эти установки по своему усмотрению.

Рис. 17.1. Окно для изменения шрифта текста

Теперь очистим свойство **Caption**, чтобы при запуске программы не появлялась надпись **Static Text1** на циферблате программы часов. Компонент **Static Text1** свернется при этом до минимальных размеров. Пусть это не пугает вас. При работе программы он автоматически примет необходимые размеры благодаря его свойству **AutoSize** (Автоматический размер).

Далее поместите на форму компонент таймера **Timer** из вкладки **System**, щелкнув по нему дважды левой кнопкой мыши. Он позволит нам автоматически обновлять показания часов с заданной частотой. Место его размещения не имеет значения, поскольку этот компонент является невидимым и не отображается на экране при работе программы. В дальнейшем мы будем применять в программах и другие подобные скрытые компоненты. Щелкните левой кнопкой мыши по компоненту таймера на форме. В открывшемся окне инспектора кода необходимо написать программу-обработчик события. Ее текст будет состоять из двух строчек (листинг 17.1).

Листинг 17.1. Обработчик события таймера

```
StaticText1->Caption=Now().TimeString();
Form1->Caption=Now().DateString();
```

Первая строка присваивает свойству **Caption** (Заголовок) компонента **StaticText1** значение текущего показания часов. Для этого используется функция извлечения из операционной системы текущего времени **Now** и выделения из него показания часов **TimeString** в строковом формате записи.

Следующая строка выполняет аналогичные действия, но уже для формы, и выделяет при этом показания даты. Свойства компонента таймера должны соответствовать значениям, приведенным в табл. 17.1.

Таблица 17.1. *Свойства компонента* ***Timer***

Свойство	**Значение**
Enabled	true
Interval	1000
Name	Timer1
Tag	0

Данные свойства разрешают автоматическую работу таймера с интервалом (периодичностью) 1000 мс.

Теперь изменим свойства формы. Очистим свойство **Caption**, чтобы при запуске программы не появлялась надпись **Form1** на заголовке окна, и установим свойству **AutoSize** значение **true**. После чего форма свернется до минимальных размеров. Последнее действие мы сделали для того, чтобы размер наших программных часов устанавливался автоматически, в зависимости от шрифта индикаторов.

Сохраните модуль (текст) программы и сам проект под названием utimedate.cpp и timedate.bpr соответственно. Для сохранения проекта создайте новую папку (каталог), например, с именем TIMEDATE. Теперь запустите программу с помощью клавиши <F9>. После трансляции проекта программа должна заработать, и на экране монитора отобразится окно часов, подобное тому, что приведено на рис. 17.2.

Рис. 17.2. Окно часов

Итак, наши часы отображают текущее время на циферблате и дату в заголовке программы. Их можно перемещать в любое место экрана или сворачивать. Если щелкнуть левой кнопкой мыши по кнопке (Развернуть),

то часы переместятся в левый верхний угол экрана, где не будут закрывать собой окна других программ.

Однако если вы запустите новую программу, например Word, и развернете ее во весь экран, то наши часы скроются. Для того чтобы устранить этот недостаток, отредактируем еще одно свойство формы нашей программы. Остановите программу и замените свойство **FormStyle** (Стиль формы) на **fsStayOnTop** (Пребывать наверху). Сохраните проект и вновь запустите программу. Убедитесь, что теперь часы всегда находятся поверх остальных окон других программ и не могут быть скрыты ими. Заметьте, что при создании программ все они имеют значок (иконку) с одним и тем же рисунком в виде квартала синих домов. Конечно, удобно иметь заготовку рисунка для значка новой программы, но согласитесь, если все наши программы будут иметь один и тот же значок, теряется смысл таких значков в качестве пиктограмм. Ведь они предназначены для графического отождествления назначения программы. Поэтому давайте научимся изменять значки для наших программ.

Для этого необходимо воспользоваться встроенным в Borland C++ Builder 6 инструментом **Image Editor** редактора рисунков. Вновь остановите программу часов и щелкните левой кнопкой мыши по кнопке **Tools** главного меню. В открывшемся списке запустите графический редактор **Image Editor**. На экране появится окно, приведенное на рис. 17.3.

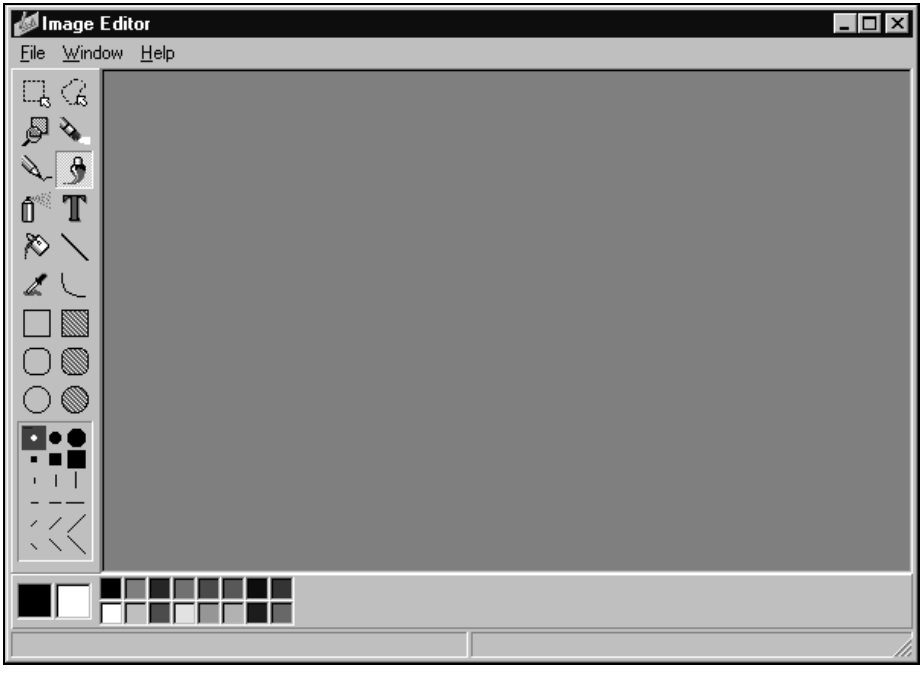


Рис. 17.3. Окно графического редактора

Выполните в графическом редакторе команды **File | New | Icon File** и в открывшемся окне **Icon Properties** (рис. 17.4) выберите **Size** (Размер) 32×32 и **Color** (цвет) 16 color. Эти установки соответствуют стандартным иконкам для программ.

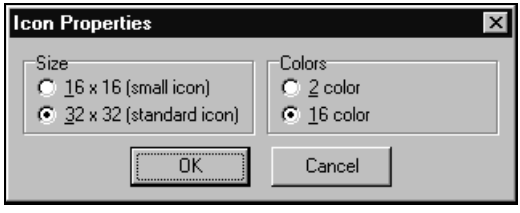


Рис. 17.4. Окно **Icon Properties**

После чего щелкните кнопку **OK**. Перед вами откроется новое окно (рис. 17.5), в котором слева приведено пустое поле новой иконки в увеличенном масштабе для рисования, а справа — в масштабе 1:1 для просмотра.

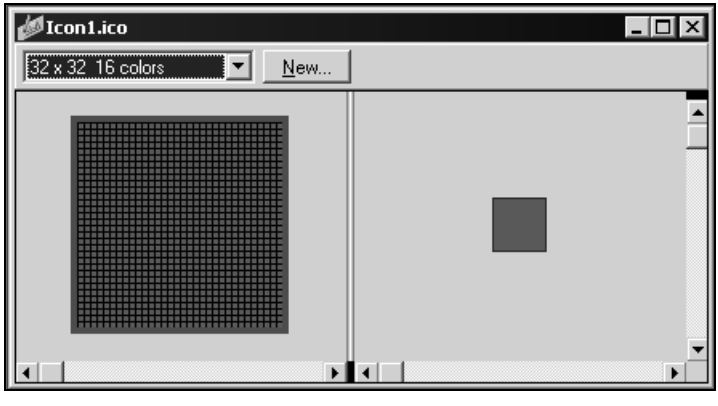


Рис. 17.5. Окно новой иконки

С помощью инструментов, расположенных слева на поле инструментов графического редактора, можно нарисовать на пустом поле иконки произвольный рисунок и раскрасить его, а также сделать любые символьные надписи. Редактор позволяет создавать иконки, курсоры, битовые рисунки и файлы ресурсов. В дальнейшем мы еще не раз будем прибегать к его помощи. Встроенная в редактор справка позволяет довольно быстро научиться пользоваться этим инструментом. Фон иконки бирюзового цвета является прозрачным по умолчанию. Поэтому области иконки с таким цветом будут просвечиваться на любом фоне. Нарисуйте на пустом поле любой рисунок, например табло с цифрами, и сохраните иконку с помощью команды **File | Save as | Icon1 ico** в каталоге с программой цифровых часов — календаря. Закройте графический редактор.

Теперь у нас есть файл иконки для программы и остается лишь внедрить его в программу. Для этого необходимо открыть свойства проекта с помощью команды **Project | Options** в среде разработки Borland C++ Builder 6 или нажать комбинацию клавиш <Ctrl>+<Shift>+<F11>. В открывшемся окне свойств проекта (рис. 17.6) необходимо открыть вкладку **Application**, и с помощью кнопки **Load Icon** выбрать созданный нами файл иконки Icon1.ico.

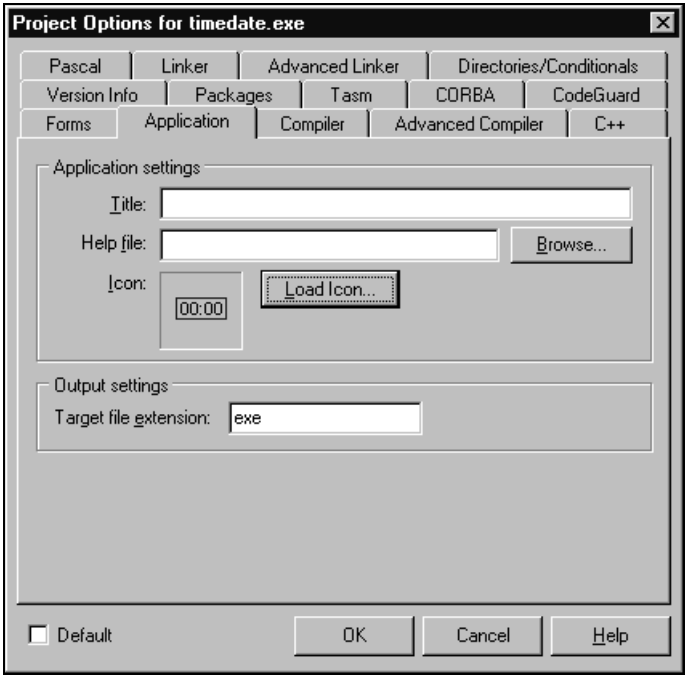


Рис. 17.6. Окно свойств проекта

Закройте окно свойств проекта, щелкнув по кнопке **OK**, и запустите проект с помощью клавиши <F9>. После запуска программы вы увидите, что теперь программа имеет созданную нами иконку, которую можно поместить на рабочий стол в качестве пиктограммы программы.

Обратите внимание на то, что программы, полученные в среде разработки с настройками "по умолчанию", могут не работать на тех компьютерах, где не установлен пакет Borland C++ Builder 6. Это связано с тем, что программы, созданные с помощью Borland C++ Builder 6, требуют для своей работы некоторые библиотечные файлы с расширением bpl, dll, lib и т. п., которых может не оказаться на другом компьютере. Для решения этой проблемы нужно изменить настройки среды разработки Borland C++ Builder 6 так, чтобы необходимые библиотечные файлы упаковывались в исполняемый файл программы. От этого размер файла исполняемой программы увеличится, но зато созданная программа будет работать на любом другом компьютере.

Настройку среды разработки Borland C++ Builder 6 нужно изменить следующим образом. Выполните команду **Project | Options**. В появившемся окне свойств проекта откройте вкладку **Compiler** и нажмите программную кнопку **Release** (рис. 17.7).

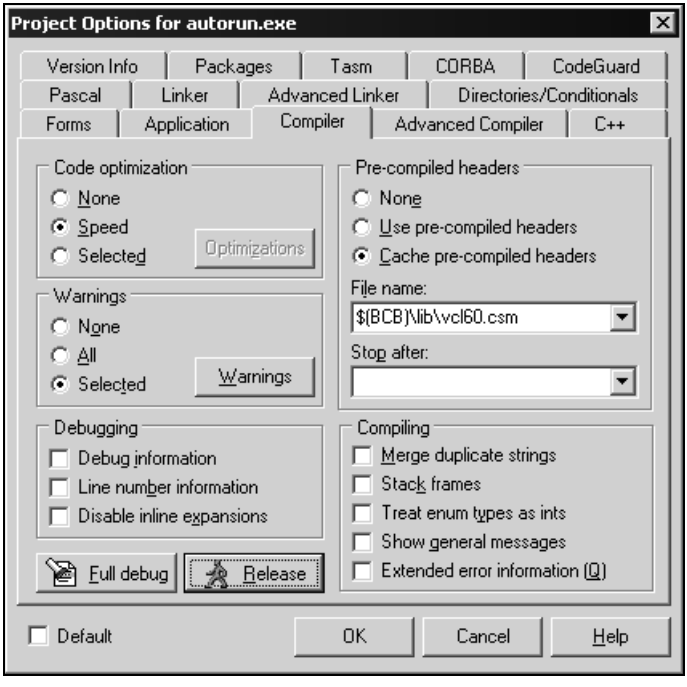


Рис. 17.7. Окно вкладки **Compiler** свойств проекта

Затем откройте вкладку **Packages** и снимите флажок перед строкой **Build with runtime packages** в группе **Runtime packages** (рис. 17.8).

Наконец откройте вкладку **Linker** и снимите флажок перед строкой **Use dynamic RTL** (рис. 17.9).

Теперь сохраните и перекомпилируйте весь проект. Полученный в результате файл программы будет работать на всех компьютерах.

Программа готова! Полный проект данной программы вместе с исполняемым файлом timedate.exe и файлом иконки Icon1.ico находятся в каталоге \PROJECT\TIMEDATE на диске, прилагаемом к книге. Вы можете использовать их для сравнения с созданным вами проектом.

Итак, мы на практике применили в нашей программе несколько визуальных компонентов и выполнили изменение некоторых их свойств. В результате чего получили работоспособное и полезное приложение.

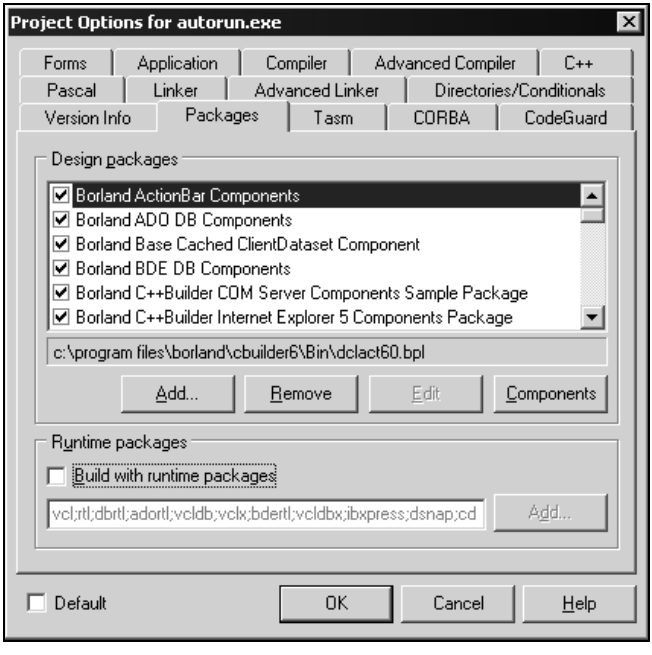


Рис. 17.8. Окно вкладки **Packages** свойств проекта

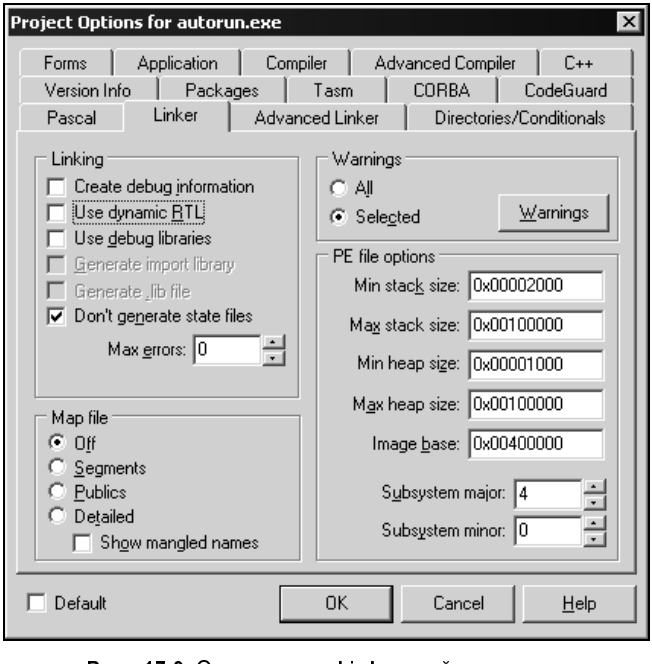
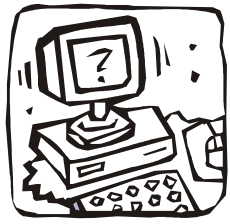


Рис. 17.9. Окно вкладки **Linker** свойств проекта



Глава 18

Визуальные компоненты

В этой главе рассказывается о назначении основных визуальных компонентов, которые используются при создании программы, являясь основными элементами оформления ее внешнего вида.

Теперь давайте изучим более подробно основные компоненты среды разработки Borland C++ Builder 6, которые применяются при разработке программ.

Как было сказано, все визуальные компоненты, из которых строится любое приложение, размещаются на палитре компонентов, состоящей из вкладок. Вкладки объединяют компоненты по группам. Перечень вкладок с описанием состава компонентов, которые они включают в себя, приведен в табл. 18.1.

Таблица 18.1. *Перечень вкладок визуальных компонентов и описание их состава*

Вкладка	Состав
Standard	Стандартная. Содержит наиболее часто используемые компоненты
Additional	Дополнительная. Содержит дополнения компонентов к стандартной вкладке
Win32	Содержит 32-битные компоненты в стиле Windows
System	Системная. Содержит такие компоненты, как таймер, окно для рисования, проигрыватель и др.
Data Access	Доступ к данным. Содержит компоненты для создания программ, работающих с базами данных и обеспечивающих доступ к этим базам
Data Controls	Управление данными. Компоненты отображения и редактирования данных
dbExpress	Связь с данными с использованием языка баз данных SQL

Таблица 18.1 *(окончание)*

Вкладка	Состав
DataSnap	Компоненты для связи с сервером приложений при построении многопоточных приложений, работающих с данными
BDE	Доступ к данным через Borland Database Engine (BDE)
ADO	Связь с базами данных через Active Data Objects (ADO) — множество компонентов ActiveX, использующих для доступа к информации баз данных Microsoft OLE DB
InterBase	Прямая связь с Interbase, минуя Borland Database Engine и Active Data Objects
WebServices	Компоненты клиентских интернет-приложений, использующие доступ к службам Web с помощью SOAP
InternetExpress	Построение приложений Internet Express — одновременно приложений Web-сервера и клиента баз данных с параллельными потоками
Internet	Компоненты для создания Web-серверов
WebSnap	Компоненты для создания Web-серверов, содержащих сложные страницы, управляемые данными
FastNet	Различные протоколы доступа к Интернету
Decision Cube	Компоненты для многомерного анализа данных
QReport	Компоненты для подготовки отчетов (диаграмм, графиков и т. п.)
Dialogs	Диалоги. Содержит системные диалоги типа "Открыть файл", "Закрыть файл" и т. п.
Win 3.1	Windows 3.1. Содержит компоненты в стиле Windows 3.1
Samples	Образцы. Содержит компоненты построения графика, календаря, выбора цвета и др.
ActiveX	Содержит некоторые компоненты ActiveX
COM+	Содержит компонент, дающий доступ к каталогу COM+, содержащему данные по конфигурации COM+
Servers	Содержит компоненты связи с серверами COM
IndyClients	Содержит компоненты клиентских приложений Internet Direct (Indy), дающих доступ к различным протоколам Интернета из приложений Delphi, C++ Builder и Kylix
IndyServers	Содержит компоненты серверных приложений Internet Direct
IndyMisk	Различные вспомогательные компоненты приложений Internet Direct
Office97	Оболочки VCL для распространенных серверов COM офисных приложений Microsoft

Пусть вас не пугают неизвестные аббревиатуры типа BDE, ADO, COM и др., встречающиеся в таблице. С этими понятиями вы разберетесь позже, когда начнете их использовать в собственных программах. Здесь они приведены лишь для общего обзора.

Теперь рассмотрим назначение наиболее часто используемых компонентов. С некоторыми из них мы уже познакомились. Начнем знакомство с вкладки **Standard**.

Компоненты *Standard*

На этой вкладке располагаются компоненты (рис. 18.1), с помощью которых происходит подключение к программе стандартных интерфейсных элементов, имеющихся во всех версиях операционных систем Windows. Рассмотрим эти компоненты по порядку слева направо.

- **Frame** (Кадр) — предназначен для создания контейнера (окна) для размещения других компонентов. Данный компонент очень похож на форму **Form**. Для размещения этого компонента на форме необходимо первый раз создать его с помощью команд **File | New | Frame**. Именно такое сообщение появляется при попытке размещения этого компонента стандартным образом (рис. 18.2).

Рис. 18.1. Палитра компонентов **Standard**

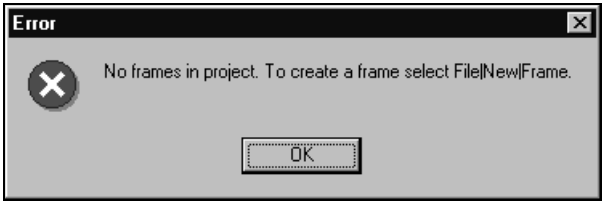


Рис. 18.2. Сообщение об ошибке размещения компонента **Frame**

Дело в том, что компонент **Frame** первый раз должен быть сконструирован через главное меню, поскольку является сложным компонентом. После его создания, или создания нескольких таких компонентов, можно пользоваться стандартным способом размещения их на форме путем выбора имеющихся фреймов в раскрывающемся списке. Для практики создайте компонент **Frame2** с помощью команды **File | New | Frame**, а затем перейдите на форму и щелкните дважды по компоненту **Frames** на вкладке

Standard панели визуальных компонентов. Перед вами должно открыться окно (рис. 18.3), из которого можно выбрать компонент **Frame2**.

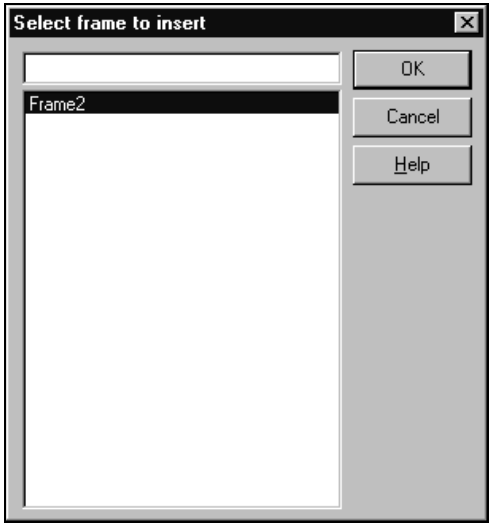


Рис. 18.3. Окно выбора компонента **Frame**

- **MainMenu** (Главное меню) — предназначен для создания главного меню программы. С этим и другими компонентами мы познакомимся поближе в процессе создания новых программ.
- **PopupMenu** (Всплывающее меню) — предназначен для создания всплывающего меню некоторых компонентов. Обычно с помощью этого компонента создается контекстное меню.
- **Label** (Этикетка) — создает на форме текстовую метку или надпись.
- **Edit** (Редактирование) — создает на форме поле для редактирования текстовой строки.
- **Memo** (Поле) — отображает на форме поле для редактирования текстовых строк. Обычно служит для создания редакторов и полей для вывода блоков данных.
- **Button** (Кнопка) — является самым распространенным компонентом. Служит для создания в приложении различных прямоугольных кнопок с текстовой надписью.
- **CheckBox** (Ячейка состояния) — позволяет создавать на форме приложения ячейку с двумя состояниями (без галочки и с галочкой) и строкой названия. Щелчок левой кнопкой мыши по этому компоненту во время работы программы вызывает изменение состояния компонента на проти-

воположное. В программе всегда можно узнать состояние этого компонента и тем самым выполнять то или иное действие.

- **RadioButton** (Радиокнопка) — создает круглое поле с двумя состояниями (с точкой и без точки) и текстовой строкой, поясняющей ее назначение в программе. Обычно несколько таких компонентов, расположенных на форме, позволяют переключить только один элемент из группы. Для наглядности сказанного и закрепления материала на практике создайте новое приложение и расположите на форме несколько компонентов **RadioButton**. После чего запустите приложение на выполнение и пощелкайте левой кнопкой мыши поочередно по каждому из них. Вы увидите, что можно изменить состояние только для одного из этих компонентов, так как остальные компоненты переключают при этом свое состояние автоматически.
- **ListBox** (Окно списка) — создает прямоугольное поле для отображения текстовых строк с возможностью их выбора, добавления или удаления при работе программы.
- **ComboBox** (Комбинированный список) — позволяет создавать на форме элемент, являющийся комбинацией строки ввода и выпадающего списка для выбора. Фактически объединяет в себе компоненты **ListBox** и **Edit**.
- **ScrollBar** (Линейка прокрутки) — создает элемент, похожий на линейку с бегунком и кнопками для прокрутки окна, к которому относится этот элемент. Кроме того, с его помощью можно изменять в пределах некоторого заданного интервала значение величины какого-либо параметра.
- **GroupBox** (Окно группы) — служит для создания области, визуально объединяющей на форме несколько интерфейсных элементов.
- **RadioGroup** (Группа радиокнопок) — позволяет создавать на форме контейнер в виде прямоугольной рамки для объединения группы взаимоисключающих радиокнопок.
- **Panel** (Панель) — создает пустую область, на которой можно разместить другие компоненты. Как правило, используется для создания панели инструментов в программе.
- **Action List** (Список действий) — осуществляет управление взаимодействием между интерфейсными элементами и логикой программы.

На рис. 18.4 приведено окно формы, на которой расположены все перечисленные компоненты в порядке их описания слева направо и сверху вниз, начиная с **MainMenu**.

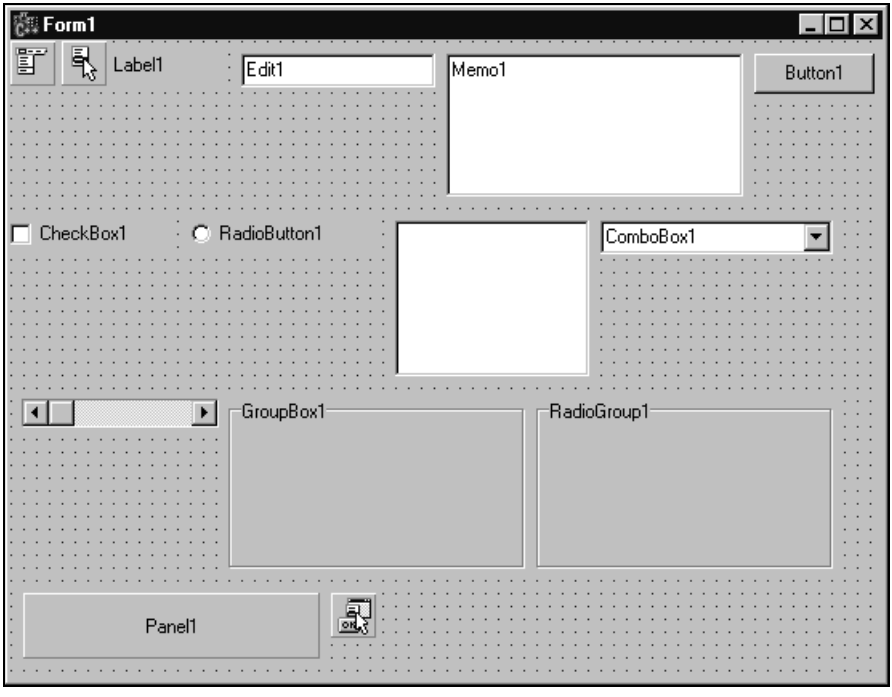


Рис. 18.4. Окно формы с компонентами

Компоненты *Addition*

Компоненты, расположенные на вкладке **Addition** (рис. 18.5), предназначены для включения в программу дополнительных интерфейсных элементов, с помощью которых можно создать более привлекательный и дружественный интерфейс программы. Рассмотрим компоненты, расположенные на этой вкладке.

- **BitBtn** (Графическая кнопка) — служит для создания на форме приложения кнопки с изображением и надписью.
- **SpeedButton** (Быстрая кнопка) — позволяет создать на форме кнопку с изображением без надписи. Знакома нам по панели быстрых кнопок.
- **MaskEdit** (Форматированный ввод) — предназначен для создания прямоугольного поля ввода данных в специально заданном формате. Позволяет проверить корректность вводимых данных с помощью маски.

Рис. 18.5. Компоненты вкладки **Addition**

- **StringGrid** (Строковая таблица) — служит для создания таблицы (сетки), состоящей из текстовых строк.
- **DrawGrid** (Графическая таблица) — создает на форме двумерную таблицу для отображения графических данных.
- **Image** (Образ) — предназначен для создания на форме невидимого контейнера, в который можно поместить один графический файл с битовым образом, пиктограммой или метафайл.
- **Shape** (Фигура) — позволяет рисовать на форме простые геометрические фигуры, такие как окружность, квадрат, эллипс или прямоугольник при изменении свойства **Shape**. Допускает изменение цвета фигур и их штриховки при изменении свойств **Color** (Цвет) и **Style** (Стиль) группы **Brush** (Кисть).
- **Bevel** (Скос) — создает объемные рамки для различных групп объектов. Служит только для обрамления и не позволяет выполнять программную функцию.
- **ScrollBox** (Контейнер прокрутки) — позволяет создавать контейнер для объектов. Автоматически снабжается линейками прокрутки, если помещенный в него объект превышает размеры самого контейнера на экране.
- **CheckListBox** (Список с ячейками) — отображает список элементов, у каждого из которых есть собственная ячейка состояния.
- **Splitter** (Разделитель) — разделяет рабочую область программы на две части и позволяет менять их размеры во время работы программы.
- **StaticText** (Статический текст) — создает на форме текстовую строку, с некоторыми возможностями ее оформления.
- **Chart** (Диаграмма) — предназначен для создания и вывода на печать многоцветных графиков и диаграмм.

Компоненты *Win32*

Компоненты, расположенные на вкладке **Win32** (рис. 18.6), обеспечивают подключение к программе интерфейсных элементов, используемых в 32-разрядных версиях операционной системы Windows. Использование данных компонентов позволяет придать программе современный и профессиональный вид. Рассмотрим основные компоненты данной вкладки.

- **TabControl** (Управление табуляцией) — служит для создания перекрывающих друг друга вкладок и для создания интерфейсов в стиле палитры компонентов.
- **PageControl** (Управление страницами) — создает на форме контейнер для размещения дочерних страниц программы. Позволяет во время работы

программы изменять ее интерфейс, перелистывая дочерние страницы на главной форме.

Рис. 18.6. Компоненты вкладки **Win32**

- **Image List** (Список образов) — создает на форме невидимый контейнер для набора графических изображений одинакового размера. Позволяет выбирать изображение из списка.
- **RichEdit** (Обогащенный редактор) — позволяет создавать редактор с готовым набором функций, свойственных большинству редакторов.
- **TrackBar** (Дорожка с полосками) — создает на форме шкалу с метками и регулятором текущего положения. Применяется в программе как регулятор громкости звука, регулятор размера изображения и т. п.
- **Progress Bar** (Прогресс-индикатор) — создает на форме прямоугольный индикатор для отображения процесса выполнения длинных процедур в программе (копирования, поиска и т. п.).
- **UpDown** (Вверх-вниз) — служит для создания интерфейсных элементов с возможностью увеличения или уменьшения какого-либо значения с помощью кнопок **Вверх** и **Вниз** данного компонента.
- **HotKey** (Горячая клавиша) — обеспечивает возможность создания пользователем горячих клавиш, определяющих быстрый доступ к разделам меню.
- **Animate** (Аниматор) — создает на форме невидимый контейнер для воспроизведения видеозаписей в формате AVI.
- **DataTimePicker** (Сборщик даты и времени) — создает в программе интерфейс для ввода даты и времени.
- **MonthCalendar** (Месячный календарь) — размещает на форме календарь с отображением всех дней месяца и возможностью перелистывания месяцев и корректировки даты текущего дня.
- **TreeView** (Вид дерева) — позволяет создавать в программе иерархическое древовидное отображение данных.
- **ListView** (Вид списка) — создает список элементов с отображением в различных стилях (крупные значки, мелкие значки, таблица и пр.).
- **HeaderControl** (Управление заголовком) — служит для управления панелями, расположенными под данным компонентом.
- **StatusBar** (Панель состояния) — создает контейнер в нижней части формы для отображения статусной информации. Например, состояние кнопок **Caps Lock**, **Num Lock** и **Scroll Lock**.

- **ToolBar** (Панель инструментов) — позволяет создать на форме контейнер для размещения быстрых кнопок.
- **CoolBar** (Холодная панель) — позволяет делать перестраиваемые панели, состоящие из полос.
- **PageScroller** (Страница прокрутки) — создает на форме контейнер для прокрутки элементов, не вмещающихся на экран целиком.
- **ComboBoxEx** (Расширенный выпадающий список) — позволяет создать на форме выпадающий список элементов.

Компоненты *System*

Компоненты вкладки **System** (рис. 18.7) обеспечивают подключение к программе специальных элементов управления и элементов межпрограммного интерфейса. Рассмотрим расположенные на данной вкладке компоненты.

- **Timer** (Таймер) — предназначен для создания невидимого на форме элемента, с помощью которого можно отслеживать интервалы времени в программе. Данным компонентом мы уже пользовались в программе **timedate** *главы 17*.

Рис. 18.7. Компоненты вкладки **System**

- **PaintBox** (Рамка рисования) — создает на форме элемент, на котором можно рисовать.
- **MediaPlayer** (Многофункциональный проигрыватель) — создает на форме панель для проигрывания файлов с различными форматами и управления устройствами мультимедиа. С его помощью можно управлять проигрывателем компакт-дисков, звуковым адаптером и т. д.
- **OleContainer** (Контейнер OLE) — реализует в программе механизм внедрения и связывания объектов OLE (Object Linking and Embeddin, связывание и встраивание объектов), с помощью которого можно передавать данные между различными программами в среде Windows.
- **DdeClientConv** (Клиент DDE) — реализует в программе механизм динамического обмена данными DDE (Dynamical Data Exchange), являющийся расширением операционной системы Windows. Данный механизм предназначен для передачи данных от одной программы — сервера — к другой — приемнику (клиенту).
- **DdeClientItem** (Элемент клиента данных DDE) — определяет элемент динамического обмена и реализует в программе механизм динамического обмена данными между клиентами.

- **DdeServerConv** (Сервер DDE) — устанавливает режим динамического обмена между программами сервера DDE.
- **DdeServerItem** (Элемент данных сервера DDE) — определяет режим динамического обмена данными сервера DDE.

Компоненты *Dialogs*

Компоненты вкладки **Dialogs** (рис. 18.8), позволяют создавать диалоги с пользователем программы. Диалоги придают программе гибкость и позволяют пользователю манипулировать объектами программы во время ее работы. Вкладка содержит десять компонентов стандартных диалогов.

- **OpenDialog** (Диалог открытия) — позволяет создать в программе диалог открытия файлов. Имеет возможность установки фильтров на имя и расширение открываемых файлов.

Рис. 18.8. Компоненты вкладки **Dialogs**

- **SaveDialog** (Диалог сохранения) — создает в программе диалог с возможностью выбора имени файла для сохранения. Обычно используется в связке с компонентом **OpenDialog**.
- **OpenPictureDialog** (Диалог открытия рисунка) — предназначен для создания диалогов открытия файлов с графическим изображением. Отличается от компонента **OpenDialog** возможностью предварительного просмотра файла изображения непосредственно в окне диалога.
- **SavePictureDialog** (Диалог сохранения рисунка) — создает диалог сохранения файла с изображением. Так же, как и **OpenPictureDialog**, имеет возможность предварительного просмотра файла изображения непосредственно в окне диалога.
- **FontDialog** (Диалог выбора шрифта) — предоставляет возможность создания диалога для выбора шрифта текста в программе.
- **ColorDialog** (Диалог выбора цвета) — предназначен для создания диалога для выбора цвета из палитры цветов Windows. Полученный числовой параметр цвета после этого можно предать любому компоненту, имеющему свойство **Color**.
- **PrintDialog** (Диалог выбора принтера) — позволяет организовать в программе диалог выбора печатающего устройства из числа установленных на компьютере. Кроме того, данный диалог позволит задать число копий, выбрать или задать диапазон печатаемых страниц и т. п.

- **PrinterSetupDialog** (Диалог установки принтера) — создает в программе диалог настройки параметров печатающего устройства, к числу которых относятся выбор типа бумаги, ее ориентация, параметры отображения графики и т. д.
- **FindDialog** (Диалог поиска) — открывает в программе диалог поиска текста с возможностью настройки параметров поиска.
- **Replace Dialog** (Диалог замены) — открывает в программе диалог поиска текста с возможностью замены найденного фрагмента на заданный пользователем. Обладает всеми возможностями компонента **FindDialog**.

Кроме рассмотренных вкладок с компонентами на палитре присутствует множество других вкладок (**Samples**, **Office2k**, **Internet Express**, **BDE**, **ADO** и т. п.) для более сложных компонентов, используемых программистами-профессионалами. Описание всех компонентов выходит за рамки данной книги. Поэтому далее приведено краткое описание содержимого лишь некоторых из этих вкладок.

Компоненты *Samples*

Компоненты вкладки **Samples** (рис. 18.9) представляют собой примеры компонентов для управления программой. Пользователь имеет возможность создавать собственные компоненты, которые будут помещаться на этой вкладке. Поскольку это всего лишь примеры, они снабжены в Borland C++ Builder 6 сокращенной документацией. Рассмотрим некоторые компоненты данной вкладки.

- **Tray Icon** (Икона поддона) — позволяет сворачивать программу в правое подвальное окно, имеющее системное название **tray** (Поддон).

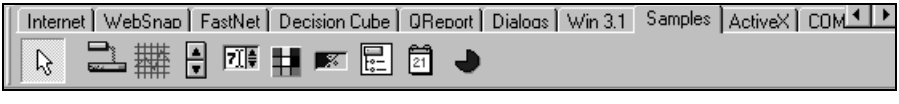


Рис. 18.9. Компоненты вкладки **Samples**

- **Performance Graph** (График исполнения) — служит для наглядного отображения используемых ресурсов компьютера.
- **CSpinButton** (Кнопка вращения) — используется для создания кнопок-счетчиков.
- **CSpinEdit** (Редактор вращения) — применяется для создания окна редактирования в комбинации с кнопкой-счетчиком.
- **CColorGrid** (Цветная сетка) служит для создания таблицы цветов, в которой пользователь может выбирать требуемый цвет.

- **CGauge** (Измеритель) — предназначен для отображения изменения какой-либо переменной в программе.
- **CDirectory Outline** (Схема каталога) — используется для отображения структуры каталогов выбранного диска.
- **CCalendar** (Календарь) — применяется для отображения календаря на указанный месяц в стандартном формате.
- **Pie** (Сектор) — используется для создания индикатора хода процесса в виде секторной диаграммы. Имеет встроенный редактор **Edit Angles** для изменения углов сектора. Редактор вызывается через контекстное меню компонента.

Компоненты *Office97*

Компоненты вкладки **Office97** (рис. 18.10) позволяют интегрировать приложения с программами из пакета Microsoft Office. В зависимости от того, какой именно пакет Microsoft Office был установлен на компьютере, компоненты описываемой вкладки могут иметь имя Office97, Office2K и т.п. Рассмотрим назначение компонентов данной вкладки.

Рис. 18.10. Компоненты вкладки **Office97**

Первые пять компонентов позволяют работать с документами, созданными в формате Microsoft Word. Они предоставляют возможность управлять шрифтами и форматированием документа, а также определяют режим динамического обмена данными (DDE) сервера.

Следующие три компонента позволяют внедрить в приложение возможность создания презентаций Microsoft PowerPoint.

Вслед за ними следует группа компонентов, позволяющая внедрять в приложение сервисные функции почтовых программ.

Затем следует группа компонентов, предназначенная для работы с электронными таблицами Microsoft Excel. Кроме того, компонент **ExcelChart** данной группы позволяет создавать в программе специальные диаграммы.

Далее на вкладке следуют компоненты, с помощью которых производится работа с базами данных Microsoft Access. Компоненты данной группы разделены по назначению. Часть из них отвечает непосредственно за работу с данными, а другая часть — за создание инструментов оперирования данными.

Компоненты *Data Access*

Компоненты вкладки **Data Access** (рис. 18.11) служат для создания в приложении элементов управления доступом к базам данных.

Рис. 18.11. Компоненты вкладки **Data Access**

Эти компоненты позволяют включать в создаваемую программу возможность работы пользователя с информацией базы данных.

Компоненты *Data Controls*

Компоненты вкладки **Data Controls** (рис. 18.12) создают специальные элементы управления базами данных, доступные пользователю из приложения.

Рис. 18.12. Компоненты вкладки **Data Controls**

Компоненты данной вкладки позволяют создавать в приложении интерфейсы баз данных. Это могут быть поля для редактирования текста, списков, рисунков и т. п.

Компоненты *Internet Express*

Компоненты вкладки **Internet Express** (рис. 18.13) позволяют осуществить построение приложений с архитектурой "клиент — сервер", которые действуют по протоколам TCP-IP и HTTP. Кроме того, они позволяют создать распределенное сетевое приложение, Web-сервер которого будет выступать в роли клиентского приложения MIDAS. MIDAS (Multi-tier Distributed Application Services, многоуровневый сервис распределенных приложений) — это технология, разработанная компанией Inprise/Borland для создания и эксплуатации надежных, высокопроизводительных, распределенных систем на основе многозвенной архитектуры.

- **XML Broker** (XML-брокер) — представляет пакеты данных от поставщика сервисного приложения с кодировкой HML.
- **InetXPage Producer** (Производитель интернет-страниц) — позволяет сгенерировать страницу HTML с информацией, полученной от сервисного приложения MIDAS.

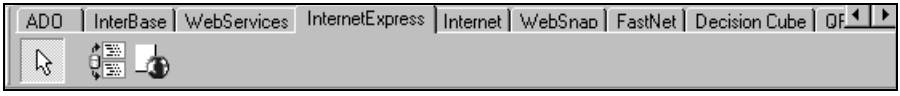


Рис. 18.13. Компоненты вкладки **Internet Express**

Компоненты *Internet*

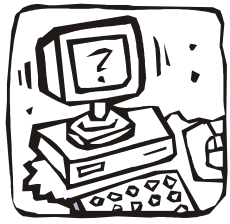
Компоненты вкладки **Internet** (рис. 18.14) обеспечивают подключение к программе специальных элементов для создания клиентских и серверных приложений с протоколами TCP/IP и HTTP.

- **ClientSockets** (Клиент подключений) — позволяет создавать приложения для клиентов сетевого подключения по протоколу TCP/IP.
- **ServerSockets** (Сервер подключений) — превращает приложение в сервер TCP/IP. Данный сервер позволяет контролировать определенный порт на предмет распознавания запросов и установления соединений.

Рис. 18.14. Компоненты вкладки **Internet**

Итог

Как видно из описания компонентов, среда разработки Borland C++ Builder 6 позволяет создавать программы самого разнообразного назначения во многих сферах компьютерных технологий. К тому же перечень компонентов постоянно пополняется с помощью многочисленной армии программистов. В настоящее время в сети Интернет доступно множество новых дополнительных компонентов, и их число постоянно продолжает расти. Применение компонентов позволяет создавать даже начинающим программистам очень мощные и надежные программы в предельно короткие сроки.



Глава 19

Программа АВС

В данной главе рассказывается о создании в среде разработки Borland C++ Builder 6 учебной программы с применением некоторых компонентов и мультимедийных функций.

Создадим еще одну программу, предназначенную для обучения детей азбуке. Интерфейс, придуманный мною для этой программы, довольно прост. Программа будет иметь только одно окно, на котором разместятся все буквы русского алфавита и один рисунок. На рис. 19.1 показано окно этой программы.

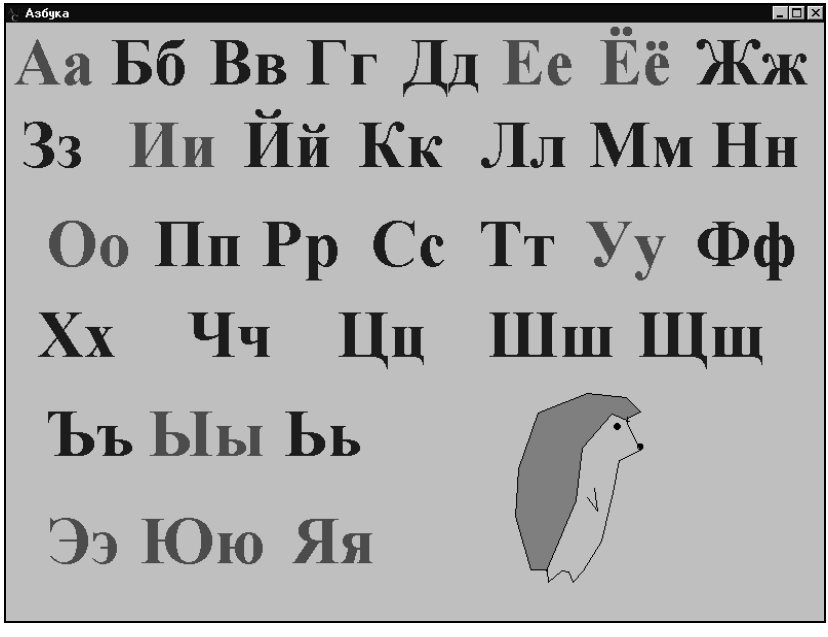


Рис. 19.1. Окно программы

При работе с данной программой после щелчка левой кнопки мыши по любой из букв на экране появляется рисунок предмета (зверька, игрушки), название которого начинается с этой буквы или содержит ее. Одновременно через акустику компьютера воспроизводится название буквы.

Приступим к созданию такой программы. Запустите среду разработки Borland C++ Builder 6 и создайте новый проект с помощью команд **File | New | Application**. Сразу же сохраните проект с помощью команд **File | Save Project As** на любом диске, создав для этого отдельную папку (каталог) с произвольным названием, например ABC. Программный модуль сохраните с именем Uabc.cpp, а сам проект — с именем abc.bpr.

Поместите на форму **Form1** проекта компонент с названием **Image1** из вкладки **Additional**. Данный компонент необходим в программе для отображения на экране рисунков из загружаемых файлов. Замените свойство **AutoSize** компонента **Image1** с **false** на **true**, для того чтобы его размер автоматически изменялся под размер отображаемого рисунка. Затем поместите на форму компонент **Label1** из вкладки **Standard** палитры компонентов. Измените свойство **Caption** данного компонента на **Аа** (большая и маленькая русская буква а). Откройте свойства **Font** компонента **Label1** и приведите их в соответствие со свойствами, изображенными на рис. 19.2.

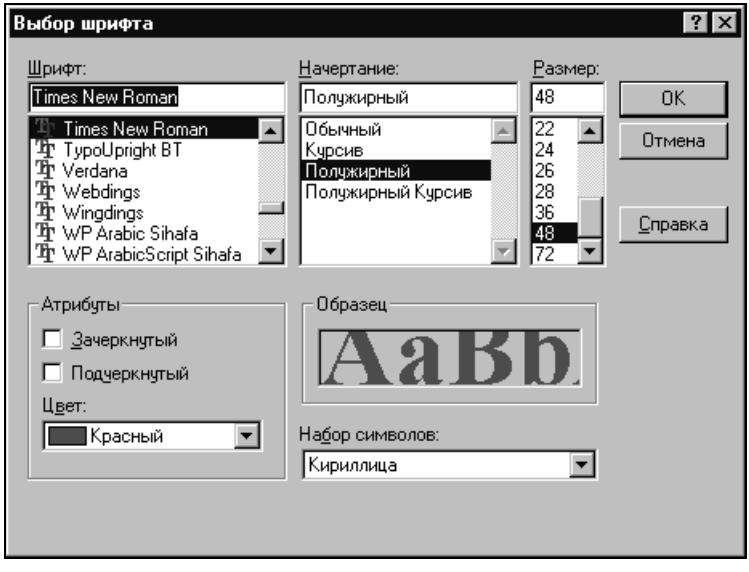


Рис. 19.2. Свойства **Font** компонента **Label1**

Скопируйте компонент **Label1** на форме 32 раза с помощью команд **Edit Copy** и **Edit Paste** контекстного меню. При выделении и копировании можно пользоваться операциями для группы элементов, что значительно ускоряет процесс копирования. В результате на форме должно быть создано 33 компо-

нента с названием от **Label** до **Labe32**. Измените свойство **Caption** (Заголовок) скопированных элементов в соответствии с буквами русского алфавита от Бб до Яя. Для облегчения данной операции воспользуйтесь табл. 19.1, в которой приведено соответствие букв их порядковому номеру в русском алфавите.

Таблица 19.1. *Соответствие букв их порядковому номеру в русском алфавите*

Номер	Буква	Номер	Буква	Номер	Буква
1	А	12	К	23	Х
2	Б	13	Л	24	Ч
3	В	14	М	25	Ц
4	Г	15	Н	26	Ш
5	Д	16	О	27	Щ
6	Е	17	П	28	Ъ
7	Ё	18	Р	29	Ы
8	Ж	19	С	30	Ь
9	З	20	Т	31	Э
10	И	21	У	32	Ю
11	Й	22	Ф	33	Я

Измените красный цвет свойства **Font** для согласных букв на синий цвет. Как известно, в русском алфавите 10 гласных и 23 согласных буквы.

Теперь щелкните по компоненту **Label1** левой кнопкой мыши и в открывшемся окне инспектора кода введите текст программы, изображенной на рис. 19.3.

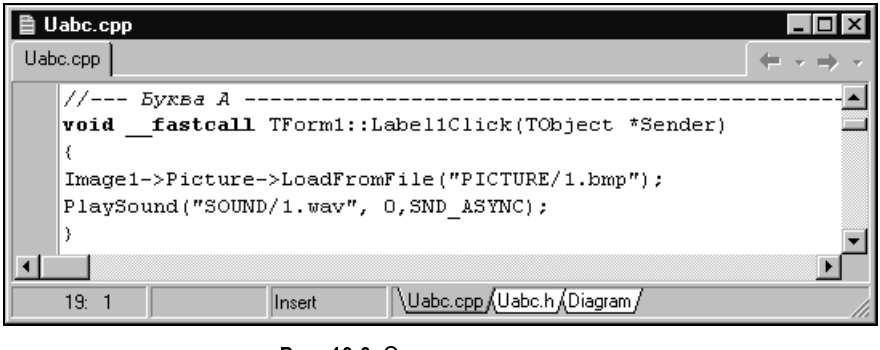


Рис. 19.3. Окно инспектора кода

Причем заготовка функции, создаваемая автоматически самой средой разработки и состоящая из строк, приведенных в листинге 19.1, уже будет присутствовать в вашей программе.

Листинг 19.1. Заготовка функции

```
void __fastcall TForm1::Label1Click(TObject *Sender)
{
}
```

Вам останется лишь добавить строку комментариев

```
//--- Буква А -------------------------------------------------------
```

перед функцией и строки команд

```
Image1->Picture->LoadFromFile("PICTURE/1.bmp");
PlaySound("SOUND/1.wav", 0,SND_ASYNC);
```

между фигурными скобками функции.

Первая строка команд будет изменять свойство **Picture** компонента **Image1** при выполнении программы путем загрузки файла 1.bmp из каталога PICTURE после щелчка левой кнопкой мыши по данной букве.

Вторая строка состоит из одной мультимедийной команды, с помощью которой будет воспроизводиться звуковой файл 1.wav, расположенный в каталоге SOUND. Естественно, что эти файлы с рисунком и звуком необходимо будет создать самостоятельно и поместить в соответствующие подкаталоги проекта с названиями PICTURE и SOUND, которые необходимо создать в основном каталоге проекта. Borland C++ Builder 6 позволяет загружать звуковые файлы в формате WAV и файлы рисунков — в форматах BMP, WMF или JPG. Поэтому при создании данных файлов их необходимо сохранять только в таких форматах. Всего нужно будет создать 33 файла с рисунками и 33 звуковых файла: по количеству букв русского алфавита. Имена этих файлов должны носить названия от 1 до 33. Давать этим файлам имена по названию русских букв нежелательно, поскольку среда разработки является англоязычной, и при использовании таких названий в дальнейшем могут возникнуть непредсказуемые проблемы.

При создании звуковых файлов я рекомендую записать звуки букв, произносимые самим ребенком, который будет затем обучаться азбуке по создаваемой нами программе. В этом случае ему будет интересно слышать собственный голос, и процесс изучения букв станет интересен. Для детей такой процесс обучения превратится в своеобразную игру.

Неплохо было бы, чтобы ребенок сам нарисовал рисунки к данной программе. Тогда при изучении азбуки он будет узнавать не только свой голос, но и свои рисунки.

Звуковые файлы легко создаются с помощью программы звукозаписи sndrec32.exe, входящей в состав стандартных программ операционной системы Windows. Данная программа запускается с помощью команд **Пуск | Программы | Стандартные | Развлечения | Звукозапись**. Рисунки можно создать с помощью любого графического редактора. Для этих целей подойдет простейший графический редактор **mspaint.exe**, также входящий в состав стандартных программ операционной системы Windows и вызываемый командами **Пуск | Программы | Стандартные | Paint**. Фон рисунков лучше всего сделать прозрачным или серым, для того чтобы они сливались при отображении с фоном формы программы.

Вставьте в начало программы строку подключения библиотеки мультимедийных функций в виде `#include <mmsystem.hpp>`. Эти функции необходимы нашей программе для выполнения команд проигрывания звуковых файлов. Окончательный вид заголовка программы будет таким, как в листинге 19.2.

Листинг 19.2. Окончательный вид заголовка программы

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include<mmsystem.hpp>
#include "Uabc.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
```

Теперь откройте форму **Form1** программы и замените ее свойство **Caption** на заголовок **Азбука**. Разверните форму на весь экран и разместите все компоненты на форме в соответствии с рис. 19.1.

Создайте функции обработки для каждой из оставшихся 32-х букв, вызывая для этого инспектор кода щелчком левой кнопки мыши по букве, и скопируйте строки программы, приведенные для буквы А (компонент **Label1**), в функцию каждой буквы. Замените в скопированных строчках комментарии и имена файлов в соответствии с названием буквы. В результате у вас должна получиться программа, состоящая из строк, похожих на строки, приведенные в листинге 19.3.

Листинг 19.3. Строки программы

```
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
}
```

```
//--- Буква А ------------------------------------------------------------
void __fastcall TForm1::Label1Click(TObject *Sender)
{
Image1->Picture->LoadFromFile("PICTURE/1.bmp");
PlaySound("SOUND/1.wav", 0,SND_ASYNC);
}
//--- Буква Б ------------------------------------------------------------
void __fastcall TForm1::Label2Click(TObject *Sender)
{
Image1->Picture->LoadFromFile("PICTURE/2.bmp");
PlaySound("SOUND/2.wav", 0,SND_ASYNC);
}
…
и т.д.
//--- Буква Я ------------------------------------------------------------
void __fastcall TForm1::Label33Click(TObject *Sender)
{
Image1->Picture->LoadFromFile("PICTURE/33.wmf");
PlaySound("SOUND/33.wav", 0,SND_ASYNC);
}
```

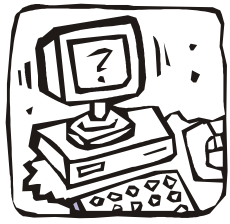
Отличие может быть в расширениях файлов рисунков, поскольку формат этих файлов может быть одним из трех разрешенных, о чем было сказано ранее.

Для изменения значка программы необходимо нарисовать его с помощью инструментальной программы графического редактора **Image Editor**, встроенной в Borland C++ Builder 6 и загрузить этот значок через свойства проекта в программу. О том, как это сделать, было рассказано при описании создания программы **timedate** в *главе 17*.

Теперь необходимо сохранить проект с помощью горячих клавиш <Ctrl>+<S> и выполнить его трансляцию и запуск с помощью клавиши <F9>. Обратите внимание на что, что при раскрытой на весь экран форме быстрые кнопки и главное меню среды разработки становятся недоступны; в этом случае для выполнения команд удобно использовать горячие клавиши.

Если при создании программы не было допущено ошибок, то она сразу начнет работать по приведенному в начале главы алгоритму. Если в программе встретятся ошибки, внесенные вами при ее создании, воспользуйтесь встроенным в среду разработки отладчиком. При необходимости можно полностью свернуть форму. Вновь отобразить форму на экране можно с помощью комбинации клавиш <Shift>+<F12>. Полностью рабочая программа abc.exe со всеми рисунками, звуковыми файлами и исходными текстами доступна в каталоге \PROJECT\ABC на диске, прилагаемом к книге.

Для практики создайте программу для обучения английскому алфавиту по аналогии с программой, описанной здесь. Уверен, для тех, кто смог самостоятельно повторить создание программы обучения русской азбуке, решение этой задачи не составит особого труда.



Глава 20

Свойства компонентов

В этой главе рассказывается об основных свойствах визуальных компонентов.

Рассмотрим основные свойства компонентов среды разработки Borland C++ Builder 6. При разработке программы удобно иметь справочную таблицу этих свойств в алфавитном порядке. Это позволяет оперативно находить нужное свойство, для того чтобы узнать его назначение. В табл. 20.1 описаны основные свойства часто используемых компонентов Borland C++ Builder 6. Эти свойства сходны по смыслу и назначению для различных компонентов.

Таблица 20.1. *Основные свойства компонентов*

Свойство	Назначение
ActiveControl	Определяет, какой элемент управления на форме будет выделен при запуске программы
Align	Выравнивание объекта внутри родителя. Принимает значения: **alMone** — нет выравнивания; **aLTop** — по верхней границе; **alBottom** — по нижней границе; **allLeft** — по левой границе; **alRight** — по правой границе; **alClient** — по всему размеру родителя
Alignment	Горизонтальное выравнивание: **taLeftJustify** — по левому краю; **taRighOustify** — по правому; **taCenter** — по центру
Anchors	Определяет, как объект будет перемещаться или сжиматься при изменении размеров родителя. Например, если включить два под свойства **akLeft** и **akRight**, то ширина объекта будет сжиматься при уменьшении его ширины. По умолчанию не используется
AutoScroll	Если равно **true**, то разрешается автоматическое появление полос прокрутки, когда содержимое объекта не умещается в его видимой части
AutoSize	Если равно **true**, то размер объекта будет автоматически подстраиваться под размер его содержимого
BiDiMode	Способ поведения объектов с учетом национальной специфики — ввод текста и прокручивание информации слева направо (**bdLeftToRighf** по умолчанию) или справа налево (**bdRightToLeft**)

Таблица 20.1 *(продолжение)*

Свойство	Назначение
BorderIcons	Набор значков в системном меню формы
BorderStyle	Вид рамки объекта. Принимает значения: **bsNone** — нет рамки; **bsSi'ngle** — простая рамка; **bsSizeable** — рамка, позволяющая изменять размеры объекта мышью; **bsDialog** — рамка в стиле диалоговых окон; **bsToolWindow** — как **bsSingle**, но с небольшим заголовком; **bsSizeToolWin** — как **bsSizeable**, но с небольшим заголовком
BorderWidth	Ширина рамки
Cancel	Определяет, будет ли происходить для данного объекта событие **OnClick**, когда пользователь нажмет клавишу <Esc>. Для этого значение **Cancel** должно быть равно **true**
Caption	Заголовок. Для одних объектов применяется, чтобы задать заголовок в окне или надпись на кнопке, для других описывает их содержимое (например, у полей надписи)
Checked	Свойство, определяющее состояние флажков (**true** — включен)
Color	Цвет объекта
Columns	Число столбцов
Constraints	Содержит четыре подсвойства, определяющие минимально и максимально допустимые размеры объекта
Cti3D	Вид объекта в объемном стиле (стандарт для Windows 95 и выше)
Cursor	Задает вид указателя, который отображается при наведении на объект
Default	Определяет, будет ли происходить для данного объекта событие **OnClick**, когда пользователь нажмет клавишу <Enter>. Для этого свойство **Default** должно иметь значение **true**
DockSite	Применяется для различных панелей, объектов группирования и т. д. При включении (**true**) позволяет использовать этот объект как базу для стыковки других объектов на форме при перетаскивании. Обеспечивает режим "прилипания" (см. свойство **DragKind**)
Down	Состояние кнопки. Нажата — **true**
DragCursor	Задает вид указателя, который отображается при перетаскивании объекта мышью
DragKind	Определяет, можно ли объект произвольно перетаскивать по окну (**dkDrag**) или же его можно перемещать как стыкуемый объект (**dkDock**), который сам определяет свою форму при стыковке с другими объектами
DragMode	Определяет, можно ли объект перетаскивать (**dmAutomatk**) или нет (**dmManual** — по умолчанию) в соответствии с режимом, указанным в **DragKind**

Таблица 20.1 *(продолжение)*

Свойство	Назначение
Enabled	Доступность объекта. Когда свойство **Enabled** имеет значение **false**, объект становится недоступным для пользователя (например, кнопка делается серой, и щелкнуть на ней нельзя)
FileName	Имя файла. Используется при сохранении выбранного файла в стандартных диалоговых окнах для указания имени файла с мультимедийным содержимым. Применяется для компонентов воспроизведения музыки, звука и т. д.
Flat	Вид границ объекта (**true** — плоские границы)
Font	Определяет шрифт для надписей внутри объекта. Содержит множество подсвойств
FormStyle	Стиль формы. Может принимать значения: **fsNormal** — обычное окно; **fsMDIChild** — внутреннее окно многооконного приложения; **fsMDIForm** — родительское окно многооконного приложения; **fsStayOnTop** — всегда на поверхности экрана
Glyph	Графический файл, содержащий от одного до четырех (отпущено, недоступно, нажато, выделено) изображений графической кнопки
Height	Высота объекта
HelpContext	Номер экрана справочной системы, который будет вызываться при нажатии клавиши <F1> при работе с объектом. Требует знания устройства справочной системы Windows
HelpFile	Имя файла справочной системы для данного приложения или окна
HorzScrollBar	Вид и стиль горизонтальной полосы прокрутки. Состоит из множества подсвойств
Hint	Текст подсказки, которая всплывает при наведении указателя мыши на объект. Эта подсказка будет показываться, если свойство **ShowHint** установлено в **true**
Icon	Имя файла со значком для приложения
Images	Список картинок, которые будут использоваться для показа кнопок на панели инструментов, в пунктах меню и т. д.
ItemIndex	Текущий выбранный элемент в списке **Items**. Нумерация начинается с нуля. Если ничего не выбрано, то значение **ItemIndex** равно –1
Items	Список объектов, хранящих основные данные. Используется во всевозможных списках, меню и наборах
Kind	Вид (тип) объекта. Для разных компонентов может принимать разные значения
Left	Левая координата объекта на родительском объекте
Lines	Список строк. Используется в многострочных текстовых элементах. По структуре похоже на свойство **Items**

Таблица 20.1 *(продолжение)*

Свойство	Назначение
Max	Максимальное значение диапазона, используемого во всевозможных компонентах прокрутки, ползунках и т. п.
MaxLength	Максимально допустимая длина вводимой строки символов
Min	Минимальное значение диапазона, используемого во всевозможных компонентах прокрутки, ползунках и т. п.
ModalResult	Значение, которое возвращает модальное диалоговое окно при его закрытии. Может принимать значения **mrNone** (по умолчанию), **mrOk**, **mrCancel**, **mrAbor**t, **mrRetry**, **mrIgnore**, **mrYes**, **mrNo** и **mrAl**l. Если во время работы диалога в это свойство записать значение, не равное **mrNone**, то диалоговое окно сразу закроется
Name	Название объекта (имя переменной, которая будет использоваться в программе для обращения к этому объекту)
Options	Параметры настройки объекта. Различаются для разных компонентов
Orientation	Ориентация объекта. Может быть горизонтальной (**udHorizontal**) или вертикальной (**udVertical**)
OwnerDraw	Характеризует, будет ли объект рисовать свой образ на экране самостоятельно (**true**) или использовать стандартный вид
Picture	Картинка. Позволяет загрузить графический файл, например, для компонента **image**
PixelsPerInch	Число точек (пикселов) на дюйм экрана. Используется для масштабирования формы в зависимости от экранного разрешения. Будет учитываться, если свойство **Scaled** имеет значение **true**
PopupMenu	Контекстное меню, связанное с объектом, и вызываемое по щелчку правой кнопки мыши над этим объектом. Выбирается в раскрывающемся списке доступных меню и должно быть подготовлено заранее
Position	Для формы — положение на экране. Принимает значения: **poDesigned** — положение окна во время работы программы соответствует положению формы на экране на этапе проектирования; **poDefault** — положение определяется Windows; **poDefaultPosOnly** — используются размеры, заданные разработчиком; **poDefaultSizeOnly** — используется положение, заданное разработчиком; **poScreenCenter** — положение в центре экрана для многомониторных систем; **poDesktopCenter** — положение в центре экрана. Для всевозможных компонентов прокрутки, движков и т. п. — текущее положение движка
Readonly	Если данное свойство имеет значение **true**, то объект будет доступен в режиме "только для чтения". Например, поле ввода с включенным свойством **Readonly** позволяет просматривать содержимое, но не позволяет изменять его

Таблица 20.1 *(окончание)*

Свойство	Назначение
Scaled	Если имеет значение **true**, то учитывается свойство **PixelsPerInch**
Show/Hint	Определяет, надо ли показывать всплывающую подсказку, хранящуюся в свойстве **Hint**
Sorted	Используется во всевозможных списках. Когда включено (**true**), содержимое списка будет автоматически сортироваться перед выводом на экран
Style	Стиль объекта. Используется для разных компонентов (кнопок, списков, шрифтов и т. д.)
TabOrder	Определяет номер объекта при передвижении по элементам управления в родительском окне с помощью клавиши <Tab>. Начинается с нуля
TabStop	Определяет, будет ли происходить выделение данного объекта и остановка на нем при передвижении по элементам управления в родительском окне с помощью клавиши <Tab>
Tag	Свойство, сделанное специально для разработчиков. Оно нигде не используется и предназначено только для хранения числа типа int. **Tag** можно рассматривать как пользовательское свойство и применять его для любых собственных нужд
Text	Содержимое различных текстовых элементов управления (полей ввода и т. п.)
Title	Титул (заголовок) приложения, колонки, сообщения
Top	Верхняя координата объекта на родителе
Transparent	Прозрачность фона объекта. Например, если свойство **Transparent** имеет значение **true** для поля надписи, помещенного на объект-изображение, то отображаться на этом изображении будет только текст, а затирания изображения фоном не произойдет
VertScrollBar	Вид и стиль вертикальной полосы прокрутки. Состоит из множества подсвойств
Visible	Определяет, будет ли виден объект во время работы программы (по умолчанию — **true**)
Width	Ширина объекта
WindowState	Статус окна при его открытии. Принимает значения: **wsNormal** — используются размеры и положение, заданные на этапе проектирования; **wsMinimized** — в свернутом виде; **wsMaximized** — в развернутом виде

Большинство компонентов среды разработки BCB6 имеют одинаковые свойства, определяющие их имя, заголовок, размеры, скрытость и т. д. Познакомившись с одним из таких свойств на примере одного компонента, становится

понятным его назначение и для другого компонента. Поэтому мы будем рассматривать основные свойства компонентов, не привязываясь к конкретному их типу.

Самыми распространенными значениями, которые могут иметь большинство свойств, являются **true** (Истина) и **false** (Ложь). Или по простому, Да и Нет. Если свойство принимает значение **true**, значит, оно активизируется. При значении **false** это свойство блокируется. Свойства любого компонента можно изменить во время создания программы с помощью инспектора объектов. При этом для большинства свойств эти изменения можно зрительно увидеть по тому, как изменится при этом сам компонент. В примере программы "Цифровые часы-календарь" *главы 17* уже показывалось, как изменения свойств компонента влияют на его форму, цвет и т. п.

Все свойства объекта доступны не только на этапе проектирования, но и во время работы программы. Их значения можно изменять и в ходе ее выполнения, в программном коде. Например, строка программы, приведенная в листинге 20.1, изменяет свойство **Picture** компонента **Image1** путем загрузки **LoadFromFile** графического файла 1.bmp из каталога PICTURE.

Листинг 20.1. Пример строки программы

```
Image1->Picture->LoadFromFile("PICTURE/1.bmp");
```

Свойство компонентов *Parent*

В описании свойств часто встречается программный термин "родитель". Свойства компонентов, начинающиеся со слова **Parent** (Родитель), определяют общение компонента со своим родителем — объектом, на котором он расположен. Например, когда поле ввода помещается на форму, то для него родителем будет форма. Если командная кнопка размещена на панели инструментов, то для нее родителем будет панель и т. д. В большинстве прикладных задач эти свойства можно не учитывать.

Расширенные свойства

Некоторые из перечисленных в таблице свойств имеют расширенные свойства (подсвойства). Примером такого свойства является **Font**. Ранее мы уже использовали данное свойство при создании программы обучения азбуке. Настройка подсвойства **Font** не представляет проблем, поскольку поддерживается оконным интерфейсом, знакомым пользователям Windows по приложениям Word, Excel и т. д.

Еще одним свойством с множеством подсвойств является **Anchors** (Якоря). Его подсвойства **akLeft**, **akTop**, **akRight** и **akButtom** со значениями **true**

и **false** позволяют смещать (фиксировать) компонент соответственно влево, вверх, вправо или вниз.

Свойство *Color*

Довольно часто при разработке программ используется свойство **Color**. Поскольку современные графические средства позволяют передавать миллионы цветовых оттенков, количество значений свойства **Color** довольно велико. Тем не менее ограниченный набор этих значений имеет индивидуальные названия, которые приведены в табл. 20.2 вместе с их описанием.

Таблица 20.2. *Цветовые константы*

Цветовая константа	Цвет
clAqua	Цвет морской воды
clBlack	Черный
clBlue	Синий
clCream	Кремовый
clDkGray	Темно-серый
clGray	Серый
clGreen	Зеленый
clime	Ярко-зеленый (липовый)
clLtGray	Светло-серый
clMaroon	Бордовый
clMedGray	Средне-серый
clMoneyGreen	Денежный (долларовый)
clNavy	Цвет военно-морского флота
clOlive	Оливковый
clPurple	Пурпурный
clRed	Красный
clSilver	Серебряный
clSkyBlue	Небесно-голубой
clTeal	Цвет "чирок"
clWhite	Белый
clYellow	Желтый
ClScrollBar	Цвет полос прокрутки

Таблица 20.2 *(окончание)*

Цветовая константа	Цвет
clBackground	Цвет фона рабочего стола
clActiveCaption	Цвет заголовка активного окна
clInactiveCaption	Цвет заголовка неактивного окна
clMenu	Цвет фона меню
clWindow	Цвет фона окон
clWindowFrame	Цвет рамок окон
clMenuText	Цвет текста меню
clWindowText	Цвет текста окон
clCaptionText	Цвет текста заголовка неактивного окна
clActiveBorder	Цвет бордюра активного окна
clInactiveBorder	Цвет бордюра неактивного окна
clAppWorkSpace	Цвет рабочей области компонента
clHighlight	Цвет фона выделенного текста
clHightlightText	Цвет выделенного текста
clBtnFace	Цвет поверхности кнопок
clBtnShadow	Цвет теней кнопок
clGrayText	Цвет текста недоступных элементов
clBtnText	Цвет текста кнопок
clInactiveCaptionText	Цвет текста заголовка неактивного окна
clBtnHighlight	Цвет выделенной кнопки
cl3DDkShadow	Для Windows 95 и NT. Цвет темных теней трехмерных компонентов
cl3DLight	Для Windows 95 и NT. Цвет светлых частей трехмерного компонента
clInfoText	Для Windows 95 и NT. Цвет текста всплывающих подсказок
clInfoBk	Для Windows 95 и NT. Цвет фона всплывающих подсказок
clGradientActiveCaption	Для Windows 98 и Windows 2000. Цвет правой стороны градиентной заливки названия активного окна
clGradientInactiveCaption	Для Windows 98 и Windows 2000. Цвет правой стороны градиентной заливки названия неактивного окна

Для использования этих значений в программе необходимо присваивать их название свойству **Color** компонента. Например, строка, приведенная в листинге 20.2, изменяет свойство **Color** компонента **Label1** на **clGreen**, то есть на зеленый цвет.

Листинг 20.2. Второй пример строки программы

```
Label1->Color= clGreen;
```

Ранее мы уже применяли некоторые компоненты в создаваемых программах. Рассмотрим применение других компонентов, которые часто используются в программах, с одновременным изучением их свойств.

Применение компонента *MainMenu*

При создании программ довольно часто требуется включить в программу главное меню. Этот компонент присутствует практически во всех программах, работающих под управлением операционной системы Windows. Для внедрения главного меню в программу используется компонент **MainMenu**, который располагается на вкладке **Standard** палитры компонентов. Запустите среду разработки Borland C++ Builder 6 и разместите на форме этот компонент. Место его размещения на форме не имеет значения. Теперь необходимо произвести настройку пунктов меню. Для этого существует дизайнер меню, встроенный в среду разработки. Вызов дизайнера осуществляется двойным щелчком левой кнопки мыши по компоненту **MainMenu** на форме или одинарным щелчком по кнопке с тремя точками в поле **Menu** свойства **Items** на вкладке **Properties** инспектора объектов. При этом появится окно дизайнера меню (рис. 20.1.), а в инспекторе кодов откроются дополнительные свойства. Введите в свойстве **Caption** название первого пункта меню, например **File**.

После ввода названия и нажатия клавиши <Enter> фокус в окне дизайнера перемещается на следующий пункт, правее первого. После одинарного щелчка по этому пункту в инспекторе объектов можно произвести настройку его свойств. Введите в свойстве **Caption** для этого пункта меню название **Edit**. Так осуществляется наращивание пунктов меню по горизонтали. Для добавления команд в пункты меню необходимо щелкнуть левой кнопкой мыши по уже созданному пункту меню, например, по пункту **File**. При этом ниже пункта появится поле для команды. Введите в свойстве **Caption** для этого поля команды название **Open**. После нажатия клавиши <Enter> фокус в окне дизайнера перемещается на следующее поле команды, ниже первого. Так осуществляется наращивание пунктов команд по вертикали.

Рис. 20.1. Окно дизайнера меню

Если перед какой-либо буквой названия команды вставить символ & (амперсанд), то следующая за этим символом буква будет подчеркнута. Этот способ выделения буквы в названии применяется для показа буквы, которая используется для выполнения команды по горячей клавише, или комбинации со специальными клавишами типа <Ctrl>, <Alt> и <Shift>.

Для создания разделителя команд на группы введите для очередного поля команды в свойстве **Caption** название в виде знака "минус". После нажатия клавиши <Enter> это поле превратится в разделитель команд. Введите в следующее поле команды название **Close** для свойства **Caption**. В результате окно дизайнера меню примет вид, показанный на рис. 20.2.

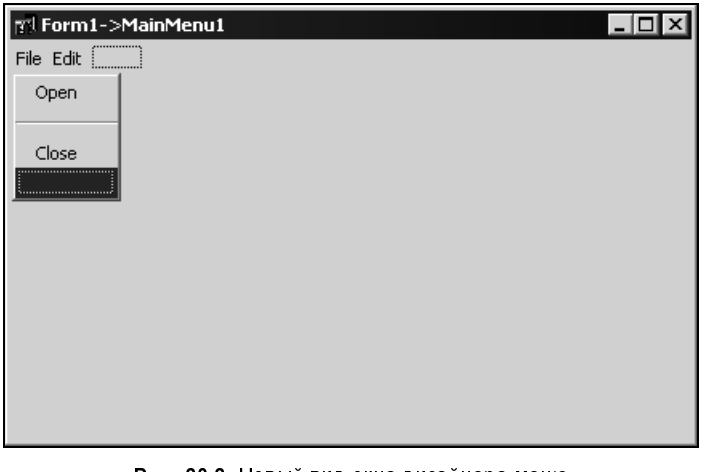


Рис. 20.2. Новый вид окна дизайнера меню

На этом заканчивается создание меню, и окно дизайнера меню можно закрыть. Теперь на форме присутствует главное меню. Но оно является лишь оболочкой и не способно выполнять команды, присутствующие в нем. Для того чтобы команды меню выполнялись, нужно создать функции обработки этих команд. Сделать это очень просто. Необходимо раскрыть щелчком левой кнопки мыши меню и щелкнуть по команде, для которой необходимо создать функцию обработки. При этом откроется окно инспектора кодов с заготовкой такой функции. В эту заготовку вводится исполняемый код программы, например, `Form1->Close();` для команды **Close** пункта меню **File**. Данная команда приведет к закрытию приложения при выборе команды **Close**.

Аналогично вставляются коды программы для остальных пунктов меню. Причем допустимые к использованию коды программы (функции) предложит сама среда разработки после того, как будет введена строка `Form1->`. На экране при этом откроется окно (рис. 20.3) с разрешенными функциями, из которых можно выбрать необходимую команду. Запустите созданную программу на выполнение и проверьте, как она работает.

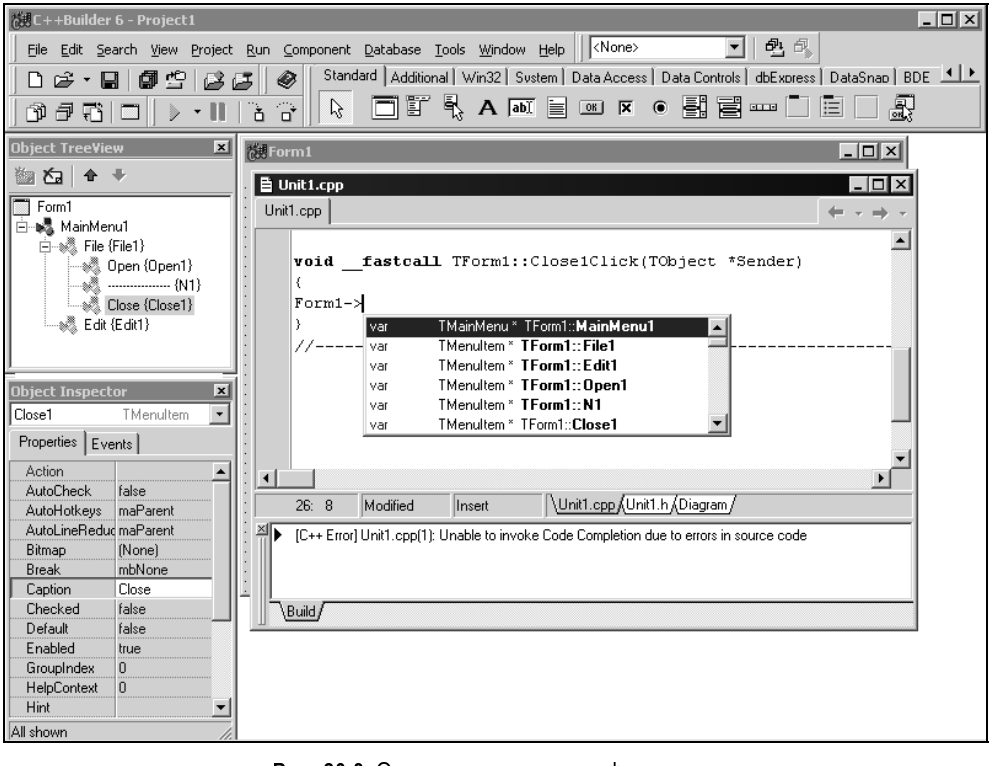


Рис. 20.3. Окно с разрешенными функциями

Применение компонента *Memo*

Теперь рассмотрим применение компонента **Memo** из той же вкладки **Standard** палитры компонентов. С его помощью на форме создается прямоугольная область для многострочного ввода.

Создайте новое приложение и поместите на форму компонент **Memo**. Переместите этот компонент на форме так, чтобы он был расположен в ее верхней левой четверти (квадранте). Произведем настройку свойств данного компонента. С помощью свойства **Color** зададим цвет **clSilver** для фона компонента. С помощью свойства **Cursor** можно задать тип курсора, который будет принимать выбранный нами вид при попадании на область ввода компонента **Memo** при работе программы. Процедура выбора типа курсора представлена на рис. 20.4.

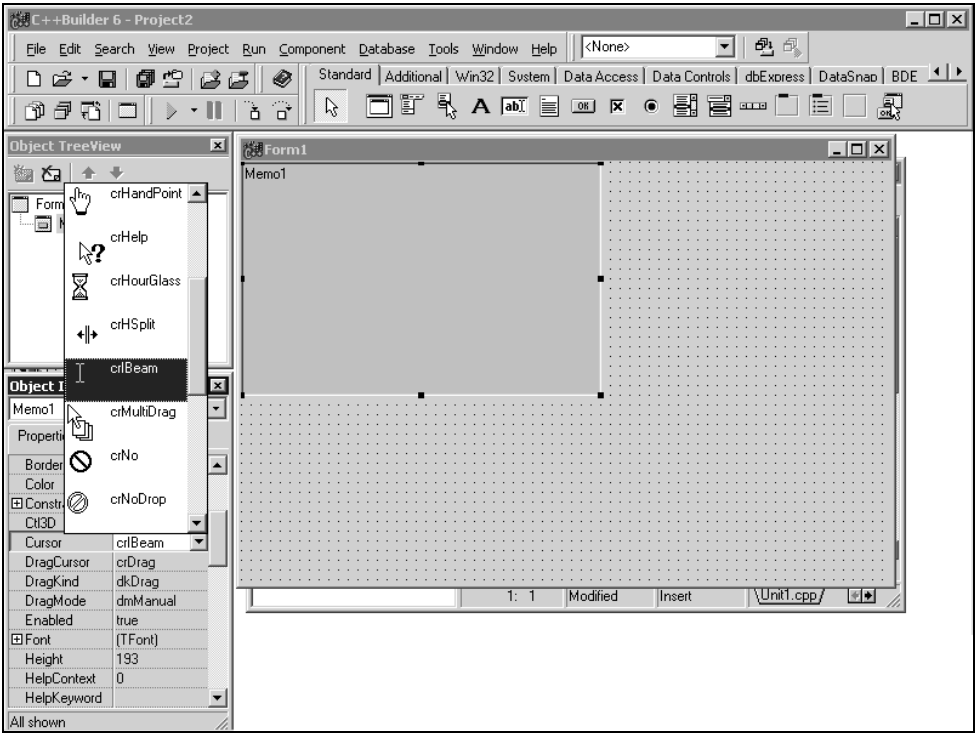


Рис. 20.4. Процедура выбора типа курсора

Задайте для свойства **Cursor** тип курсора **crIBeam**. На рисунке он выделен среди других типов курсора.

Для задания текста, отображаемого в поле компонента **Memo**, необходимо воспользоваться свойством **Lines** и вызвать окно редактора строки

String List Editor, щелкнув левой кнопкой мыши по кнопке с тремя точками правее свойства **Lines**. Измените свойство **Lines** так, как показано на рис. 20.5, просто набрав приведенный на рисунке текст в окне редактора строки **String List Editor**.

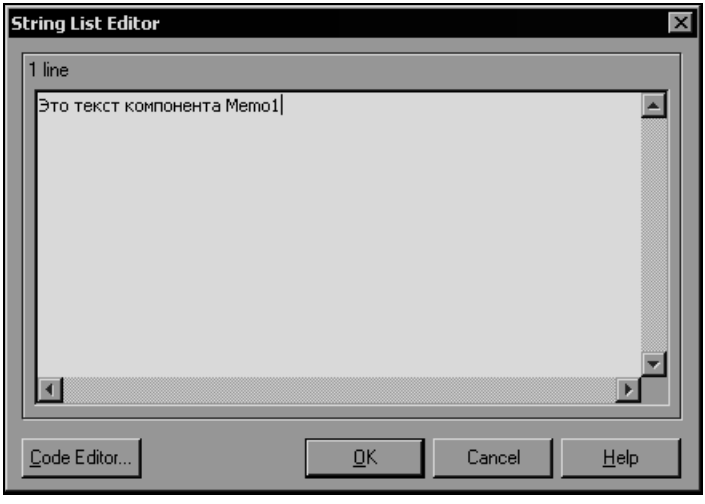


Рис. 20.5. Окно редактора строки **String List Editor**

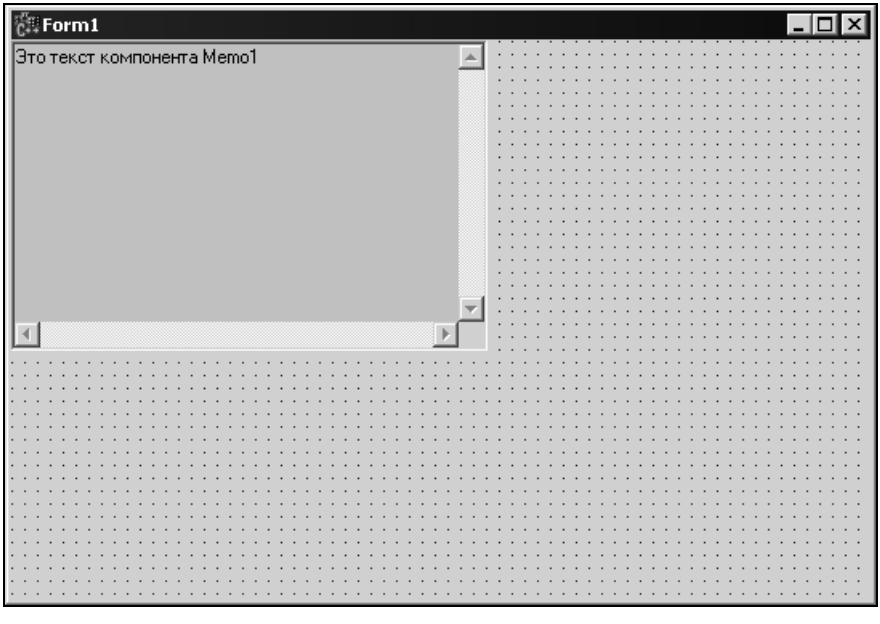


Рис. 20.6. Конечный вид формы

Часто вводимый текст не помещается в окно компонента **Memo** при выполнении программы. Для устранения этой проблемы можно добавить полосы прокрутки для компонента **Memo**. Это осуществляется с помощью свойства **ScrollBars**. Выберите в этом свойстве горизонтальную **ssHorizontal** или вертикальную **ssVertical** полосы прокрутки. Можно выбрать обе полосы прокрутки, выбрав значение свойства **ssBoth**. Конечный вид формы с применением компонента **Memo** приведен на рис. 20.6.

Позже мы рассмотрим пример создания программы текстового редактора, где будет рассмотрено применение компонентов **Memo** и **MainMenu** с множеством пунктов, снабженных кодами программы.

Применение компонента *ImageList*

Еще один полезный компонент **ImageList** из вкладки **Win32** палитры компонентов часто применяется для хранения изображений для графического оформления программы и создания анимации в программе. Создайте новое приложение и поместите на его форму этот компонент. Дважды щелкните по нему левой кнопкой мыши. При этом откроется окно редактора ресурсов компонента, приведенное на рис. 20.7.

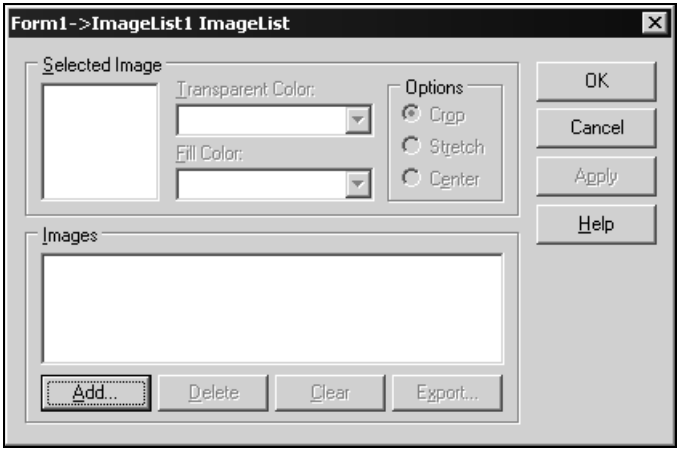


Рис. 20.7. Окно редактора ресурсов компонента

Для добавления изображения в компонент необходимо щелкнуть по кнопке **Add** и в открывшемся диалоговом окне выбрать файл с рисунком в формате BMP или ICO. Этому рисунку будет автоматически присвоен нулевой индекс. Следующий загружаемый рисунок будет иметь очередной индекс. Все загружаемые рисунки должны иметь одинаковые размеры.

Множество файлов с рисунками хранится в каталоге C:\ProgramFiles\CommonFiles\BorlandShared\Images, созданном автоматически после установки Borland C++ Builder 6.

Компонент **ImageList** относится к типу невидимых компонентов и может располагаться в любом месте на форме. Он используется как контейнер рисунков для других компонентов, которые извлекают эти рисунки из него по мере необходимости.

Применение компонента *BitBtn*

Довольно часто в программе используются кнопки с широко распространенными названиями типа **OK**, **Yes**, **No** и т. п. В среде разработки имеется возможность определять вид таких кнопок путем применения компонента кнопок **BitBtn** и изменения всего одного его свойства **Kind**. На рис. 20.8. показана форма приложения, на которой расположено 11 кнопок со всеми возможными состояниями свойства **Kind**.

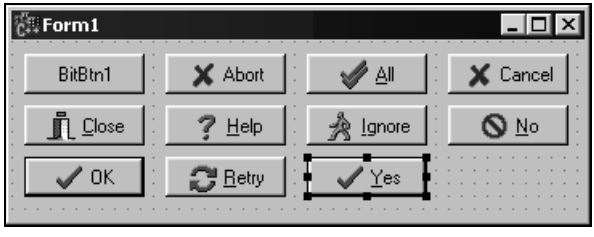
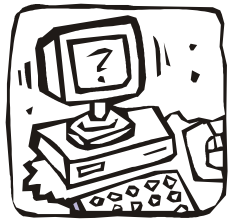


Рис. 20.8. Форма приложения с кнопками

Таким образом, используя данный компонент, можно без особого труда вставить в разрабатываемую программу кнопки с широко распространенными обозначениями и соответствующими им рисунками.



Глава 21

Текстовый редактор

В этой главе рассказывается о том, как легко можно создать в среде разработки Borland C++ Builder 6 программу текстового редактора.

В *главе 20* мы рассмотрели свойства компонентов. В данной главе на примере создания программы текстового редактора познакомимся с применением этих свойств на практике.

Запустите Borland C++ Builder 6 и создайте новый проект с именем edit. Поскольку текстовый редактор предназначен для работы с текстовыми строками, нам понадобится компонент **Memo** из вкладки **Standard**. Поместите его в центре формы проекта. Теперь поместите на форму компонент **MainMenu** из той же вкладки. Наконец, добавьте на форму три стандартные кнопки **BitButton** из вкладки **Additional**. Они понадобятся для формирования в программе органов управления для открытия и сохранения файлов и для других функций. Поскольку редактор должен при открытии и сохранении файлов обращаться к дисковым накопителям, необходимо добавить на форму диалоги **Open Dialog** и **Save Dialog** из вкладки **Dialogs**. Кроме того, добавьте на форму диалог **Font Dialog** из этой же вкладки. Он облегчит нам задачу настройки шрифтов из программы. Эти компоненты являются невидимыми при работе программы, поэтому место их размещения на форме не имеет значения. Но все же лучше располагать такие компоненты без наложения, для того чтобы они не скрывали друг друга. Последний компонент, который необходимо поместить на форму, — **PopMenu** из вкладки **Standard**. Он позволит создать в программе контекстное меню, появляющееся при нажатии правой кнопки мыши.

Теперь разместим выбранные компоненты на форме и настроим их свойства. Начнем с главного компонента — формы приложения. Выберите в инспекторе объектов в поле селектора объектов **Form1**. Измените свойство **Caption** этого объекта на название программы редактора **Edit**. Размеры формы установите с помощью мыши, захватив любой угол или сторону формы и перемещая их в любом направлении. Размер формы можно будет поправить

при необходимости на любом этапе разработки программы. В свойстве **Position** установите значение **poDesigned**. Это обеспечит центровку нашего приложения при запуске по центру экрана. Щелкните по компоненту **MainMenu** левой кнопкой мыши. При этом откроется окно дизайнера меню. В свойстве **Caption** инспектора объектов введите название **&File** и нажмите клавишу <Enter>. На форме приложения при этом появится строка главного меню. Таким образом, на форме можно будет видеть поле главного меню и учитывать его размеры. Теперь можно временно закрыть окно дизайнера меню и приступить к размещению остальных компонентов на форме. Вначале разместите на форме компонент **Memo** так, чтобы он занимал всю ее свободную область за исключением правой полосы для кнопок. Кнопки расположите друг под другом, не изменяя при этом их размеров. В оставшейся свободной части формы можно расположить оставшиеся невидимые компоненты приложения. Не забывайте, что перемещать компоненты можно группами, предварительно выделив эти компоненты с помощью мыши. В результате после размещения всех компонентов должна получиться форма, подобная приведенной на рис. 21.1.

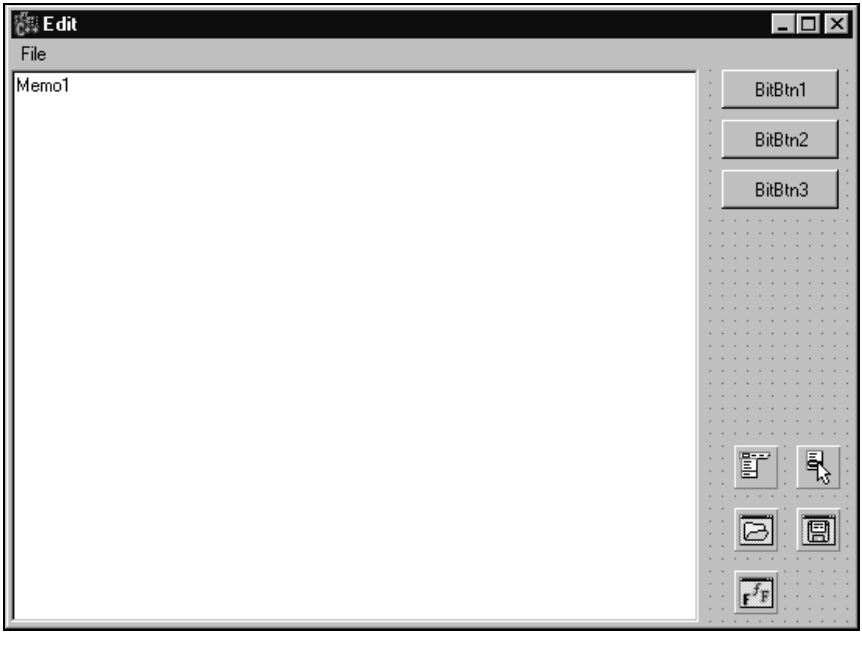


Рис. 21.1. Окно формы программы

Теперь приступим к изменению свойств остальных компонентов. Начнем с компонента **Memo**, выбрав его в инспекторе объектов. Вначале удалим надпись **Memo1** с компонента, для того чтобы она не появлялась при запуске приложения. Для этого изменим свойство **Lines**, щелкнув по кнопке

с тремя точками правее надписи **Tstrings** этого свойства. При этом откроется окно редактора строки **String List Editor**, в котором необходимо удалить строку **Memo1** и нажать кнопку **OK**. Теперь добавим для этого компонента линейки прокрутки для обеспечения возможности просмотра и редактирования больших областей текста с помощью мыши. Для этого в свойстве **ScrollBars** необходимо выбрать значение **ssBoth**. Для того чтобы работало контекстное меню, необходимо в свойстве **PopupMenu** выбрать значение **PopupMenu1**. Это значение стало доступным после помещения на форму проекта диалога **PopMenu**. Настройку данного меню произведем позже.

Настроим свойства кнопок в соответствии с выполняемыми ими функциями. Для этого изменим поочередно свойство **Caption** для всех трех кнопок на **Open**, **Save** и **Font** соответственно. Для того чтобы тип курсора изменялся при попадании на эти кнопки, изменим еще одно групповое свойство кнопок. Для этого с помощью мыши выделите все три кнопки, и в появившемся объекте **3 items selected** в инспекторе объектов измените свойство **Cursor** на значение **crHandPoint**.

Настройку главного меню произведем с помощью дизайнера меню, вызываемого двойным щелчком левой кнопки мыши по компоненту **MainMenu**. Ранее мы уже ввели один пункт меню под названием **&File**. Символ "амперсанд" перед **File** производит выделение первой буквы названия в меню для быстрого доступа, путем ее подчеркивания. Следующий по горизонтали пункт меню назовем **&Edit**. Теперь заполним пункты меню по вертикали. Щелкните левой кнопкой мыши по пункту меню **File**, а затем ниже него по пустому полю. Свойство **Caption** этого пункта меню измените на **Open** и нажмите клавишу <Enter>. Следующий пункт назовите **Save**. Аналогично расширим по вертикали пункты меню **Edit**, добавив пункты **Font** и **Clear**.

На этом этап создания главного меню программы редактора завершен. Все введенные пункты меню теперь будут присутствовать в приложении, но пока не будут выполняться, поскольку не созданы функции для выполнения соответствующих команд. Создадим их. Для этого воспользуемся компонентами диалогов, помещенных ранее на форму приложения. С помощью диалогов работы с файлами можно задать направленность приложения путем определения типов открываемых файлов. Это делается с помощью свойства **Filter**.

Выберите в инспекторе объектов компонент **OpenDialog1** и нажмите кнопку с тремя точками правее свойства **Filter**. При этом появится диалоговое окно **Filter Editor**, в котором производится настройка типов файлов. Окно разбито на два поля. В левом поле **Filter Name** вводятся строки пояснения. В правом поле **Filter** — расширения файлов. Для указания нескольких типов файлов необходимо записать их через точку с запятой. Если задать несколько строк с пояснениями и типами файлов, то при работе приложения их можно будет выбирать с помощью выпадающего списка. Заполните оба поля диалогового окна **Filter Editor** в соответствии с рис. 21.2 и нажмите кнопку **OK**.

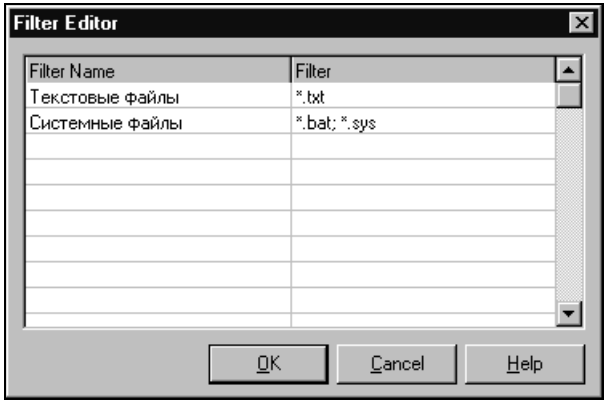


Рис. 21.2. Окно **Filter Editor**

Аналогично выполните настройку компонента **SaveDialog1**. Для корректности работы приложения оба эти диалога должны иметь одинаковую настройку свойства **Filter**. Чтобы облегчить эту работу и избежать ошибок, можно использовать два способа. Первый способ уже был показан ранее, когда несколько компонентов кнопок выделялось в группу и задавалось свойство одновременно для всех выделенных компонентов. Второй способ заключается в том, что, задав свойство для одного из компонентов, можно скопировать и вставить его для других компонентов с помощью комбинации клавиш <Ctrl>+<C> и <Ctrl>+<V> соответственно. Так, после задания свойства **Filter** компонента **OpenDialog1** с помощью **Filter Editor** в поле этого свойства появится запись: Текстовые файлы |*.txt| Системные файлы |*.bat; *.sys. Выделив эту строку с помощью клавиш <Home> и <Shift>+<End>, скопируйте ее с помощью клавиш <Ctrl>+<C>. Теперь можно вставить эту строку в свойство **Filter** для компонента **SaveDialog1**. Замечательно то, что работу этих компонентов можно проверить еще до выполнения программы. Дело в том, что после задания свойства **Filter** этих компонентов двойной щелчок по компонентам приводит к открытию диалога на основе заданных свойств. Например, после двойного щелчка по компоненту **SaveDialog1** откроется окно, изображенное на рис. 21.3.

Теперь настроим компонент **FontDialog1**. Основным свойством данного компонента является **Font**, с помощью которого можно выбрать тип шрифта, его цвет, размер, стиль и т. п. Эти установки будут использоваться в программе редактора по умолчанию. С помощью свойства **Options** можно настроить вид и поведение диалога. Например, можно добавить в него кнопку справки, установив для подсвойства **fdShowHelp** значение **true**.

Нам осталось произвести настройку контекстного меню **PopMenu**. Она производится аналогично настройке главного меню **MainMenu**. Добавим в это меню пункты с названиями **Font** и **Clear** через свойство **Caption**, как это показано на рис. 21.4.

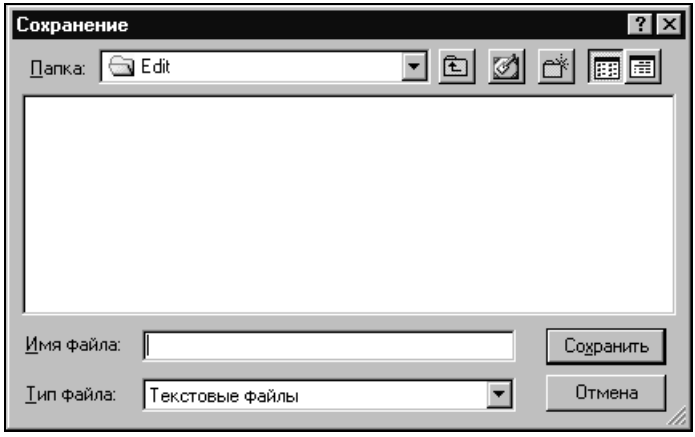


Рис. 21.3. Окно компонента **Save Dialog**

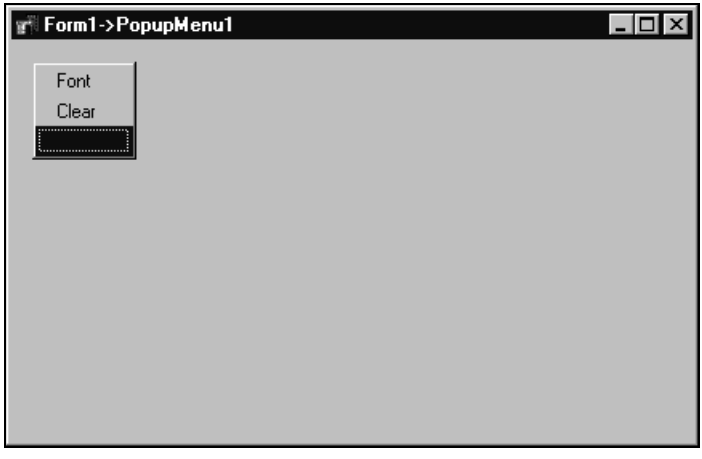


Рис. 21.4. Настройка контекстного меню **PopMenu**

Кроме того, добавьте название **PopupMenu1** в свойство **PopupMenu** главной формы приложения.

На этом настройка компонентов завершена и получена визуальная оболочка программы редактора. Однако без программных строк это приложение не будет работать. При запуске приложения будут открываться пункты меню, вводиться и редактироваться текст, но только потому, что эти функции заложены в сами компоненты. Нажатие на кнопки и вызов команд из главного меню не будут приводить к каким-либо действиям программы, поскольку не заданы функции обработки событий. Вдохнем в приложение жизнь. В программе редактора имеется четыре типа операций. Это открытие и сохранение файлов, настройка шрифтов и очистка окна редактора. Все эти операции можно вызвать разными способами. Например, с помощью главного меню,

кнопок или контекстного меню. Можно написать функции для каждого способа, но это усложнит программу и внесет трудности при ее создании и отладке. Правильнее будет создать такую функцию один раз, а затем задать ее выполнение другим компонентам с помощью инспектора объектов. Создайте функцию для команды главного меню **Open**. Для этого откройте на главной форме пункт меню **File** и щелкните левой кнопкой мыши по команде **Open**. При этом активизируется редактор кода с заготовкой функции обработки. Введите между фигурными скобками следующий текст программы:

```
if (OpenDialog1->Execute())
 Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

Обратите внимание на то, что среда разработки сама предоставляет выбрать функцию из множества разрешенных для компонента функций после введения символа стрелки. Эти программные строки позволят при запуске диалога открытия файла загрузить содержимое открываемого файла в свойство **Lines** компонента **Memo1** для отображения на экране. Теперь задайте выполнение этой функции кнопке команды **Open** с помощью инспектора объектов. Для этого щелкните левой кнопкой мыши по кнопке **Open** и выберите в инспекторе объектов вкладку **Events** (События). Раскройте список в событии **OnClick** и выберите в нем строку **Open1Click**. Таким образом будет задано, что при нажатии кнопки в программе редактора запустится та же функция открытия файла, что и для команды **Open** из пункта главного меню **File**. Запустите программу с помощью команды **Run** и убедитесь, что команды открытия файла работают и выбирают заданные нами типы файлов.

Аналогично создадим функции для оставшихся трех типов команд. Начнем с команды сохранения файлов **Save**. Откройте на главной форме пункт меню **File** и щелкните левой кнопкой мыши по команде **Save**. При этом активизируется редактор кода с заготовкой функции обработки. Введите между фигурными скобками следующий текст программы:

```
if (SaveDialog1->Execute())
 Memo1->Lines->SaveToFile(SaveDialog1->FileName);
```

Эти программные строки позволят при запуске диалога сохранения файла загрузить содержимое свойство **Lines** компонента **Memo1** в сохраняемый файл. Задайте выполнение этой функции кнопке с командой **Save** с помощью инспектора объектов. Для этого щелкните левой кнопкой мыши по кнопке **Save** и выберите в инспекторе объектов вкладку **Events**. Раскройте список в событии **OnClick** и выберите в нем строку **Save1Click**. Таким образом будет задано, что при нажатии кнопки **Save** в программе редактора запустится та же функция сохранения файла, что и для команды **Save** из пункта главного меню **File**.

Для команды **Font** из пункта **Edit** главного меню введите с помощью инспектора кодов строки:

```
if (FontDialog1->Execute())
 Memo1->Font=FontDialog1->Font;
```

Это позволит изменять шрифт текста компонента **Memo1** при открытии диалога **Font Dialog1**. Задайте выполнение этой функции кнопке с командой **Font** с помощью инспектора объектов, для чего раскройте список в событии **OnClick** и выберите в нем строку **Font1Click**.

Теперь нужно добавить вызов этой функции в контекстное меню. Выделите на форме компонент **PopupMenu1** и в инспекторе объектов щелкните по кнопке с тремя точками в строке свойства **Items**. При этом откроется окно **Form1 | PopupMenu**. В этом окне щелкните на поле **Font** и в инспекторе объектов на закладке **Events** выберите для события **OnClick** строку **Font1Click**.

Наконец создадим функцию обработки очистки редактора. Для команды **Clear** из пункта **Edit** главного меню введите с помощью инспектора кодов одну-единственную строку:

```
Memo1->Clear();
```

Эта программная строка будет производить очистку содержимого компонента **Memo1**. Аналогично предыдущему описанию назначьте выполнение данной функции через контекстное меню, выбрав строку **Clear1Click** для события **OnClick**.

На этом создание программы редактора заканчивается, и можно приступить к его проверке. Предварительно сохраните весь проект в отдельном каталоге, например, с именем **Edit**, а затем запустите его на выполнение. На экране должно появиться окно, аналогичное изображенному на рис. 21.5.

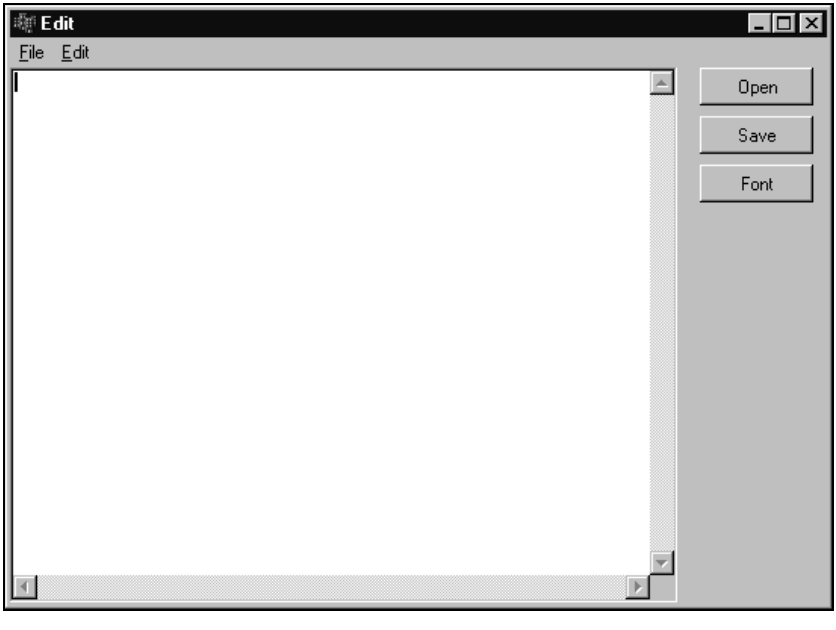
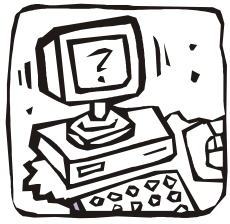


Рис. 21.5. Окно работающей программы

Нажмите кнопку **Open** и выберите какой-либо файл на диске. После чего загрузите его в редактор. Измените с помощью клавиатуры текст файла и сохраните его с новым именем, нажав кнопку **Save**. Нажмите кнопку **Font** и измените шрифт текста редактора. Убедитесь в том, что текст изменился в соответствии с вашими установками. Из главного меню выполните команду **Clear** и убедитесь, что поле редактирования очистилось. Проверьте работу остальных пунктов главного меню. Далее проверьте работу контекстного меню, нажав правую кнопку мыши на поле редактора и выбрав одну из двух команд. Обратите внимание на то, что при наведении курсора на кнопки он должен менять свой вид.

Редактор готов к работе. Создайте для него иконку, как это делали раньше, и используйте редактор как отдельное законченное приложение. Со временем вы сможете добавлять в него новые функции и возможности, повышая тем самым его функциональность и свое мастерство.

Готовую программу текстового редактора edit.exe вместе с проектом можно найти в каталоге \PROJECT\EDIT на диске, прилагаемом к книге.



Глава 22

События компонентов

В этой главе рассказывается о событиях визуальных компонентов и примерах их использования при создании программ.

Мы неоднократно рассматривали применение различных событий компонентов в примерах программ. Настало время рассмотреть каждое из них в отдельности. В табл. 22.1 приведены основные события компонентов и даны описания ситуаций, когда они происходят.

Таблица 22.1. *Основные события компонентов и описание ситуаций, когда они происходят*

Событие	Когда происходит событие
OnActivate	При активизации объекта
OnCanResize	При изменении размеров
OnContextPopup	При вызове контекстного меню
OnClick	По щелчку мыши на компоненте и при другой активизации компонента
OnClose	При закрытии
OnDblClick	По двойному щелчку мыши на компоненте
OnChange	После изменения графического объекта
OnDragDrop	В момент отпускания перетаскиваемого компонента над данным компонентом
OnDragOver	Когда пользователь перемещает перетаскиваемый объект над компонентом
OnEndDrag	В момент прерывания или окончания перетаскивания компонента
OnEnter	В момент получения элементом фокуса

Таблица 22.1 *(окончание)*

Событие	Когда происходит событие
OnExit	В момент потери элементом фокуса
OnKeyDown	При нажатии пользователем любой клавиши
OnKeyPress	При нажатии пользователем клавиши символа
OnKeyUp	При отпускании пользователем любой клавиши
OnMouseUp	В момент нажатия пользователем клавиши мыши над компонентом
OnMouseMove	При перемещении курсора мыши над компонентом
OnPaint	При получении сообщения Windows о необходимости перерисовать изображение
OnProgress	Во время медленных процессов изменения графического изображения
OnStartDrag	Когда пользователь начал перетаскивание объекта
OnEnter	При вводе
OnExit	При выходе

Событие *OnClick*

Самым часто используемым и уже знакомым нам событием является **OnClick**. Обычно событие **OnClick** наступает, если пользователь нажал и отпустил левую кнопку мыши в то время, когда указатель мыши находился на компоненте. Кроме того, это событие происходит в следующих случаях:

- пользователь нажал клавишу пробела, когда кнопка или индикатор были в фокусе;
- пользователь нажал клавишу <Enter>, а активная форма имеет кнопку по умолчанию, указанную свойством **Default**;
- пользователь нажал клавишу <Esc>, а активная форма имеет кнопку прерывания, указанную свойством **Cancel**;
- пользователь нажал клавиши быстрого доступа к кнопке или индикатору;
- пользователь выбрал элемент в сетке, дереве, списке или выпадающем списке, нажав клавишу со стрелкой;
- приложение установило в **true** свойство **Checked** радиокнопки **RadioButton**;
- приложение изменило свойство **Checked** индикатора **CheckBox**;
- вызван метод **Click** элемента меню.

Как видим, данное событие может происходить в большинстве случаев. Недаром оно часто используется в программах.

Для компонента формы событие **OnClick** наступает, если пользователь щелкнул на пустом месте формы или на недоступном компоненте.

Создайте новое приложение и поместите на форму одну кнопку **Button1**. Откройте для нее в инспекторе объектов вкладку событий **Events** и щелкните дважды левой кнопкой мыши на поле правее события **OnClick**. При этом откроется инспектор кодов и появится заготовка программного кода (листинг 22.1) для функции обработки события **OnClick**, создаваемая автоматически средой разработки Borland C++ Builder 6.

Листинг 22.1. Функции обработки события OnClick

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
}
```

Параметр **Sender** данного события содержит объект, в котором произошло событие, и может быть использован для извлечения информации о том, какой компонент сформировал событие **OnClick**.

Впишите между фигурными скобками функции обработки события **OnClick** для кнопки **Button1** следующую программную строку:

```
ShowMessage("Событие OnClick произошло в "+((TControl *)Sender)->Name);
```

Аналогичные действия выполните для формы **Form1**, после чего запустите приложение. В результате при возникновении события **OnClick** на кнопке или форме будет формироваться окно сообщения с информацией о компоненте, в котором произошло данное событие. Проверьте реакцию программы на щелчок левой кнопкой мыши по кнопке, нажатию клавиш <пробел> и <Enter> и другие действия пользователя программы.

Событие *OnDblClick*

Другое событие — **OnDblClick** наступает, если пользователь осуществил двойной щелчок, то есть дважды нажал и отпустил основную кнопку мыши с коротким интервалом, когда указатель мыши находился на компоненте. К одному и тому же компоненту нельзя написать обработчики событий **OnClick** и **OnDblClick**, поскольку первый из обработчиков всегда перехватит щелчок мыши. Параметр **Sender** для события **OnDblClick** может быть использован аналогично.

Довольно часто в программах необходимо определять, какую клавишу на клавиатуре нажал пользователь, для того чтобы в соответствии с ней назначить

выполнение определенных действий. Для этих целей хорошо подходят события, описываемые далее.

Событие *OnKeyDown*

Событие **OnKeyDown** наступает, если компонент находится в фокусе, при нажатии пользователем любой клавиши, включая функциональные и специальные, такие как <Shift>, <Alt> и <Ctrl>. Кроме обычного параметра **Sender**, указывающего на компонент, в котором произошло событие, в обработчик события передаются такие параметры, как **Key** и **Shift**. Параметр **Key** позволяет определить код нажатой клавиши клавиатуры.

Для большинства клавиш имеются также символьные имена API Windows. Коды клавиш и соответствующих им символьных имен приведены в табл. 22.2.

Таблица 22.2. *Коды клавиш и соответствующих им символьных имен*

Клавиша	Цифровое значение кода		Символьное имя
	десятичное	шестнадцатеричное	
На основном поле клавиатуры			
F1	112	0x70	VK_F1
F2	113	0x71	VK_F2
F3	114	0x72	VK_F3
F4	115	0x73	VK_F4
F5	116	0x74	VK_F5
F6	117	0x75	VK_F6
F7	118	0x76	VK_F7
F8	119	0x77	VK_F8
F9	120	0x78	VK_F9
F10	121	0x79	VK_F10
пробел	32	0x20	VK_SPACE
BackSpace	8	0x8	VK_BACK
Tab	9	0x9	VK_TAB
Enter	13	0x0D	VK_RETURN
Shift	16	0x10	VK_SHIFT
Ctrl	17	0x11	VK_CONTROL
Alt	18	0x12	VK_MENU

Таблица 22.2 *(продолжение)*

Клавиша	Цифровое значение кода		Символьное имя
	десятичное	шестнадцатеричное	
На основном поле клавиатуры			
CapsLock	20	0x14	VK_CAPITAL
Esc	27	0x1B	VK_ESCAPE
Insert	45	0x2D	VK_INSERT
PageUp	33	0x21	VK_PRIOR
PageDown	34	0x22	VK_NEXT
End	35	0x23	VK_END
Home	36	0x24	VK_HOME
Стрелка ←	37	0x25	VK_LEFT
Стрелка ↑	38	0x26	VK_UP
Стрелка →	39	0x27	VK_RIGHT
Стрелка ↓	40	0x28	VK_DOWN
Delete	46	0x2E	VK_DELETE
PrintScreen	44	0x2C	VK_SNAPSHOT
ScrollLock	145	0x91	VK_SCROLL
0,)	48	0x30	—
1 !	49	0x31	—
2 @	50	0x32	—
3 #	51	0x33	—
4 $	52	0x34	—
5 %	53	0x35	—
6 ^	54	0x36	—
7 &	55	0x37	—
8 *	56	0x38	—
9 (	57	0x39	—
‘ ~	192	0xC0	—
- _	189	0xBD	—
= +	187	0xBB	—
[{	219	0xDB	—

Таблица 22.2 *(продолжение)*

Клавиша	Цифровое значение кода		Символьное имя
	десятичное	шестнадцатеричное	
На основном поле клавиатуры			
] }	221	0xDD	—
; :	186	0xBA	—
' "	222	0xDE	—
\ |	220	0xDC	—
, <	188	0xBC	—
. >	190	0xBE	—
/ ?	191	0xBF	—
a A	65	0x41	—
b B	66	0x42	—
c C	67	0x43	—
d D	68	0x44	—
e E	69	0x45	—
f F	70	0x46	—
g G	71	0x47	—
h H	72	0x48	—
i I	73	0x49	—
j J	74	0x4A	—
k K	75	0x4B	—
l L	76	0x4C	—
m M	77	0x4D	—
n N	78	0x4E	—
o O	79	0x4F	—
p P	80	0x50	—
q Q	81	0x51	—
r R	82	0x52	—
s S	83	0x53	—
t T	84	0x54	—
u U	85	0x55	—

Таблица 22.2 *(окончание)*

Клавиша	Цифровое значение кода		Символьное имя
	десятичное	шестнадцатеричное	
На основном поле клавиатуры			
v V	86	0x56	—
w W	87	0x57	—
x X	88	0x58	—
y Y	89	0x59	—
z Z	90	0x5A	—
Win(Л)	91	0x5B	VK_LWIN
Win(Пр)	92	0x5C	VK_RWIN
На дополнительном (правом цифровом) поле клавиатуры, при выключенном индикаторе NumLock			
0	96	0x60	VK_NUMPAD0
1	97	0x61	VK_NUMPAD1
2	98	0x62	VK_NUMPAD2
3	99	0x63	VK_NUMPAD3
4	100	0x64	VK_NUMPAD4
5	101	0x65	VK_NUMPAD5
6	102	0x66	VK_NUMPAD6
7	103	0x67	VK_NUMPAD7
8	104	0x68	VK_NUMPAD8
9	105	0x69	VK_NUMPAD9
*	106	0x6A	VK_MULTIPLY
+	107	0x6B	VK_ADD
-	108	0x6C	VK_SUBTRACT
.	109	0x6D	VK_DECIMAL
/	110	0x6E	VK_DIVIDE

Параметр **Shift** является множеством, которое может быть пустым или включать символьные значения, описываемые в табл. 22.3.

Таблица 22.3. *Символьные значения* ***Shift***

Символьное значение	Назначение
ssShift	Нажата или отпущена клавиша <Shift>
ssAlt	Нажата или отпущена клавиша <Alt>
ssCtrl	Нажата или отпущена клавиша <Ctrl>

Событие *OnKeyPress*

Другое событие — **OnKeyPress** наступает, если компонент находится в фокусе при нажатии пользователем клавиши символа. Параметр **Key** в обработчике этого события имеет тип char и соответствует символу нажатой клавиши. Функциональные клавиши и специальные клавиши типа <Shift>, <Alt> и <Ctrl> не вызывают этого события. Поэтому нажатие таких комбинаций клавиш, как, например, <Shift>+<A>, генерирует только одно событие **OnKeyPress**, при котором параметр **Key** равен **A**. Поскольку параметр **Key** передается в обработчик события как переменная, его можно изменять, передавая для дальнейшей обработки другой символ. В некоторых программах этот прием использовать полезно.

Для того чтобы распознавать комбинации клавиш или клавиши, не соответствующие символам, необходимо использовать обработчики событий **OnKeyDown** и **OnKeyUp**.

Следует отметить, что событие **OnKeyPress** заведомо наступает, если нажимается только клавиша символа или клавиша символа при нажатой клавише <Shift>. Если же клавиша символа нажимается одновременно с какой-то из вспомогательных клавиш, то событие **OnKeyPress** может не наступить (произойдут только события **OnKeyDown** при нажатии и **OnKeyUp** при отпускании), или если и наступит, то укажет на неверный символ. Например, при нажатой клавише <Alt> событие **OnKeyPress** при нажатии символьной клавиши не наступает. А при нажатой клавише <Ctrl> событие **OnKeyPress** при нажатии символьной клавиши наступает, но символ не распознается.

Событие *OnKeyUp*

Событие **OnKeyUp** наступает, если компонент находится в фокусе при отпускании пользователем любой ранее нажатой клавиши, включая функциональные и вспомогательные, такие как <Shift>, <Alt> и <Ctrl>. В обработчик события передается параметр **Sender**, указывающий на компонент, в котором произошло событие. Кроме этого, в обработчик события передаются параметры **WORD &Key** и **TShiftState Shift**, предназначенные для возврата через них кодов клавиш.

Событие **OnKeyUp** наиболее удобно для распознавания нажатых клавиш и комбинации клавиш. Необходимо обратить внимание, что параметр **Key** для обработчика данного события имеет тип word в отличие от типа char для события **OnKeyPress**. Поэтому параметр **Key** допускается сравнивать с целым числом, соответствующим коду символа, или применить к параметру **Key** функцию char, переводящую его в символ для сравнения с символами. К тому же надо учитывать, что виртуальный символьный код одинаков для символов в верхнем и нижнем регистре и для различной раскладки клавиатуры (русской или английской).

Рассмотрим способ определения кода нажатой клавиши на примере небольшой программы. Создайте в среде разработки Borland C++ Builder 6 новое приложение и поместите на форму один компонент **Label1**. В инспекторе кодов, для формы **Form1** на вкладке событий щелкните дважды по полю правее **OnClick** и введите в заготовку обработчика события программный код из листинга 22.2.

Листинг 22.2. Обработчик события OnClick

```
Label1->Caption = "Было событие OnClick";
```

Этот код программы позволит изменять свойство **Caption** компонента на сообщение о возникновении события **OnClick** на форме приложения.

Аналогично введите для формы **Form1** код обработки события **KeyDown** из листинга 22.3 и код обработки события **KeyPress** из листинга 22.4.

Листинг 22.3. Обработчик события KeyDown

```
if(Shift.Contains(ssAlt)) Label1->Caption = "Alt";
if(Shift.Contains(ssCtrl)) Label1->Caption = "Ctrl";
if(Shift.Contains(ssShift)) Label1->Caption = "Shift";
switch (Key)
{
case VK_INSERT: Label1->Caption = "Insert"; break;
case VK_CAPITAL: Label1->Caption = "CapsLock"; break;
case VK_NUMLOCK: Label1->Caption = "NumLock"; break;
}
```

Листинг 22.4. Обработчик события KeyPress

```
switch (Key)
{
case ' ': Label1->Caption = "Был нажат пробел"; break;
default : Label1->Caption = Key; break;
}
```

Теперь запустите приложение на выполнение и понажимайте любые клавиши на клавиатуре, а также пощелкайте левой кнопкой мыши по форме приложения — и вы увидите реакцию программы на ваши действия.

Событие *OnEnter*

Событие **OnEnter** наступает в момент получения элементом фокуса. Это событие не наступает при переключениях между формами или между приложениями. При переключениях между элементами, расположенными в разных контейнерах, например, на разных панелях, событие **OnEnter** сначала наступает для контейнера, а потом — для содержащегося в нем элемента.

Событие *OnExit*

Событие **OnExit** наступает в момент потери элементом фокуса, в момент его переключения на другой элемент. Это событие не наступает при переключениях между формами или между приложениями. Значение свойства **ActiveControl** изменяется прежде, чем происходит событие **OnExit**. При переключениях между элементами, расположенными в разных контейнерах, например, на разных панелях, событие **OnExit** сначала наступает для элемента, а потом для содержащего его контейнера. Это противоположно последовательности событий **OnEnter**, которые при переключении из другого контейнера на компонент данного наступают сначала для контейнера, а потом для компонента.

Создайте новое приложение и поместите на него три компонента **Edit** из вкладки **Standard**. Создайте обработчик события **OnEnter** для компонента **Edit2** и вставьте в него программный код:

```
ShowMessage("Фокус на элементе");
```

Теперь создайте обработчик события **OnExit** для этого же компонента и вставьте в него программный код:

```
ShowMessage("Фокус переключен");
```

Запустите приложение и нажимайте клавишу табуляции. Фокус будет переключаться между компонентами **Edit**, и при переходе через **Edit2** на экране будет формироваться сообщение о состоянии фокуса.

Далее рассмотрим события, связанные с работой мыши.

События *OnMouseDown* и *OnMouseUp*

Событие **OnMouseDown** наступает в момент нажатия пользователем кнопки мыши над компонентом. Имеется также парное к нему событие **OnMouseUp**, наступающее при отпускании нажатой кнопки мыши над объектом.

Обработка событий **OnMouseDown** и **OnMouseUp** используется для операций, требуемых при нажатии и отпускании пользователем какой-либо кнопки мыши. Обработчики этих событий имеют параметры **Sender**, **Shift**, **Button**, **X** и **Y**. С первыми двумя параметрами мы уже знакомы из описания предыдущих событий. Значения параметра **Button** определяют, какая кнопка мыши нажата: **mbLeft** — левая, **mbRight** — правая, **mbMiddle** — средняя. Параметры **X** и **Y** определяют координаты указателя мыши в клиентской области компонента.

Создайте новое приложение и поместите на его форму два компонента **Label**. Для формы **Form1** создайте обработчик события **OnMouseDown** и введите в него следующие строки программного кода из листинга 22.5.

Листинг 22.5. Обработчик события OnMouseDown

```
if(Button == mbLeft) Label1->Caption = "Нажата левая кнопка мыши";
if(Button == mbRight) Label1->Caption = "Нажата правая кнопка мыши";
if(Button == mbMiddle) Label1->Caption = "Нажата средняя кнопка мыши";
Label2->Caption = "Координаты мыши: X="+String(X)+" Y="+String(Y);
```

Запустите приложение и пощелкайте разными кнопками мыши на его форме. Вы увидите сообщения программы о ваших действиях и координаты курсора мыши.

Событие *OnMouseMove*

Событие **OnMouseMove** наступает при перемещении курсора мыши над компонентом. Обработчик события **OnMouseMove** вставляется в программу, если необходимо произвести какие-то операции при перемещении курсора мыши над компонентом. Параметр **Shift**, являющийся множеством, содержит элементы, позволяющие определить, какие кнопки мыши и какие вспомогательные клавиши (<Shift>, <Ctrl> или <Alt>) нажаты в этот момент. Параметры **X** и **Y** определяют координаты указателя мыши в клиентской области компонента.

Добавьте в предыдущую программу обработчик события **OnMouseMove** для формы **Form1** и вставьте в него следующую строку программного кода из листинга 22.6:

Листинг 22.6. Обработчик события OnMouseMove

```
Label2->Caption = "Координаты мыши: X="+String(X)+" Y="+String(Y);
```

Вновь запустите приложение и проверьте, как ведет себя программа, когда вы перемещаете курсор мыши над формой и другими компонентами.

Следующие описываемые события относятся к графическим компонентам и перемещению объектов.

Событие *OnChange*

Событие **OnChange** связано с графическим компонентом канвы (рамки) и наступает после изменения графического объекта. Обработчик события **OnChange** должен осуществить необходимые операции при изменении графического объекта и отразить его новые установки. Событие происходит сразу после изменения изображения на канве. При вызове любого метода рисования осуществляется следующая последовательность операций:

1. Происходит событие **OnChanging**.
2. Вызванный метод канвы **TCanvas** осуществляет изменения в изображении.
3. Наступает событие **OnChange**.

Событие канвы **OnChange** наступает при изменении именно самого изображения, а не свойств канвы. Такие свойства канвы, как объекты **Font** (Шрифт), **Brush** (Кисть) и **Pen** (Перо), имеют свои собственные события **OnChange**.

Событие *OnPaint*

Событие **OnPaint** наступает, когда возникает сообщение Windows о необходимости перерисовать испорченное изображение. Изображение может испортиться из-за временного перекрытия данного окна другим окном того же или постороннего приложения. Обработчик данного события должен перерисовать изображение. При перерисовке изображения канвы **Canvas** можно использовать свойство **ClipRect**, которое указывает область канвы, внутри которой изображение испорчено.

Событие *OnProgress*

События **OnProgress** наступают во время медленных процессов изменения графического изображения, таких как загрузка, сохранение, трансформация. Заготовка его обработчика приведена в листинге 22.7.

Листинг 22.7. Обработчик события OnProgress

```
void __fastcall TForm1::Image1Progress(TObject *Sender, TProgressStage
Stage, BYTE PercentDone, bool RedrawNow, const TRect &R, const AnsiString Msg)
{
}
```

Эти события позволяют построить в приложении индикатор хода процесса, обеспечивающий обратную связь с пользователем. Разработчики новых компонентов могут генерировать события **OnProgress**, вызывая защищенный метод **Progress**. Параметр **Stage** указывает стадию процесса: начало, продолжение, окончание. Он может соответственно принимать значения **psStarting**, **psRunning** и **psEnding**. Если приложение предусматривает индикацию процесса, то можно создавать индикатор при **Stage** = **psStarting**, изменять его показания, пока **Stage** = **psRunning** и закрывать при **Stage** = **psEnding**. Параметр **PercentDone** показывает, какая примерно часть процесса выполнена. Этот параметр может использоваться в индикаторе процесса. Параметр **RedrawNow** указывает на то, возможно ли в данный момент успешно отобразить изображение на экране. Параметр **R** указывает область изображения, которая изменена и нуждается в перерисовке. Параметр **Msg** содержит краткую справку о протекающем процессе. Например, **Loading** (Загрузка) или **Storing** (Запоминание) или **Reducing colours** (Уменьшение цветов). Строка **Msg** может быть и пустой.

Событие *OnDragDrop*

Событие **OnDragDrop** наступает в момент отпускания над компонентом другого перетаскиваемого компонента. В обработчике данного события необходимо описать, что в этот момент должно произойти. Параметр **Source** соответствует перетаскиваемому объекту, а параметр **Sender** — объекту, над которым другой объект был отпущен. Параметры **X** и **Y** содержат координаты позиции курсора мыши над компонентом в системе координат клиентской области этого компонента.

Событие *OnDragOver*

Событие **OnDragOver** происходит в момент, когда перетаскиваемый объект пересек границу данного компонента и оказался внутри его контура. Заканчивается событие, когда объект покидает компонент, пересекая его границу. Обработчик события **OnDragOver** используется для того, чтобы дать сигнал о готовности компонента принять перетаскиваемый объект в том случае, если пользователь отпустит его над данным компонентом. Если компонент готов принять объект, то в обработчике надо задать значение параметра **Accept**, равное **true**.

Впрочем, это значение по умолчанию равно **true**, так что его можно не задавать. Обработчик может быть и пустым, что будет означать готовность компонента принять любой объект. Обработчик нужен в любом случае (даже пустой), иначе приложение не получит сообщения о приеме компонента.

Во время перетаскивания над компонентом объекта форма курсора мыши может изменяться, сигнализируя пользователю о готовности компонента принять объект. Для обеспечения этого необходимо во время проектиро-

вания задать соответствующее значение свойства компонента **DragCursor**. Параметр **Source** обработчика события определяет перетаскиваемый объект, а параметр **Sender** — сам компонент. Параметры **X** и **Y** дают координаты точки экрана в пикселах. Параметр **State** типа **TDragState** определяет состояние перетаскиваемого объекта по отношению к другим объектам. Значения этого параметра приведены в табл. 22.4.

Таблица 22.4. *Значения параметра* ***State***

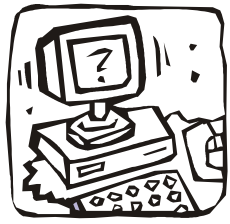
Значение	Описание
dsDragEnter	Курсор мыши входит в пределы компонента
dsDragMove	Курсор мыши перемещается в пределах компонента
dsDragLeave	Курсор мыши выходит за пределы компонента

Событие *OnStartDrag*

Событие **OnStartDrag** наступает, когда пользователь нажал левую кнопку мыши над объектом и, не отпуская ее, начал смещать курсор мыши, то есть начал перетаскивание. Обработчик события **OnStartDrag** позволяет описать какие-либо специальные действия, необходимые перед началом перетаскивания. Параметр **Sender** является тем компонентом, который должен перетаскиваться или который содержит объект будущего перетаскивания. Переменная **DragObject** по умолчанию является пустой, то есть имеет значение **NULL**. Это означает, что переноситься будет сам компонент или его объект. При этом автоматически создается объект типа **TDragControlObject**, с которым среда разработки Borland C++ Builder осуществляет весь процесс перетаскивания. Но обработчик может создавать и новый объект перетаскивания, в котором определен какой-нибудь особый вид курсора и т. п.

Событие *OnEndDrag*

Событие **OnEndDrag** наступает при любом окончании процесса перетаскивания компонента, которое может быть успешным (компонент перетащен в приемник) или безуспешным (компонент отпущен над формой или компонентом, не способным его принять). Данное событие наступает в момент прерывания или окончания перетаскивания компонента. Событие наступает в перетаскиваемом компоненте. Обработка этого события не требуется для осуществления процесса перетаскивания. Соответствующий обработчик может быть написан, если требуется какое-либо действие или сообщение, подтверждающее результат перетаскивания, или какая-то реакция в перетаскиваемом компоненте. Параметр **Sender** обработчика сам является объектом перетаскивания. Параметр **Target** является компонентом-приемником, если объект был им принят, или пустым, если перетаскивание закончилось неудачей. Параметры **X** и **Y** дают координаты экрана в пикселах.



Глава 23

Методы компонентов

В этой главе рассказывается о назначении и применении на практике некоторых методов визуальных компонентов.

Наряду с обработчиками событий среда разработки Borland C++ Builder 6 позволяет использовать при написании программы методы компонентов. Мы уже применяли в рассмотренных программах некоторые из них, например, `LoadFromFile`, `SaveToFile` и др. В отличие от обработчиков событий, методы не вызываются в результате каких-либо действий пользователя, а выполняются при достижении программой строки, в которой они записаны.

Методы вставляются в программу самим разработчиком по мере необходимости, например, для отображения рисунка на экране монитора, работы с файлами и т. д. По сути дела методы являются функциями или подпрограммами. Так же, как свойства и события, одни и те же методы могут быть применимы для разных типов компонентов. В табл. 23.1 приведено краткое описание большинства существующих методов.

Таблица 23.1. *Краткое описание методов компонентов*

Метод	Описание
`Add`	Добавляет новый элемент в список
`Assign`	Копирует изображение одного графического объекта в другой
`BeginDrag`	Используется при перетаскивании компонента
`BringToFront`	Позволяет изменять последовательность перекрытия компонентов
`BrushCopy`	Копирует часть изображения битовой матрицы на канву
`CanFocus`	Определяет, может ли компонент получать сообщения пользователя

Таблица 23.1 *(продолжение)*

Метод	Описание
`ChangeScale`	Используется для изменения масштаба компонента
`Chord`	Рисует заполненную замкнутую фигуру, ограниченную дугой окружности или эллипса и хордой
`ClassName`	Возвращает имя типа объекта
`Clear`	Очищает список или текст
`ClientToScreen`	Преобразует координаты клиентской области в координаты экрана
`ContainsControl`	Определяет, является ли указанный компонент прямым или косвенным наследником данного оконного компонента
`ControlAtPos`	Определяет, какой дочерний компонент имеется в указанной позиции
`CopyRect`	Копирует часть изображения
`Delete`	Удаляет элемент с указанным индексом из списка
`DisableAlign`	Временно запрещает выравнивание дочерних компонентов
`Dormant`	Создает изображение битовой матрицы в памяти
`Draw`	Рисует графическое изображение в указанную позицию канвы
`DrawFocusRect`	Рисует изображение прямоугольника в виде, используемом для отображения рамки фокуса, с помощью операции XOR
`DisableAlign`	Временно запрещает выравнивание дочерних компонентов
`Ellipse`	Рисует заполненный эллипс или окружность
`Expand`	Увеличивает емкость списка типа **TList**
`FillRect`	Заполняет указанный прямоугольник канвы, используя текущее значение **Brush**
`FindNextControl`	Возвращает следующий в последовательности табуляции оконный дочерний компонент
`FloodFill`	Закрашивает текущей кистью замкнутую область канвы, ограниченную определенным цветом
`Focused`	Определяет, находится ли оконный элемент в фокусе
`FrameRect`	Рисует на канве текущей кистью прямоугольную рамку

Таблица 23.1 *(продолжение)*

Метод	Описание
`Free`	Освобождает память, занимаемую объектом
`GetTabOrderList`	Строит список дочерних оконных компонентов в последовательности табуляции
`HandleAllocated`	Проверяет наличие дескриптора окна компонента
`HandleNeeded`	Создает дескриптор окна, если до этого он не существовал
`Hide`	Делает компонент невидимым
`IndexOf`	Определение первого вхождения в список заданного элемента
`Insert`	Вставляет элемент в заданную позицию списка
`Invalidate`	Сообщает Windows о необходимости полностью перерисовать компонент
`LineTo`	Рисует на канве прямую линию
`LoadFromClipboardFormat`	Загружает изображение из буфера обмена в формате **Clipboard**
`LoadFromFile`	Загружает изображение, хранящееся в файле
`LoadFromResourceID`	Загружает битовую карту из файла ресурсов по указанному идентификатору
`LoadFromResourceName`	Загружает битовую карту из файла ресурсов по указанному имени
`LoadFromStream`	Загружает графическое изображение из указанного потока
`Lock`	Блокирует канву, запрещая рисовать на ней
`Move`	Меняет текущую позицию элемента в списке на позицию, заданную в параметрах метода
`MoveTo`	Устанавливает позицию пера на позицию, заданную в параметрах метода
`Pie`	Рисует заполненную замкнутую фигуру в виде сегмента окружности или эллипса
`Polygon`	Рисует на канве заполненную замкнутую фигуру в виде многоугольника
`Polyline`	Рисует на канве ломаную линию
`DisableAlign`	Временно запрещает выравнивание дочерних компонентов
`Rectangle`	Рисует на канве закрашенный прямоугольник

Таблица 23.1 *(продолжение)*

Метод	Описание
`Refresh`	Перерисовывает изображение компонента на экране
`Remove`	Удаляет элемент с заданным значением из списка **TList**
`Repaint`	Перерисовывает изображение компонента на экране
`ReplaceDockedControl`	Встраивает компонент на место другого компонента
`RoundRect`	Рисует на канве прямоугольную рамку со скругленными углами
`SaveToClipboardFormat`	Создает копию изображения в формате **Clipboard**
`SaveToFile`	Сохраняет графическое изображение в файле
`SaveToStream`	Сохраняет графическое изображение в потоке
`ScaleBy`	Масштабирует оконный элемент и все содержащиеся в нем компоненты
`ScaleControls`	Масштабирует дочерние компоненты оконного элемента, не изменяя масштаба самого элемента
`ScreenToClient`	Преобразует координаты экрана в координаты клиентской области компонента
`ScrollBy`	Сдвигает содержимое оконного элемента
`SelectFirst`	Передает фокус дочернему компоненту, первому в последовательности табуляции
`SelectNext`	Передает фокус дочернему компоненту, следующему в последовательности табуляции за указанным компонентом
`SendCancelMode`	Прерывает модальное состояние элемента управления
`SendToBack`	Переносит компонент ниже других компонентов
`SetBounds`	Устанавливает одновременно свойства **Left**, **Top**, **Width** и **Height**
`SetChildOrder`	Изменяет позицию компонента в списке дочерних компонентов оконного элемента
`SetFocus`	Передает фокус элементу
`SetZOrder`	Перемещает компонент на вершину или вниз Z-последовательности
`Show`	Делает видимым невидимый компонент

Таблица 23.1 *(окончание)*

Метод	Описание
`StretchDraw`	Рисует графическое изображение в указанную прямоугольную область канвы, подгоняя размер изображения под заданную область
`TextExtent`	Возвращает длину и высоту в пикселах текста, который предполагается написать на канве текущим шрифтом
`TextHeight`	Возвращает высоту в пикселах текста, который предполагается написать на канве текущим шрифтом
`TextOut`	Пишет указанную строку текста на канве, начиная с указанной позиции
`TextRect`	Пишет указанную строку текста на канве, начиная с указанной позиции и усекая текст, выходящий за пределы указанной прямоугольной области
`TextWidth`	Возвращает в пикселах длину текста, который предполагается написать на канве текущим шрифтом
`TryLock`	Блокирует канву, если она не была блокирована, не разрешая другим нитям многопоточного приложения рисовать на ней
`Unlock`	Уменьшает на единицу значение свойства **LockCount**, способствуя тем самым разблокировке канвы, когда **LockCount** станет равным нулю
`Update`	Перерисовывает компонент

Рассмотрим более подробно назначение и описание некоторых наиболее часто используемых методов с примерами их применения.

Начнем с создания небольшой программы, которая наглядно покажет нам, как применяются методы `Add` (Добавить), `Hide` (Скрыть), `Show` (Показать) и `Delete` (Удалить). Создайте новое приложение и поместите на его форму компонент **ListBox** и четыре кнопки типа **Button**. Измените размеры формы и разместите на ней компоненты так, как показано на рис. 23.1.

Задайте свойство названия — **Caption** для формы и кнопок в соответствии с этим же рисунком. Щелкните дважды левой кнопкой мыши по компоненту первой кнопки **Button1** и впишите для нее код обработки события **OnClick** в виде программной строки (листинг 23.1).

Листинг 23.1. Код обработки события OnClick для кнопки Button1

```
ListBox1->Items->Add(String(i++));
```

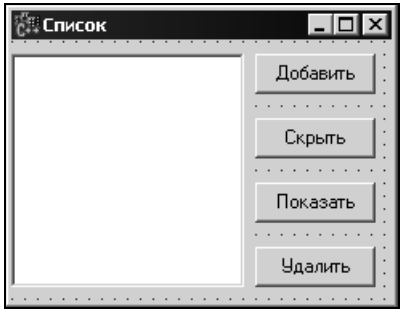


Рис. 23.1. Окно формы программы

Эта строка содержит метод `Add`, который позволит добавить в компонент **ListBox1** элемент списка, представляющий собой строку, формирующуюся из переменной `i` с помощью функции `String`. Каждый раз при вызове данной программной строки с методом `Add` переменная `i` будет увеличивать свое значение благодаря тому, что после нее вписан оператор автоинкремента `++` (увеличение значения на единицу). Описание переменной с присвоением ей начального значения в виде строки `int i=0;` необходимо вписать в программу в инспекторе объектов сразу после строк с описанием включений `#include`. В этом случае переменная будет доступна любому обработчику программы. Если описать эту переменную в самом обработчике события, то она станет локальной (недоступной другим обработчикам) и будет инициализироваться каждый раз при формировании этого события.

Аналогично сформируйте заготовку кода обработки события **OnClick** для компонента кнопки **Button2** и впишите в него программную строку (листинг 23.2).

Листинг 23.2. Код обработки события OnClick для кнопки Button2

```
ListBox1->Hide();
```

Эта строка содержит метод `Hide`, который скрывает изображение компонента на форме от пользователя программы.

Подобным образом впишите в код обработки события **OnClick** для компонента кнопки **Button3** программную строку (листинг 23.3).

Листинг 23.3. Код обработки события OnClick для кнопки Button3

```
ListBox1->Show();
```

Эта строка содержит метод `Show`, который показывает изображение компонента на форме для пользователя программы.

Наконец впишите код обработки события **OnClick** для компонента кнопки **Button4** в виде программной строки (листинг 23.4).

Листинг 23.4. Код обработки события OnClick для кнопки Button4

```
if(i>0) ListBox1->Items->Delete(--i);
```

Эта строка содержит метод `Delete`, который удаляет указанный элемент списка из компонента **ListBox1**.

Обратите внимание, что в качестве номера удаляемого элемента списка мы использовали ту же переменную `i`, которая применялась при добавлении элементов списка в компонент. При этом перед ее использованием она автоматически декрементируется (уменьшает свое значение на единицу) благодаря применению оператора "– –".

Метод удаления имеет смысл выполнять только в том случае, если элементы в списке есть, то есть имеется, что удалять. Для проверки данного условия в программную строку включен оператор сравнения переменной индекса элементов списка с нулем: `if(i>0)`, который позволяет выполнять удаление элемента из списка только при натуральном значении переменной `i`, то есть когда элементы в списке есть.

Сохраните созданный проект и запустите приложение на выполнение. После чего проверьте, как оно работает. Если вам понадобится помощь, найдите этот проект в каталоге \PROJECT\SPISOK на диске, прилагаемом к книге, и сравните его с тем, что у вас получилось.

Теперь можно рассмотреть некоторые детали методов, которые часто используются в программах.

Метод *Add*

Метод `Add` применяется к компонентам, имеющим списки или строки. Метод возвращает при выполнении индекс добавленного элемента. Например, программная строка `i=ListBox1->Items->Add (Новый элемент списка);` присвоит переменной `i` значение возвращаемого параметра. Индекс первого элемента списка всегда равен нулю. Данный метод увеличивает значение свойства **Count** (Номер элемента) на единицу. Если значение **Count** равно значению свойства **Capacity** (Размер списка), то увеличивается значение **Capacity** и перераспределяется память под новые элементы. Напомню, что доступ к свойствам **Count** и **Capacity** осуществляется с помощью программной строки `ListBox1->Item->Count` и `ListBox1->Item->Capacity` соответственно. Используя данные свойства, можно исключить в предыдущей программе переменную `i`, сохранив при этом все ее функции. Предлагаю читателям проделать это самостоятельно.

Метод *Hide*

Метод `Hide` делает компонент невидимым, задавая свойству **Visible** компонента значение **false**. Если компонент является контейнером для других компонентов, то эти дочерние компоненты также делаются невидимыми. Несмотря на то, что компонент становится невидимым, его свойства и методы остаются доступными в программе.

Метод *Show*

Метод `Show` делает видимым ранее невидимый компонент. Он задает значение **true** свойству **Visible** компонента и проверяет, является ли видимым родительский компонент.

Метод *Delete*

Метод `Delete` удаляет из списка элемент с указанным индексом.

Метод *CanFocus*

Метод `CanFocus` определяет, может ли компонент получать сообщения пользователя, то есть может ли он получать фокус. Функция возвращает **true**, если у компонента и всех его родителей свойства **Visible** и **Enabled** установлены в **true**. В противном случае возвращается значение **false**.

Метод *ChangeScale*

Метод `ChangeScale(M,D)` используется для изменения масштаба компонентов. При его применении масштабируются такие свойства компонента, как **Top** и **Left**, определяющие его местоположение, а также **Width** и **Height**, определяющие его размер. Параметры метода `M` и `D` определяют соотношение новых размеров компонентов к их прежним размерам. Например, чтобы уменьшить размеры компонента до 50% начального значения, нужно задать `M = 50`, а `D = 100`. То же самое можно сделать, задав `M = 1` и `D = 2`. Если вы хотите увеличить размер, то нужно задать значение `M` больше, чем значение `D`. Так, например, программная строка `ChangeScale(2,1);` увеличит размер компонента в 2 раза.

Следующая группа описываемых методов связана с графическими компонентами и предназначена для рисования. Рассмотрим их применение на примере программы.

Создайте новое приложение и поместите на его форму компонент **Image** и одну кнопку **Button**. Установите размеры компонента **Image** равными 200 единиц по высоте и 300 единиц по ширине, с помощью свойства **Height** и **Width**, соответственно. Свойство **Caption** формы **Form1** заполните названием **Draw**. Задайте свойство **Caption** компонента **Button1**, назвав кнопку именем **Рисовать**, и расположите компоненты на форме в соответствии с рис. 23.2.

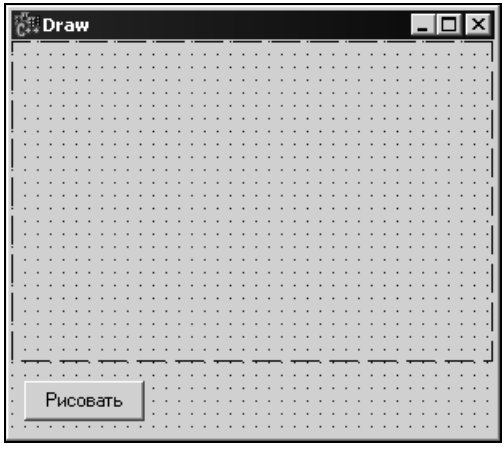


Рис. 23.2. Окно формы программы рисования

Объявите целую переменную `i`, с помощью оператора `int i;` после строки `public` в файле описаний модуля с расширением h, и задайте ее начальное значение с помощью оператора `i=1;` в обработчике создания формы **Form1**.

Сформируйте для кнопки заготовку обработчика события **OnClick** и вставьте в него следующие программные строки из листинга 23.5.

Листинг 23.5. Код обработки события OnClick для кнопки Form1

```
Image1->Canvas->Pen->Color=clBlack;
Image1->Canvas->Brush->Color=clBlue;
switch(i){
 case 1: Image1->Canvas->Ellipse(0,0,50,50); break;
 case 2: Image1->Canvas->Chord(50,0,100,50,75,0,100,25); break;
 case 3: Image1->Canvas->Pie(100,0,150,50,125,0,150,25); break;
 case 4: Image1->Canvas->Arc(150,0,200,50,150,0,200,50); break;
 case 5: Image1->Canvas->Rectangle(200,0,250,50); break;
 case 6: Image1->Canvas->RoundRect(250,0,300,50,20,20); break;
 case 7: Image1->Canvas->TextOutA(0,100,"Графический текст"); break;
 case 8: Image1->Canvas->MoveTo(0,200); Image1->Canvas->LineTo(300,0); break;
 case 9: Image1->Canvas->FloodFill(0,0,clRed,fsBorder);
```

```
 Image1->Canvas->Brush->Color=clWhite;
 Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));
 i=0;
 break;
 default: break;
 }
Button1->Caption="Фигура"+String(++i);
```

Текст этой программы под названием **Draw** записан в каталоге \PROJECT\DRAW на диске, прилагаемом к книге.

Первые две строки данной программы задают свойства **Pen** (Пера) и **Brash** (Кисти) для **Canvas** (Канвы) **Image** (Рисунка). Далее следует оператор `switch`, который сравнивает значение переменной `i` с целыми значениями от 1 до 9. Каждому из этих значений сопоставлена программная строка с применением различных методов графического отображения фигур, заканчивающаяся оператором `break` для прерывания сравнения. Познакомимся подробнее с этими методами программных строк.

Метод *Ellipse*

Метод `Ellipse(X1, Y1, X2, Y2)` рисует эллипс или окружность с помощью текущих параметров **Pen** (Пера). Фигура заполняется текущим значением **Brush** (Кисти). **X1**, **Y1** и **X2**, **Y2** являются координатами верхнего левого и правого нижнего углов прямоугольника, описывающего эллипс. Метод вызывается с помощью строки: `Image1->Canvas->Ellipse(X1, Y1, X2, Y2);`. Операционная система Windows 95 накладывает ограничение на параметры этого и последующих методов. Они связаны с тем, что сумма координат прямоугольника не должна превышать значение 32 767.

Метод *Chord*

Метод `Chord(X1, Y1, X2, Y2, X3, Y3, X4, Y4)` рисует фигуру в виде дуги эллипса или окружности, с помощью текущих параметров **Pen** (Пера), замкнутую хордой. Фигура заполняется текущим значением **Brush** (Кисти). Координаты `X1`, `Y1` и `X2`, `Y2` определяют прямоугольник, описывающий эллипс. Дуга начинается с точки, являющейся пересечением эллипса с прямой, проходящей через его центр и точку с координатами `X3`, `Y3`. Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку с координатами `X4`, `Y4`. Дуга рисуется против часовой стрелки от начальной до конечной точки. Хорда соединяет концы дуги.

Метод *Pie*

Метод `Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4)` (Пирог) рисует замкнутую фигуру в виде сектора эллипса или окружности с помощью текущих параметров **Pen** (Пера). Фигура заполняется текущим значением **Brush** (Кисти). Точки с координатами `X1`, `Y1` и `X2`, `Y2` определяют углы прямоугольника, описывающего эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку с координатами `X3`, `Y3`. Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку с координатами `X4`, `Y4`. Дуга рисуется против часовой стрелки от начальной до конечной точки. Кроме того, рисуются прямые, ограничивающие сегмент и проходящие через центр эллипса и точки с координатами `X3`, `Y3` и `X4`, `Y4`. Как видно из описания, данный метод очень схож с методом `Chord`.

Метод *Arc*

Метод `Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4)` (Дуга) рисует разомкнутую фигуру в виде дуги эллипса или окружности с помощью текущих параметров **Pen** (Пера). Фигура не заполняется значением **Brush** (Кисти). Точки с координатами `X1`, `Y1` и `X2`, `Y2` определяют углы прямоугольника, описывающего эллипс. Координаты фигуры определяются аналогично методам `Chord` и `Pie`.

Метод *Rectangle*

Метод `Rectangle(X1, Y1, X2, Y2)` (Прямоугольник) рисует текущим пером на канве прямоугольник, координаты верхнего левого и нижнего правого углов которого имеют координаты `X1`, `Y1` и `X2`, `Y2` соответственно. Прямоугольник закрашивается текущей кистью. Рисование прямоугольника без рамки можно осуществить методом `FillRect`. Прямоугольник со скругленными углами рисуется методом `RoundRect`. Прямоугольник без внутренней закраски рисуется методом `FrameRect`.

Метод *RoundRect*

Метод `RoundRect(X1, Y1, X2, Y2, X3, Y3)`, в отличие от `Rectangle`, имеет дополнительные координаты `X3` и `Y3`, которые определяют ширину и высоту эллипса, определяющего рисование закругленных углов.

Метод *TextOut*

Метод `TextOut (X, Y, "строка")` пишет строку текста на канве, начиная с позиции с координатами `X` и `Y`. Надпись делается в соответствии с текущими

установками **Font** (Шрифта). Фон надписи определяется установками текущей кисти. Текущая позиция пера **PenPos** перемещается к концу выведенного текста. Для выравнивания позиции текста на канве можно использовать методы, возвращающие перед выводом текста его высоту и длину в пикселах. Для этих целей подходят методы `TextExtent`, `TextHeight` и `TextWidth`. Например, программная строка из листинга 23.6 выведет на канву текстовую строку "123", выравнивая ее при любом шрифте посередине канвы и отступив одну строку сверху.

Листинг 23.6. Программа вывода на канву текстовой строки "123"

```
Image1->Canvas->TextOut((ClientWidth-Canvas->TextExtent("123").cx)/2,
Canvas->TextExtent("123").cy,"123");
```

Если цвет кисти в момент вывода текста отличается от того, которым закрашена канва, то текст получится выведенным в цветной прямоугольной рамке. Но размеры рамки будут точно равны размерам надписи. Если требуется более красивая рамка с отступом от текста или если надо ограничить выводимый текст размерами определенной рамки, следует применять метод **TextRect**.

Метод *TextRect*

Метод `TextRect(Rect, X, Y, "строка")` выводит текстовую строку на канве, начиная с позиции левого верхнего угла надписи, с координатами `X`, `Y`. Часть текста, не помещающаяся в прямоугольную область **Rect**, усекается. Надпись делается в соответствии с текущими установками шрифта **Font**. Пространство внутри области **Rect** закрашивается текущей кистью **Brush**. Для выравнивания позиции текста внутри области на канве также можно использовать методы `TextExtent`, `TextHeight` и `TextWidth`, дающие перед выводом высоту и длину текста в пикселах.

Метод *MoveTo*

Метод `MoveTo(X, Y)` изменяет текущую позицию пера **PenPos** на позицию, заданную координатами точки `X`, `Y`. Это действие эквивалентно непосредственной установке свойства **PenPos**. При перемещении пера методом `MoveTo` на экране ничего не отображается.

Метод *LineTo*

Метод `LineTo(X, Y)` рисует на канве прямую линию, начинающуюся с текущей позиции пера **PenPos** и заканчивающуюся точкой с координатами `X`, `Y`. Текущая позиция пера **PenPos** перемещается в точку `X`, `Y`. При рисовании используются текущие установки пера **Pen**.

Метод *FloodFill*

Метод `FloodFill(X, Y, Color, FillStyle)` закрашивает текущей кистью **Brush** замкнутую область канвы, определенную цветом и начальными координатами закрашивания `X`, `Y`. Точка с координатами `X` и `Y` должна находиться внутри заполняемой области, имеющей произвольную форму. Граница этой области определяется сочетанием параметров `Color` и `FillStyle`. Параметр `Color` указывает цвет, который используется для задания границы закрашиваемой области, а параметр `FillStyle` определяет, как именно по этому цвету задается граница. Если `FillStyle = fsSurface`, то заполняется область, окрашенная цветом `Color`, а другие цвета не изменяются. Если `FillStyle = fsBorder`, то наоборот, заполняется область, окрашенная любыми цветами, отличающимися от `Color`, а цвет `Color` не изменяется.

Метод *FillRect*

Метод `FillRect(Rect)` заполняет прямоугольник канвы, указанный параметром `Rect`, используя текущее значение кисти. Заполняемая область включает в себя верхнюю и левую стороны прямоугольника, но не включает правую и нижнюю стороны. При использовании метода `FillRect` параметр `Rect` можно задавать в самом методе с помощью строки: `Rect(X1, Y1, X2, Y2)`, где `X1`, `Y1`, `X2` и `Y2` являются координатами левого верхнего и нижнего правого углов прямоугольника.

Последние два метода используются в приведенной в *разд. "Метод ChangeScale" этой главы* программе **Draw** для очистки канвы. Далее следует оператор обнуления переменной `i` и выход из тела оператора `Switch`. Последняя строка программы `Button1->Caption="Фигура"+String(++i);` позволяет изменять свойство **Caption** для кнопки **Button1** на надпись **ФигураN**, где **N** принимает значение переменной `i` от 1 до 9. Запустите программу и проверьте, как она работает. Каждый раз при нажатии кнопки приложения должна рисоваться новая фигура в соответствии с методом, используемым в программе.

Пример программы *Function*

Помимо простых фигур программно можно отображать на экране монитора сложные кривые, например, в виде графика функций. Рассмотрим очередной пример программы, осуществляющий именно такие операции. Создайте новое приложение и поместите на его форму пять компонентов кнопок **Button** и один компонент **Image**. Задайте свойство **Caption** для кнопок от первой до пятой, присваивающие им соответственно названия: **Рисовать**, **+**, **−**, **Очистить** и **Функция**. Размер пятой кнопки увеличьте по горизонтали

примерно в два раза. Компонент **Image** растяните и разместите на оставшейся части формы. Свойство **Caption** формы заполните названием **Function**. Разместите все компоненты на форме в соответствии с рис. 23.3.

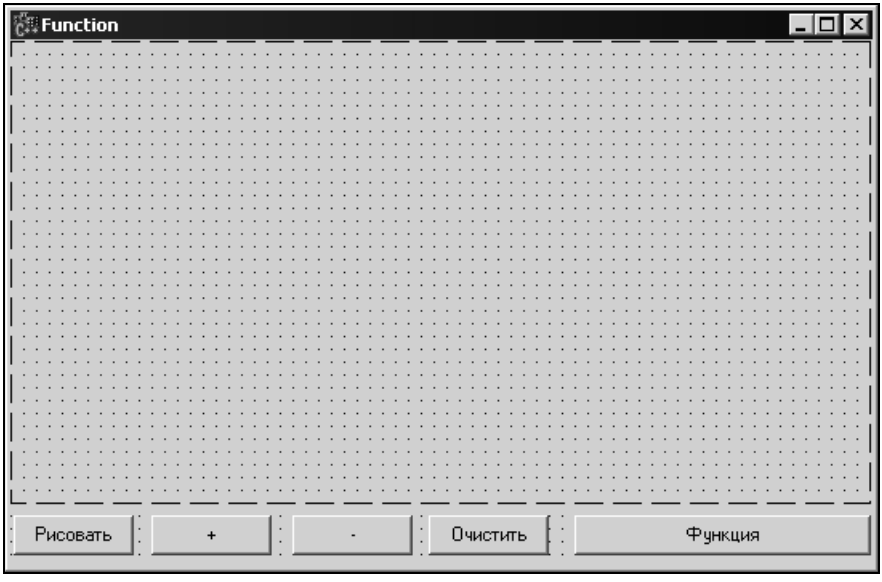


Рис. 23.3. Окно формы программы **Function**

Теперь сохраните модуль проекта в файле с названием Ufunction.cpp, а сам проект в файле с названием Function.bpr. Вставьте в файл описаний Ufunction.h после строки `public: // User declarations` следующие программные строки:

```
int n;          // Номер функции
float m;        // Масштаб
```

описывающие переменные номера функций и масштаба.

После оператора `#include "Ufunction.h"` в файле Ufunction.cpp впишите строку:

```
#include <math.h>,
```

которая предназначена для подключения к проекту библиотеки математических функций, что позволит нам использовать их в программе, в том числе и тригонометрические функции.

Далее впишем в программу обработчики события **OnClick** для всех кнопок. Начнем с четвертой кнопки, отвечающей за очистку экрана. Для нее обработчик события **OnClick** должен выглядеть в соответствии с листингом 23.7.

Листинг 23.7. Обработчик события OnClick

```
void __fastcall TForm1::Button4Click(TObject *Sender)
{
Image1->Canvas->FillRect(Rect(0,0,Image1->Width,Image1->Height));
}
```

Программная строка обработчика содержит знакомый нам метод **FillRect**.

Для обработчика события **OnClick** кнопки рисования введем строки программного кода (листинг 23.8).

Листинг 23.8. Обработчик события OnClick кнопки рисования

```
#define PI 3.14159
float X,Y; // Переменные функции
int PX, PY; // Координаты пикселов
Form1->Button4Click(0);
Image1->Canvas->MoveTo(0,Image1->Height/2);
for(PX=0; PX<=Image1->Width; PX++)
 {
 X=PX*4*PI/Image1->Width;
 switch(n) // Вычисление функций
 {
 case 1: Y=m*sin(X); break;
 case 2: Y=m*X*sin(X*X); break;
 case 3: Y=m*cos(X)*sin(X); break;
 default: break;
 }
 PY=Image1->Height-(Y+1)*Image1->Height/2;
 Image1->Canvas->LineTo(PX,PY);
}
```

Первая из этих строк описывает константу `PI`, принимающую значение числа Пифагора. Далее следует описание переменных `X`, `Y`, `PX` и `PY`, служащих переменными функции и координатами точек рисунка. После этого следует вызов функции обработчика события **OnClik** кнопки очистки экрана, созданной нами ранее и предназначенной для очистки канвы компонента **Image** (то есть рисунка на экране). Далее вписана строка установки пера для рисования в левую среднюю точку экрана. После этой строки в программе присутствует оператор для организации цикла вычисления значения переменной `Y` для множества значений точек `PX` экрана от 0 до максимальной величины ширины экрана в пикселах. Переменная `X` вычисляется в самом цикле. Одновременно масштабируется ее размер по вертикали экрана.

Оператор `switch` служит для сравнения номера выбранной функции и вычисления для нее значения переменной `Y`.

Функции могут быть произвольными. В дальнейшем при необходимости можно будет изменять и расширять список этих функций. Далее в программе вычисляется значение координат `PY` и рисуется фрагмент линии функции.

В обработчик события **OnClick** кнопки **+** (плюс) вставьте строки удвоения коэффициента функции и вызова обработчика перерисовки графика функции, который принадлежит кнопке рисования **Button1**:

```
m=m*2;
Form1->Button1Click(0);
```

В обработчик события **OnClick** кнопки **−** (минус) вставьте строки уменьшения вдвое коэффициента функции и тот же обработчик перерисовки графика функции:

```
m=m/2;
Form1->Button1Click(0);
```

Наконец, для кнопки выбора функции **Button5** впишите обработчик события **OnClick**, состоящий из строк, приведенных в листинге 23.9.

Листинг 23.9. Обработчик события OnClick кнопки выбора функции

```
n++; if(n>3) n=1;
switch(n) // Выбор функций
 {
 case 1: Button5->Caption="Y=m*sin(X)"; break;
 case 2: Button5->Caption="Y=m*X*sin(X*X)"; break;
 case 3: Button5->Caption="Y=m*cos(X)*sin(X)"; break;
 default: break;
 }
```

Здесь будет происходить автоматический инкремент номера функции и присвоение ему начального значения при превышении максимального значения `n>3`. В зависимости от номера функции свойство **Caption** кнопки **Button5** будет менять свое значение на строку с записью самой математической функции.

Для завершения программы необходимо задать начальные значения для переменных масштаба и номера функции. Это делается записью программных строк инициализации переменных `m` и `n` в теле функции формирования формы. Здесь же необходимо вписать вызов обработчика события **OnClick** для кнопки очистки экрана. В конечном виде функция формирования формы должна иметь запись, соответствующую листингу 23.10.

Листинг 23.10. Функция формирования формы

```
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
m=0.05; n=1;
Form1->Button4Click(0);
}
```

Полностью проект программы с названием **Function** находятся в каталоге \PROJECT\FUNCTION на диске, прилагаемом к книге.

Теперь запустите приложение и проверьте его работу. Результат работы программы при выводе на экран функции `Y=m*X*sin(X*X)` представлен на рис. 23.4.

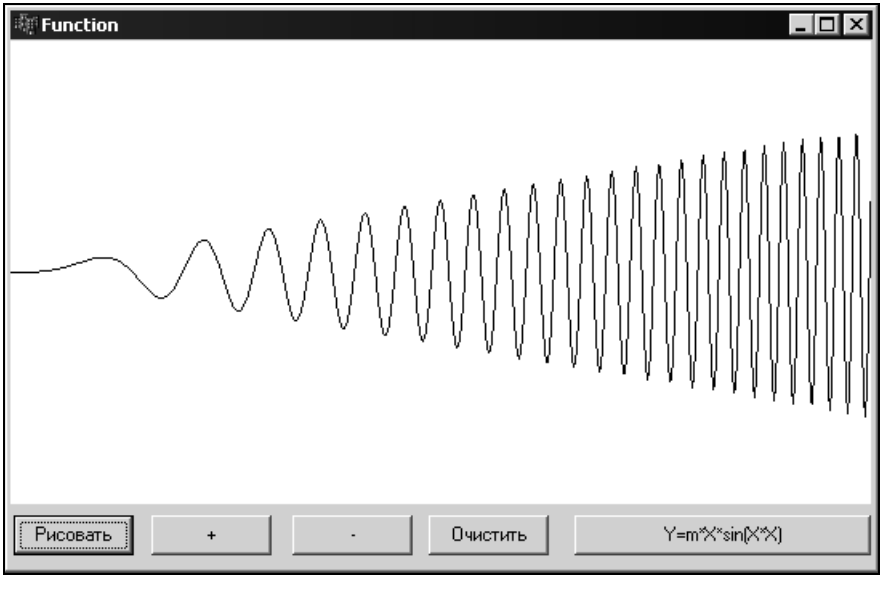


Рис. 23.4. Результат работы программы

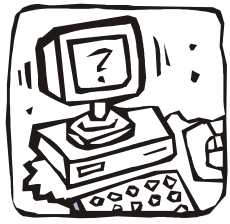
В этой программе мы использовали не только методы, но и некоторые функции, например `sin(X)` и `cos(X)`. Среда разработки Borland C++ Builder 6 имеет огромное количество встроенных библиотек с готовыми к применению функциями, что позволяет писать мощные и элегантные программы. Для знакомства с функциями можно прибегнуть к встроенной справочной системе среды разработки, вызвав ее нажатием клавиши <F1> или <Ctrl>+<F1>, когда курсор редактора находится на самой функции. Аналогично можно поступать при знакомстве с другими методами, не рассмотренными здесь.

Кроме того, среда разработки имеет встроенный суфлер кода, который автоматически выводит на экран список доступных методов, свойств и функций объектов при вводе символа "->", "." или "(" после записи имени объекта.

Полезной является комбинация клавиш <Ctrl>+<Space>, вызывающая список доступных функций и методов для объекта, на котором расположен курсор редактора.

Комбинация клавиш <Ctrl>+<J> открывает окно с перечнем функций-заготовок, которые выбираются программистом с помощью стрелок курсора и вставляются в программу после нажатия клавиши <Enter>.

Этими замечательными инструментами среды разработки удобно пользоваться для вызова встроенной справочной системы, ускорения ввода кода программы и исключения ошибок в записи программных строк.



Глава 24

Программа *Square*

Итак, мы рассмотрели все необходимые инструменты, компоненты и способы программирования в среде разработки программ Borland C++ Builder 6. Теперь читатели вооружены достаточными знаниями, для того чтобы начать составление и разработку собственных программ. Но для закрепления изученного материала на практике, в этой главе вниманию читателей предлагается разработать несколько программ, которые помогут обрести уверенность в своих силах начинающим программистам.

Предлагаемая программа предназначена для вычисления корней квадратного уравнения и может быть особенно полезна учащимся школ. На ее примере можно будет создать множество других программ, облегчающих вычисления различных уравнений и функций.

Сформируем интерфейс программы. Создайте новое приложение и поместите на его форму пять компонентов **Label**, три компонента **Edit**, две кнопки **Button** и компонент **MainMenu**.

Задайте для формы **Form1** свойство **Name** с именем **FormMain**, а свойство **Caption** с названием **Square**. Замените свойства **Name** компонентов **Label1**-**Label5** на **Labelformul**, **Labela**, **Labelb**, **Labelc** и **Labelx1Ч2**, соответственно, и задайте для них следующие свойства **Caption**: **Квадратное уравнение a*x^2+b*x+c=0**, **a=**, **b=**, **c=** и **Ч1=?**, **Ч2=?**.

Замените свойства **Name** компонентов **Button1** и **Button2** на **ButtonExe** и **ButtonOut**, соответственно, и задайте для них следующие свойства **Caption**: **Решить** и **Выход**.

Измените значение свойства **Name** компонентов **Edit1—Edit3** на **Edita**, **Editb** и **Editc**, соответственно, а свойства **Text** этих компонентов заполните значениями **1**, **1** и **0**.

Щелкните дважды левой кнопкой мыши по компоненту **MainMenu1** и в открывшемся окне **FormMain** задайте свойство **Caption** для первого пункта меню в виде названия **О программе**.

Закройте окно **FormMain** и расположите компоненты на форме в соответствии с рис. 24.1.

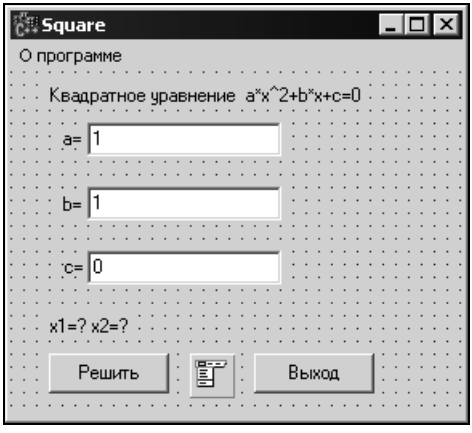


Рис. 24.1. Расположение компонентов на основной форме программы

Сохраните проект в файле с названием Square.bpr а модуль программы — в файле Usquare.cpp.

Интерфейс программы сформирован. Теперь можно приступить к редактированию программного кода. Щелкните дважды левой кнопкой мыши по кнопке **Решить** и введите в обработчик события **OnClick** строки программы из листинга 24.1.

Листинг 24.1. Строки обработчика события OnClick программы Square

```
double a,b,c,x1,x2;
try
 {
 a=Edita->Text.ToDouble();
 b=Editb->Text.ToDouble();
 c=Editc->Text.ToDouble();
 }
catch(EConvertError&)
 {
 ShowMessage("Неправильно введен коэффициент\rПовторите ввод");
 Labelx1x2->Caption="x1=? x2=?"; // Убрать предыдущий результат
 return;
 }
int d=b*b+4*a*c;
if (a==0){
```

```
        if(b==0){
                Label1x1x2->Caption="Неправильно заданы коэффициенты a и b";
                return;
                };
        x1=-c/b;
        Label1x1x2->Caption="Корень один.\rx="+String(x1)+";";
        return;
        };
if (d<0){
        Label1x1x2->Caption="Детерминант меньше нуля. Нет корней!";
        return;
        };
if (d==0){
        x1=(-b/(2*a));
        Label1x1x2->Caption="Детерминант равен нулю. Корень
один.\rx="+String(x1)+";";
        return;
        };
x1=(-b+sqrt(d))/(2*a);
x2=(-b-sqrt(d))/(2*a);
Label1x1x2->Caption="x1="+String(x1)+"; x2="+String(x2)+";";
```

Эти программные строки и весь проект программы под названием **Square** можно скопировать с диска, прилагаемого к книге. В них описываются переменные, которые участвуют в вычислениях корней квадратного уравнения, и включены операторы и функции для вычисления корней. Поскольку при вычислении могут встретиться запрещенные ситуации, наподобие деления на ноль и пр., в программу включены операторы проверки подобных ситуаций с выводом соответствующих сообщений пользователю программы. В программе эти сообщения присутствуют в виде текста, заключенного в кавычки. Если этого не делать, то программа будет выдавать системные предупреждения на английском языке или "зависать". Хорошим стилем программирования считается предупреждение всех ошибочных ситуаций и недопущение тупикового выполнения программы приводящего к ее "зависанию". В противном случае программа будет раздражать пользователя, и он от нее откажется.

Так как в тексте программы присутствуют непростые арифметические операции, например вычисление квадратного корня `sqrt()`, необходимо подключить к программе математическую библиотеку. Для этого в модуле программы после строки `#include "USquare.h"` впишите строку подключения библиотеки с математическими функциями:

```
#include <math.h>.
```

Теперь сформируйте обработчик события **OnClick** для кнопки **Выход** и вставьте в него программную строку для закрытия приложения:

```
Close();
```

Приложение уже можно запустить, и оно будет работать. Но в нем не будет работать главное меню. Создадим поддержку для него.

Сформируйте обработчик события **OnClick** для первого пункта меню **N1** компонента **MainMenu1** путем двойного щелчка левой кнопкой мыши по самому компоненту и вставьте в него строку для модального вызова формы **FormAbout** с описанием программы:

```
FormAbout->ShowModal();
```

Модальный вызов не позволяет программе переключиться на другое окно до тех пор, пока текущее окно не будет закрыто. В отличие от него немодальное окно вызывается функцией `Show()` и не налагает такого условия.

Поскольку формы с описанием программы нет, ее необходимо создать. Для этого выполните из главного меню команду: **File | New | Form**. Для вновь созданной формы задайте свойство **Name** с именем **FormAbout** и свойство **Caption** с названием **О программе**.

Теперь поместите на эту форму четыре компонента **Label**, один компонент **Image** и одну кнопку **BitBtn** из палитры **Additional**, а также компонент **MonthCalendar** из палитры **Win32**.

Для компонентов **Label1—Label4** задайте свойство **Caption** с названиями: **Программа решения квадратных уравнений**, **Версия 1.0**, **Послать E-mail** и **Поиск в Интернете**.

Для компонентов **Label3** и **Label4** установите шрифт через свойство **Font** как синий, подчеркнутый, полужирный курсив. Свойство **Cursor** для этих компонентов поменяйте на **crHandPoint**. Это делается для того, чтобы выделить надписи данных компонентов и изменить при наведении на них вид курсора мыши.

Создайте обработчик события **OnClick** для компонента **Label3** и вставьте в него программную строку вызова почтовой программы:

```
ShellExecute(Handle,"open","mailto:sandh@narod.ru?x-
priority=normal&subject=TEMA_ПИСЬМА",NULL,NULL,SW_SHOWNORMAL);
```

Здесь в программной строке можно изменить адрес получателя письма `sandh@narod.ru` и заполнить тему письма собственным текстом.

Далее создайте обработчик события **OnClick** для компонента **Label4** и вставьте в него программную строку вызова обозревателя и вхождения с его помощью в Интернет:

```
ShellExecute(Handle, "open", "http://www.ya.ru", 0, 0, SW_SHOW);
```

Естественно, что адрес страницы http://www.ya.ru можно заменить на любой другой подходящий и существующий адрес.

Для кнопки **BitBtn1** установите свойство **Kind**, как **bkOk**. Создайте обработчик события **OnClick** для этой кнопки и вставьте в него программную строку выхода:

```
Close();
```

Наконец, скопируйте в каталог с программой предварительно созданный файл с рисунком или фотографией в формате BMP или JPG и загрузите его через свойство **Picture** компонента **Image**. Для того чтобы рисунок совпадал с размерами компонента **Image**, установите его свойство **AutoSize** в значение **true**. Расположите все компоненты на форме в соответствии с рис. 24.2 и сохраните модуль программы в файле с именем about.cpp.

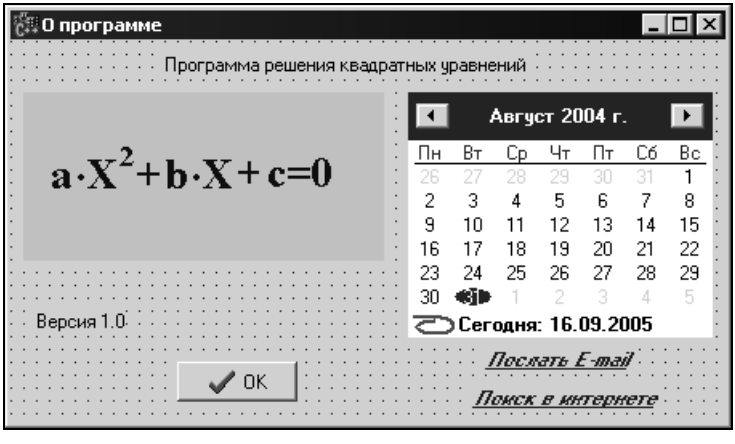


Рис. 24.2. Расположение компонентов на дополнительной форме программы

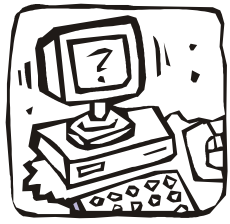
Поскольку мы добавили в наше приложение еще одну форму, необходимо включить в основной модуль Usquare.cpp имя файла описания новой формы с названием about.h. Для этого впишите в модуле Usquare.cpp после строки `#include <math.h>` программную строку:

```
#include "About.h".
```

Теперь остается лишь заменить значок-иконку программы с помощью команды главного меню **Projec | Options | Application** на созданный во встроенном редакторе **Image Edit** файл иконки с расширением ico и откомпилировать программу.

Программа готова. Запустите ее на выполнение и проверьте в работе.

В этой программе мы использовали новый компонент **MonthCalendar**, который позволяет без труда внедрить в программу красочный календарь с богатыми функциями. Кроме того, мы научились запускать из программы почтовую службу отправки электронных писем и входить в Интернет из работающей программы. Подобные приемы можно применять во вновь создаваемых программах.



Глава 25

Программа *MultiPlayer*

В этой главе рассмотрен пример программы, позволяющей управлять мультимедийным оборудованием, таким как проигрыватель видеофайлов и устройство CD-ROM.

На вкладке **System** располагается компонент **MediaPlayer**, с помощью которого можно управлять устройствами мультимедиа на компьютере. Данный компонент будет служить основой для создания программы **MultiPlayer**.

Определимся с необходимыми компонентами и сформируем интерфейс программы. Создайте новое приложение и поместите на его форму компонент **MediaPlayer**. Он будет выполнять в программе функции органов управления для проигрывания файлов, их остановки и перемотки. Теперь поместите на форму компонент **Panel** из вкладки **Standard**. Этот элемент послужит в качестве экрана для вывода изображения. Далее поместите на форму компонент **OpenDialog** из вкладки **Dialog** для обеспечения открытия окна загрузки файлов. Для создания в программе главного меню поместите на форму компонент **MainMenu** из вкладки **Standard**. С целью придания программе удобства пользования поместите на форму компонент **ToolBar**, который позволит создать кнопки для выполнения различных команд. Наконец, поместите на форму компонент **ImageList** из палитры **Win32** для загрузки и хранения в нем рисунков иконок кнопок и пунктов главного меню.

Теперь необходимо задать свойства некоторых компонентов в соответствии с табл. 25.1 по аналогии с тем, как мы это уже неоднократно делали в предыдущих программах. Остальные свойства можно оставить такими, какими они были по умолчанию.

Кроме того, можно отключить некоторые кнопки в компоненте **MediaPlayer1** с помощью его свойств **VisibleButtons**, установив значение **false** для ненужных кнопок, например для кнопки записи **btRecord**.

У компонента **OpenDialog1** необходимо настроить свойство фильтра файлов. Чтобы это сделать, щелкните левой кнопкой мыши по кнопке с тремя

точками, расположенной правее поля **Filter**, и в открывшемся окне редактора **Filter Editor** введите строки в соответствии с рис. 25.1.

Таблица 25.1. *Свойства компонентов*

Компонент	Свойство	Значение
MediaPlayer1	AutoOpen	false
	Cursor	CrHandPoint
	Display	Panel1
Panel1	Caption	Пустая строка

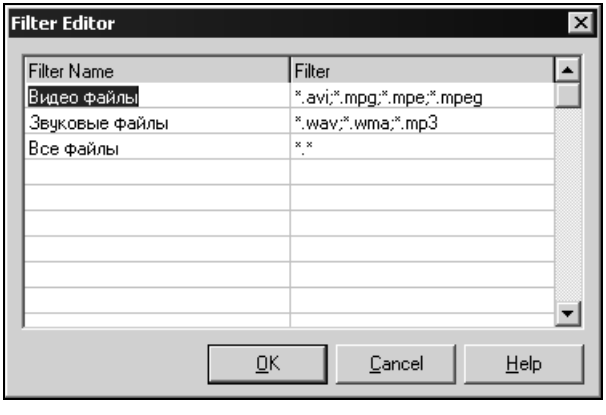


Рис. 25.1. Окно редактора **Filter Editor**

Из содержимого этих строк видно, каким образом осуществляется фильтрация файлов. При желании вы можете изменить или добавить свои строки в поля редактора.

Для того чтобы поставить в соответствие кнопкам и пунктам меню рисунки иконок, необходимо загрузить файлы таких рисунков в компонент **ImageList**. Это делается путем двойного щелчка левой кнопкой мыши по компоненту **ImageList**. При этом откроется окно, показанное на рис. 25.2.

Щелкните в этом окне по кнопке **Add** для добавления в компонент файлов с рисунками. При этом откроется окно выбора файлов (рис. 25.3).

Выберите в нем каталог C:\Program Files\Common Files\Borland Shared\Images\Buttons\ и загрузите файлы с рисунками, подходящими по смыслу для кнопок с функциями открытия и закрытия файлов, проигрывания и останова мультимедийных файлов, открытия и закрытия дверки CD-ROM и для выхода из программы. Обратите внимание на то, что все рисунки состоят из двух частей: цветной и серо-белой. Это позволит в программе показывать кнопки как доступные, так и недоступные.

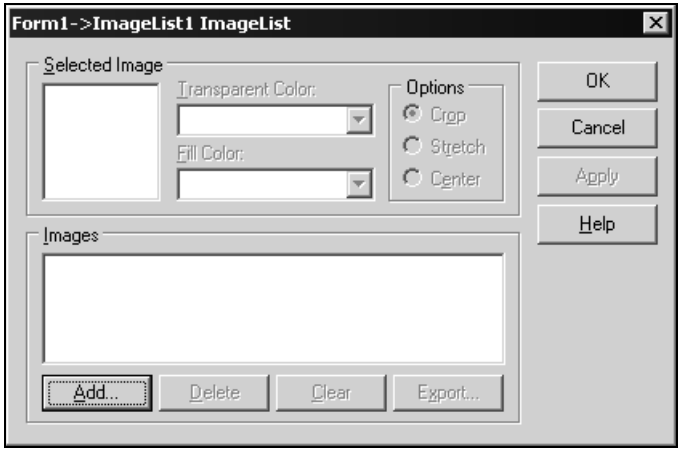


Рис. 25.2. Окно редактора **ImageList**

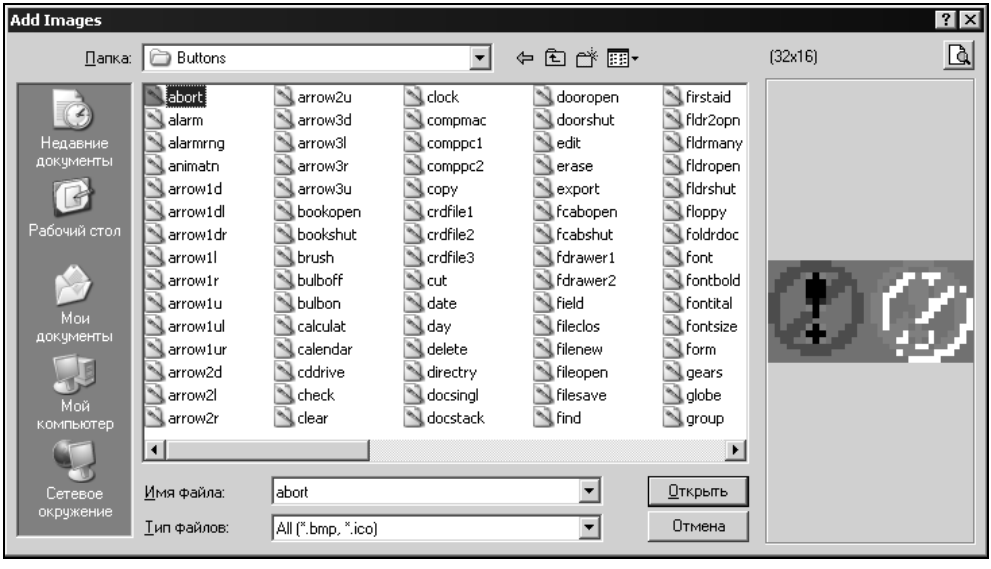


Рис. 25.3. Окно выбора файлов

Каждый раз при загрузке очередного файла рисунки будут помещаться в окно формы заполнения компонента **ImageList1** с автоматическим присвоением им номеров от нуля. В дальнейшем эти номера рисунков будут использоваться в программе как индексы рисунков.

Для обеспечения доступа к рисункам **ImageList1** компоненты **ToolBar1** и **MainMenu1** должны иметь в свойстве **Images** имя компонента **ImageList1**.

Оно выбирается из разворачиваемого списка в свойстве **Images** инспектора объектов (рис. 25.4).

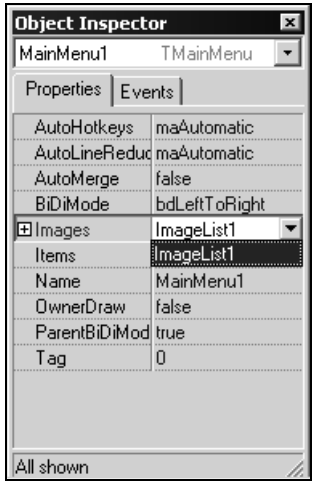


Рис. 25.4. Окно инспектора объектов

Теперь создайте пункты главного меню и панели инструментов. Для этого щелкните дважды левой кнопкой мыши по компоненту **MainMenu1**, и в открывшемся окне редактора меню задайте свойство **Caption** в виде строки **File** для первого пункта меню и нажмите клавишу <Enter>.

Далее щелкните по пункту меню **File**, а затем по пустому полю, появившемуся ниже. Задайте для него свойство **Caption** в виде строки **Open**. Ниже создайте пункт с названием **Close**. Еще ниже — разделитель, присвоив свойству **Caption** значение в виде символа "-" (минус). Последним в разделе **File** создайте пункт меню **Exit**. Правее пункта **File** создайте еще две группы пунктов меню с названиями **Play** и **CD** (рис. 25.5).

Группу **Play** дополните пунктами меню **Go** и **Pause**, а группу **CD** — пунктами **Open** и **Close**. Закройте окно редактора меню.

Для добавления кнопок панели инструментов **ToolBar1** необходимо щелкнуть по ней правой кнопкой мыши, и в открывшемся окне (рис. 25.6) выбрать команду **New Button**.

При этом на панели инструментов появится новая пустая кнопка. Вновь откройте окно редактирования панели и выберите команду **New Separator**. На панели инструментов появится разделитель. Аналогично добавьте на панель две сдвоенные кнопки (с разделителем) и одну одиночную.

Теперь задайте компонентам созданных кнопок осмысленные имена с помощью свойства имен **Name** в соответствии с табл. 25.2.

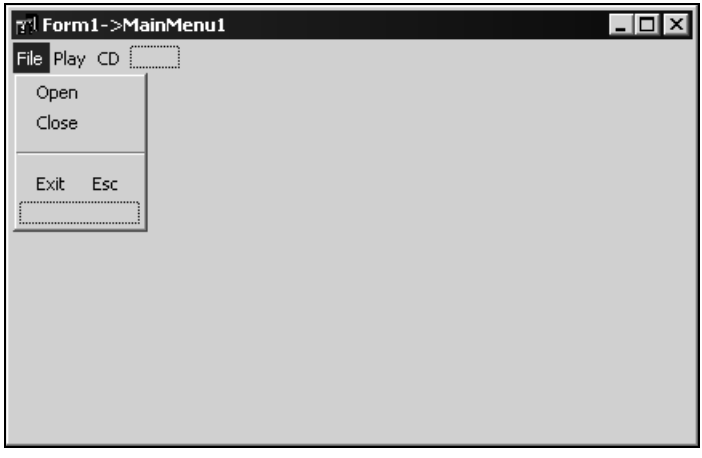


Рис. 25.5. Окно редактора меню

Рис. 25.6. Окно редактора **ToolBar1**

Таблица 25.2. *Имена компонентов*

Компонент	Значение свойства Name	Компонент	Значение свойства Name
ToolButton1	ToolButtonFileOpen	ToolButton6	ToolButtonSp2
ToolButton2	ToolButtonFileClose	ToolButton7	ToolButtonCDOpen
ToolButton3	ToolButtonSp1	ToolButton8	ToolButtonCDClose
ToolButton4	ToolButtonPlay	ToolButton9	ToolButtonSp3
ToolButton5	ToolButtonPause	ToolButton10	ToolButtonExit

Приступайте к графическому оформлению кнопок и пунктов меню. Начните с кнопок панели инструментов **ToolButton1**. Выделите первую слева кнопку на этой панели, и в инспекторе объектов найдите для нее свойство **ImageIndex**. Щелкните левой кнопкой мыши на поле правее этого свойства, а затем по кнопке со стрелкой, направленной вниз, для раскрытия списка доступных рисунков. Выберите из этих рисунков, загруженных вами в компонент ранее, рисунок, подходящий по смыслу для команды открытия файла. При этом свойство **ImageIndex** примет значение, соответствующее индексу выбранного рисунка.

Повторите эту операцию для остальных кнопок панели инструментов за исключением разделителей. Естественно, что кнопкам должны назначаться рисунки, подходящие по смыслу выполняемых ими команд. В результате должна получиться форма, похожая на представленную на рис. 25.7.

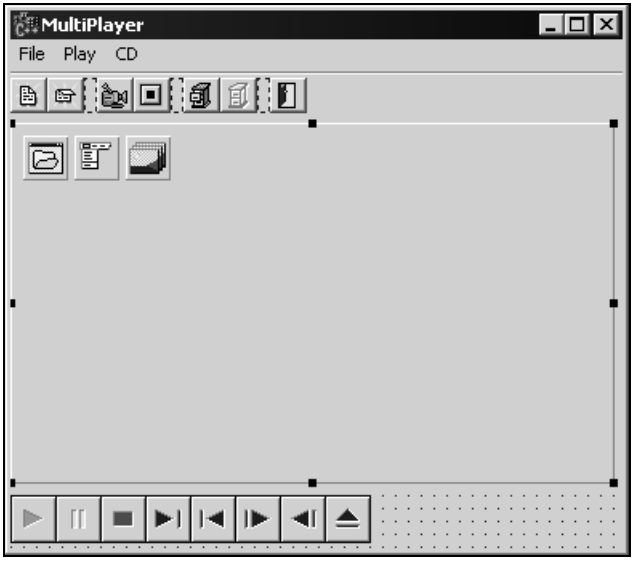


Рис. 25.7. Окно формы с компонентами

Проделайте аналогичную операцию для пунктов главного меню программы **MainMenu1**. Для этого щелкните дважды левой кнопкой мыши по компоненту **MainMenu1** и в открывшемся редакторе меню выберите первый пункт **File**. Задайте его свойство **ImageIndex** аналогично тому, как это было сделано для первой кнопки панели инструментов. Подобным образом назначьте индексы рисунков всем остальным пунктам меню и закройте редактор меню. Теперь при запуске программы пункты меню будут содержать не только текст команды, но и рисунок, соответствующий этой команде, как показано на рис. 25.8.

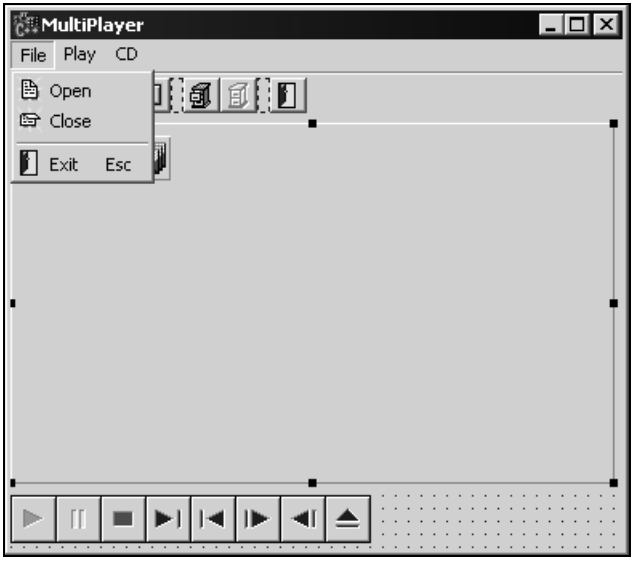


Рис. 25.8. Окно программы с рисунками в меню команд

Такое оформление обогащает интерфейс программы и делает ее намного привлекательнее для пользователей.

Осталось ввести код программы для обработчиков команд — и программа будет готова. С целью сокращения трудозатрат и придания программе изящности создайте обработчики команд только для пунктов главного меню, а кнопкам панели инструментов назначьте имена созданных обработчиков, чтобы не дублировать записи.

Начните с пункта меню **Open** группы **File**. Выберите этот пункт меню, при этом откроется редактор кода с заготовкой функции обработки события **OnClick**:

```
void __fastcall TForm1::Open1Click(TObject *Sender)
{
}
```

Впишите между фигурными скобками этого обработчика следующие программные строки:

```
if(OpenDialog1->Execute()){
 MediaPlayer1->FileName=OpenDialog1->FileName;
 MediaPlayer1->Open();}
```

Они позволят выполнить диалог открытия файлов и загрузить его в компонент **MediaPlayer1**.

Аналогично назначьте функции обработки для пункта меню **File | Close**, введя в заготовку обработчика команды код программы:

```
MediaPlayer1->Close();.
```

Для пункта меню **Play | Go** введите программную строку проигрывания файла:

```
MediaPlayer1->Play();.
```

Пункт меню **Play | Pause** снабдите обработчиком с программной строкой приостановки проигрывания файла:

```
MediaPlayer1->Pause();.
```

Обработчик команды меню **CD | Open** заполните программными строками открытия устройства CD-ROM:

```
mciSendString("Set cdaudio door open wait", NULL, 0, Handle);.
```

Для пункта меню **CD | Close** введите программную строку закрытия привода CD-ROM:

```
mciSendString("Set cdaudio door closed wait", NULL, 0, Handle);.
```

Все приведенные здесь программные строки вместе с проектом приложения MultiPlayer можно скопировать из каталога \PROJECT\MULTIPLAYER с диска, прилагаемого к книге.

Замените значок (иконку) программы с помощью команды главного меню **Project | Options | Application** на предварительно созданный во встроенном редакторе **Image Edit** файл иконки с расширением ico. Сохраните и откомпилируйте программу.

Теперь программа будет работать. Осталось активизировать кнопки панели инструментов. Для этого зададим их обработчик события **OnClick**. Щелкните левой кнопкой мыши по кнопке открытия файла на панели инструментов программы. В инспекторе объектов откройте вкладку событий **Events**. Щелкните по полю правее события **OnClick** и выберите в раскрывающемся списке имя обработчика события **Open1Click**. Этот обработчик был создан нами для выполнения пункта главного меню **File | Open**. Аналогично назначьте имена обработчиков событий для остальных кнопок панели инструментов программы. Имена обработчиков несут в себе смысл выполняемой ими команды, а сами обработчики принадлежат пунктам главного меню программы.

Программа готова. Запустите ее и проверьте, как она работает. При необходимости устраните обнаруженные компилятором ошибки.

Внимание

Если вы случайно ввели ненужные заготовки обработчиков, не удаляйте их самостоятельно. Просто удалите весь текст между фигурными скобками этих обработчиков, а компилятор самостоятельно аккуратно удалит лишние функции из программы.

Добавьте в программу еще один эффект. Ранее было упомянуто, что рисунки для иконок состоят из двух частей: цветной и серо-белой, что позволяет программе

показывать кнопки как доступные, так и недоступные. Воспользуемся этим. Сделаем так, чтобы при запуске программы доступной была кнопка панели инструментов **CD Open**, а кнопка **CD Close** — недоступной. И пусть после каждого открытия и закрытия CD-ROM эта ситуация меняется на противоположную. Для этого изменим свойство **ImageIndex** компонента **ToolButtonCDClose** на индекс серо-белого рисунка закрытия CD-ROM. Аналогично необходимо поступить с компонентом пункта **CD | Close** главного меню программы. Кроме того, в обработчике открытия CD-ROM необходимо добавить строки программы для изменения порядка индексов рисунков компонента **ImageList1**:

```
ImageList1->Move(10,11);
ImageList1->Move(13,12);
```

А в обработчике закрытия CD-RO надо добавить строки программы для восстановления первоначального порядка индексов рисунков компонента **ImageList1**:

```
ImageList1->Move(12,13);
ImageList1->Move(11,10);.
```

Теперь после запуска программы вы увидите, что кнопка закрытия устройства CD-ROM будет серой в отличие от кнопки **CD Open**. А после операции открытия CD-ROM картина поменяется на противоположную.

Обратите внимание, то же самое происходит и с рисунками пунктов главного меню программы для команд открытия и закрытия CD-ROM!

Введите в программу еще одну полезную функцию, которая позволит закрывать приложение при нажатии кнопки <Esc> на клавиатуре. Для этого сформируйте для формы **Form1** приложения обработчик события **OnKeyPress** и введите в него программную строку:

```
if(Key==VK_ESCAPE) Close();.
```

Теперь программу легко закрыть, нажав одну-единственную кнопку <Esc> на клавиатуре.

Для того чтобы в программе появилась памятка об этом, задайте в свойстве **ShortCut** для пункта меню **File | Exit** значение в виде строки **Esc**. После этого данная строка будет присутствовать в записи самого пункта меню как напоминание об этой возможности. Аналогично можно поступить и с другими пунктами меню, назначив для их выполнения новые клавиши или их комбинации.

Для придания нового качества программе научимся сворачивать ее в так называемый **tray** (Поддон), который представляет собой служебное подвальное окно на рабочем столе компьютера, где обычно размещаются значки программ часов, регулятора громкости, настройки экрана и т. д. (рис. 25.9).

Рис. 25.9. Служебное подвальное окно

Для этого поместите на форму приложения компонент **Tray** из вкладки **Samples** и задайте его свойство **Visible** равным **true**. Теперь при запуске и сворачивании программы ее значок будет всегда находиться в подвале рабочего стола для служебных программ. Разворачивание программы осуществляется двойным щелчком левой кнопки мыши по этому значку.

Наконец придайте программе еще большую изящность, добавив в нее контекстное меню. Как известно, это меню вызывается нажатием правой кнопки мыши на поле программы. Создайте меню, состоящее из двух пунктов-команд: разворачивающей и сворачивающей программу. Поместите на форму программы компонент **PopupMenu** из вкладки **Standard**. Щелкните по нему дважды левой кнопкой мыши и в открывшемся редакторе пунктов меню создайте пункты **Развернуть** и **Свернуть** (рис. 25.10) по аналогии с тем, как это делалось при создании пунктов главного меню программы.

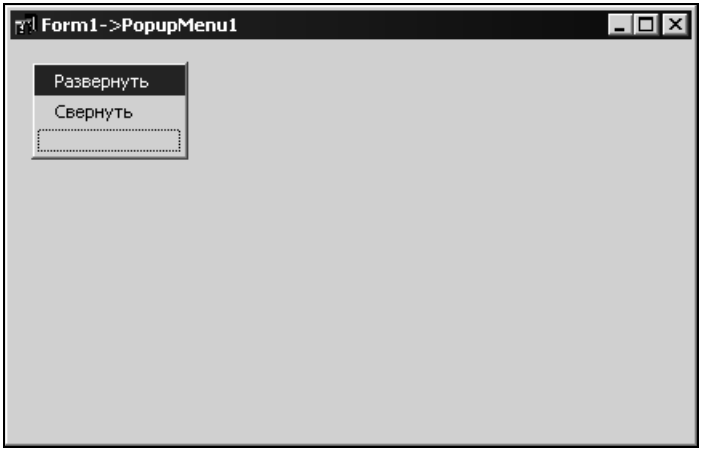


Рис. 25.10. Окно редактора пунктов контекстного меню

Закройте редактор пунктов контекстного меню.

Теперь необходимо обеспечить доступ из главной формы **Form1** и из компонента **TrayIcon1** к этому контекстному меню. Для этого в инспекторе объектов задайте для свойства **PopupMenu** этих компонентов значение **PopupMenu1**, выбрав его из разворачиваемого списка. Имя данного компонента появилось в списке после того, как мы поместили этот компонент на форму приложения. Естественно, что таких компонентов может быть несколько, они будут автоматически получать порядковые номера и могут использоваться в программе разными компонентами. В нашей программе можно обойтись одним-единственным компонентом контекстного меню.

Теперь, когда созданы пункты контекстного меню, необходимо сформировать для них обработчики события **OnClick**. О том, как это делается, неоднократно говорилось ранее, поэтому описание этой процедуры опускается.

Для пункта меню **Развернуть** впишите в обработчик события **OnClick** программную строку:

```
TrayIcon1->Restore();.
```

Следующую программную строку вставьте в обработчик события **OnClick** для пункта меню **Свернуть**:

```
TrayIcon1->Minimize();.
```

Теперь при запуске программы контекстное меню будет появляться для развернутого приложения. После сворачивания программы в подвал меню будет недоступно. Для того чтобы оно стало доступным из подвала, замените значение **imNone** свойства **PopupMenuOn** компонента **TrayIcon1** на значение **imRightClickUp**, которое позволит вызывать меню по щелчку правой кнопки мыши на компоненте.

Пункты контекстного меню можно оформить графически так же, как это сделано для пунктов главного меню. Для этого задайте в свойстве **Images** компонента **PopupMenu1** имя **ImageList1**, выбрав его из разворачивающегося списка, и добавьте в компонент **ImageList1** пару рисунков (рис. 25.11).

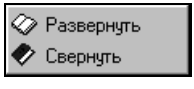


Рис. 25.11. Окно контекстного меню

Для придания программе большей привлекательности можно добавить в нее всплывающие подсказки, которые появляются при наведении курсора мыши на быструю кнопку и сообщают пользователю ее назначение. Для этого необходимо задействовать свойство **Hint** компонентов быстрых кнопок.

Активизируйте окно инспектора объектов и щелкните левой кнопкой мыши по кнопке программы, отвечающей за открытие файла. Инспектор объектов автоматически переключится на редактирование свойств этого компонента. В строке **Hint** введите строку **Открыть файл**, которая будет появляться в качестве справки.

Теперь установите свойство **ShowHint** этого же компонента в состояние **true**, для того чтобы разрешить отображение строки справки при работе программы. Аналогичную процедуру необходимо проделать для остальных быстрых кнопок программы, задав их свойству **Hint** значения в виде строк: **Закрыть файл**, **Начать**, **Пауза**, **Открыть CD-ROM** и **Закрыть CD-ROM** соответственно.

Запустите программу и проверьте, что она работает так, как было задумано.

Вот теперь стиль приложения выдержан хорошо. Все усилия по созданию этой программы не пропадут напрасно, поскольку они позволили освоить на практике способы создания красивого графического интерфейса, которые могут быть применены многократно при создании новых программ.

Заключение

В заключение я хочу сообщить очень приятный факт для тех читателей, кто от начала до конца прочел эту книгу про Borland C++ Builder 6. Дело в том, что интерфейс описанной среды разработки практически полностью совпадает с интерфейсом Delphi 7 (рис. 31). Поэтому, освоив Borland C++ Builder 6, читатели, сами того не подозревая, освоили и интерфейс Delphi 7. С чем я вас и поздравляю!

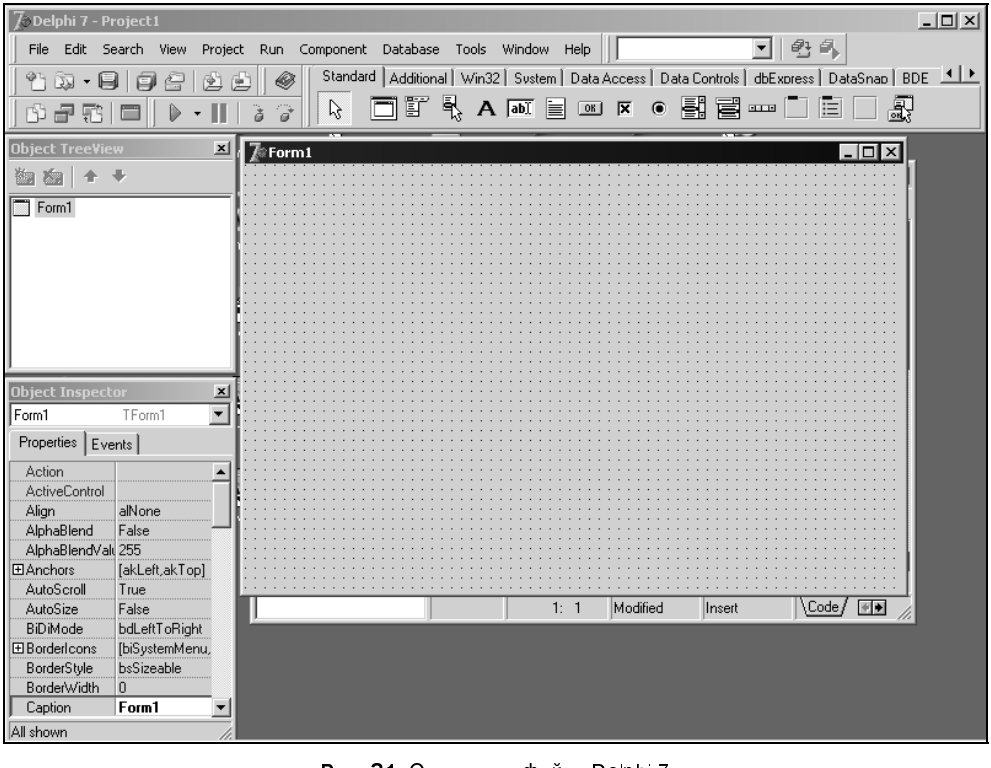


Рис. 31. Окно интерфейса Delphi 7

Отличие при написании программ на Borland C++ Builder 6 и Delphi 7 заключается в основном лишь в использовании разных языков программирования. Если в Borland C++ Builder 6 используется язык C++, то в Delphi 7 — Паскаль. Остальные отличия несущественные.

Таким образом, можно с уверенностью сказать, что теперь, после изучения небольшого курса Borland C++ Builder 6, читатели вполне смогут разобраться в программах, написанных на Delphi 7, подучив перед этим язык Паскаль.

Совсем недавно компания Borland объявила о выходе новейших версий C++ Builder 2006 и Delphi 2006. Это вовсе не означает, что Borland C++ Builder 6 окончательно устарел и про него можно забыть. Напротив, как азбука является основой для обучения чтению, так и Borland C++ Builder 6 является школой для обучения программированию. В подтверждение этих слов я предлагаю читателям взглянуть на окно интерфейса среды разработки C++ Builder 2006, приведенный на рис. 32.

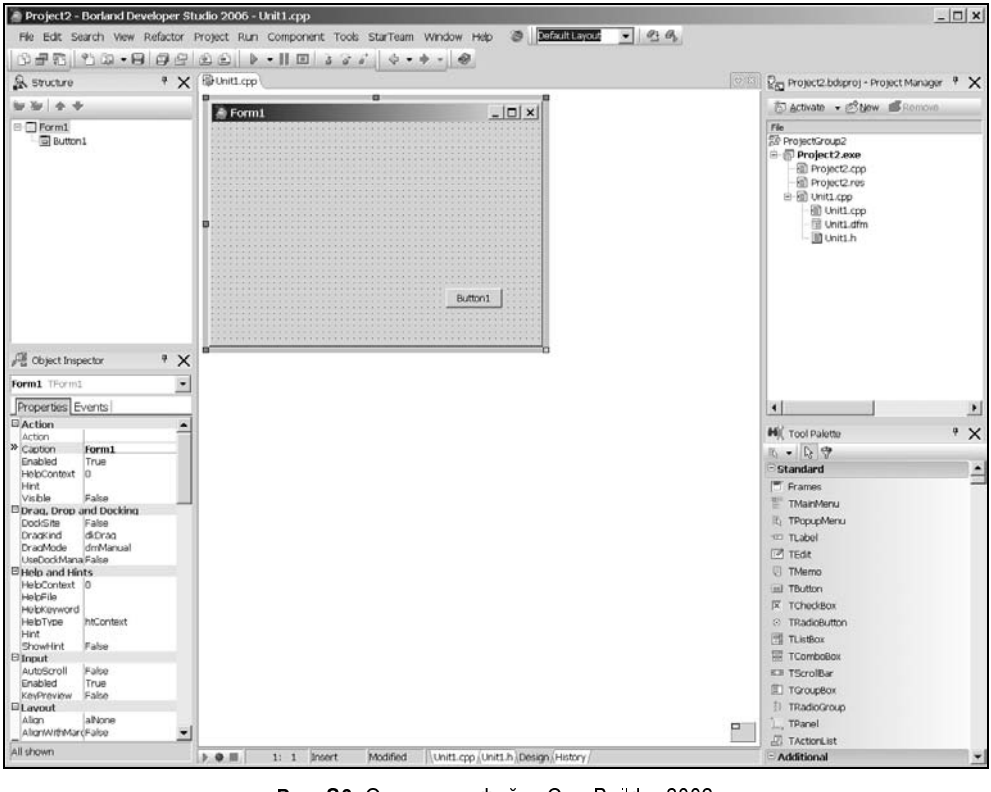


Рис. 32. Окно интерфейса C++ Builder 2006

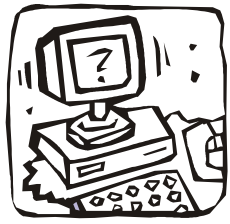
Здесь без труда легко заметить сходство нового интерфейса с изученным нами интерфейсом Borland C++ Builder 6. Разве что новый интерфейс более красив. Каким и следует быть новым продуктам. Но, изучив Borland C++ Builder 6, вы уже без особого труда начнете создавать программы в новой среде разработки.

Если вы последовательно прочли весь изложенный мною учебный материал от начала до конца и повторили создание программ на основе рассмотренных примеров, то смело можете приступать к самостоятельному созданию новых программ. Если вам понадобится более детальное описание среды разработки, то воспользуйтесь источником[1].

Кроме того, в каталоге Examples Borland C++ Builder 6 имеется множество примеров рабочих программ. Изучайте эти примеры, и вскоре вы активно будете программировать...

[1] Архангельский А. Я. Программирование в C++ Builder 6. — М.: БИНОМ, 2003.

ПРИЛОЖЕНИЯ



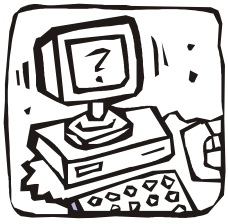
Приложение 1

Горячие клавиши

Клавиши	Описание
Клавиши для редактирования проекта	
<Ctrl>+<Space>	Подсказка кода программы
<Ctrl+Shift>+<Space>	Подсказка параметров кода программы
<Ctrl>+<J>	Подсказка операторов программы
<Ctrl>+<F11>	Открыть проект
<Ctrl>+<F12>	Открыть модуль проекта
<Shift>+<F12>	Открыть форму проекта
<F12>	Переключение между формой и файлом модуля
Клавиши для выполнения проекта	
<F9>	Выполнить приложение
<F8>	Пошаговое выполнение программы без захода в функции
<F7>	Пошаговое выполнение программы с заходом в функции
<F4>	Выполнение программы до курсора
Клавиши для выделения кода программы	
<Shift>+<Стрелка вправо>	На один знак вправо
<Shift>+<Стрелка влево>	На один знак влево
<Ctrl>+<Shift>+<Стрелка вправо>	До конца слова
<Ctrl>+<Shift>+<Стрелка влево>	До начала слова

(окончание)

Клавиши	Описание
Клавиши для выделения кода программы	
<Shift>+<End>	В конец строки
<Shift>+<Home>	В начало строки
<Shift>+<Стрелка вниз>	На одну строку вниз
<Shift>+<Стрелка вверх>	На одну строку вверх
<Ctrl>+<Shift>+<Стрелка вниз>	До конца абзаца
<Ctrl>+<Shift>+<Стрелка вверх>	До начала абзаца
<Shift>+<Page Down>	На один экран вниз
<Shift>+<Page Up>	На один экран вверх
<Ctrl>+<Shift>+<Home>	В начало текста
<Ctrl>+<Shift>+<End>	В конец текста
<Alt>+<Ctrl>+<Shift>+<Page Down>	До конца текста
<Ctrl>+<A>	Выделить весь текст



Приложение 2

Содержание компакт-диска

На компакт-диске представлены следующие материалы:

Каталог	Описание	Глава
\PROJECT\ABC	Проект программы ABC, включающий в себя файлы с исходными текстами кода программ и необходимые для трансляции	19
\PROJECT\ABC\INST	Инсталляционный файл программы ABC	19
\PROJECT\ABC\ PICTURE	Каталог с файлами рисунков для программы ABC	19
\PROJECT\ABC\ SOUND	Каталог со звуковыми файлами для программы ABC	19
\PROJECT\DRAW	Проект программы графического рисования, включающий в себя файлы с исходными текстами кода программ и необходимые для трансляции	23
\PROJECT\EDIT	Проект программы простого текстового редактора	21
\PROGRAM	Исполняемые файлы всех проектов из каталога \PROJECT, не требующие установки на компьютер Borland C++ Builder 6	
\PROJECT\FUNCTION	Проект программы рисования графика функции	23
\PROJECT\MULTIPLAYER	Проект программы мультимедийного проигрывателя	25
\PROJECT\PROG1	Проект первой учебной программы	10
\PROJECT\PROG2	Проект второй учебной программы	13
\PROJECT\SQUARE	Проект программы для вычисления корней квадратного уравнения	24
\PROJECT\SPISOK	Проект программы для отображения списка вводимых данных в программу	23
\PROJECT\TIMEDATE	Проект программы "Часы-календарь"	17
\VIDEO	Учебные видеофайлы	

Предметный указатель

Б

Быстрые кнопки 31, 44
- Add 44
- Help 46
- New 44
- Pause 46
- Remove 44
- Run 46
- Save 44
- Step 46
- Toggle 45
- Trace 46
- View 45

В

Визуальные компоненты 103
- Action List 107
- Addition 108
- Animate 110
- Bevel 109
- BitBtn 108
- Button 106
- CCalendar 114
- CColorGrid 113
- CDirectory Outline 114
- CGauge 114
- Chart 109
- CheckBox 106
- CheckListBox 109
- ClientSockets 116
- ColorDialog 112
- ComboBox 107
- ComboBoxEx 111
- CoolBar 111
- CSpinButton 113
- CSpinEdit 113
- Data Access 115
- Data Control 115
- DataTimePicker 110
- DdeClientConv 111
- DdeClientItem 111
- DdeServerConv 112
- DdeServerItem 112
- Dialog 112
- DrawGrid 109
- Edit 106
- FindDialog 113
- FontDialog 112
- Frame 105
- GroupBox 107
- HeaderControl 110
- HotKey 110
- Image 109
- Image List 110
- InetXPage Producer 115
- Internet 116

(окончание рубрики см. на стр. 206)

Визуальные компоненты *(окончание)*:
Internet Express 115
Label 106
ListBox 107
ListView 110
MainMenu 106
MaskEdit 108
MediaPlayer 111
Memo 106
MonthCalendar 110
Office2k 114
OleContainer 111
OpenDialog 112
OpenPictureDialog 112
PageControl 109
PageScroller 111
PaintBox 111
Panel 107
Performance Graph 113
Pie 114
PopupMenu 106
PrintDialog 112
PrinterSetupDialog 113
Progress Bar 110
RadioButton 107
RadioGroup 107
Replace Dialog 113
RichEdit 110
Samples 113
SaveDialog 112
SavePictureDialog 112
ScrollBar 107
ScrollBox 109
ServerSockets 116
Shape 109
SpeedButton 108
Splitter 109
Standard 105
StaticText 109
StatusBar 110
StringGrid 109
System 111
TabControl 109
Timer 111
ToolBar 111
TrackBar 110
Tray Icon 113
TreeView 110
UpDown 110
Win32 109
XML Broker 115
VCL 47

Г

Главное меню 37
About 43
Component 41
Database 41
Edit 38
File 37
Help 42
Project 40
Run 40
Search 39
Tools 42
View 39
Windows 42

Д

Дизайнер форм 32, 50

И

Инспектор объектов 32, 58

М

Метод 4
Add 166
Arc 170
CanFocus 167
ChangeScale 167
Chord 169
Delete 166, 167
Ellipse 169
FillRect 172
FloodFill 172
Hide 165, 167
LineTo 171
MoveTo 171

Pie 170
Rectangle 170
RoundRect 170
Show 165, 167
TextOut 170
TextRect 171
Методы компонентов 160

О

Объект 4
Объектно-ориентированное программирование 4

П

Палитра компонентов 47
Add 49
Delete 49
Help 49
Hide 49
Move 49
Properties 49
Show 49
Tabs 49

Р

Редактор кода 32, 53

С

Свойства компонентов 123
Событие 4
OnChange 157
OnClick 147
OnDblClick 148
OnDragDrop 158
OnDragOver 158
OnEndDrag 159
OnEnter 155
OnExit 155
OnKeyDown 149
OnKeyPress 153
OnKeyUp 153
OnMouseDown 155
OnMouseMove 156
OnPaint 157
OnProgress 157
OnStartDrag 159