ЧАСТЬ

I

# Основы C++Builder

**В ЭТОЙ ЧАСТИ...**

1. **Введение в C++Builder**
2. **Проекты C++Builder и дополнительные сведения об IDE-среде**
3. **Программирование в C++Builder**
4. **Создание пользовательских компонентов**
5. **Создание редакторов свойств и компонентов**

ГЛАВА

1

# Введение в C++Builder

**В ЭТОЙ ГЛАВЕ...**

- Язык C++
- Что нового в C++Builder 6

В этой главе представлен программный продукт компании Borland C++Builder — одна из ведущих сред разработки для создания Internet-приложений, "настольных" и распределенных приложений, а также приложений, основанных на модели клиент/сервер. C++Builder сочетает простоту среды быстрой разработки приложений, или RAD-среды (Rapid Application Development — RAD), с мощью и производительностью языка C++, совместимого со стандартом ANSI.

Основная часть работы по созданию приложений выполняется в интегрированной среде разработки (Integrated Development Environment — IDE) C++ Builder, пользовательский интерфейс которой показан на рис. 1.1.

Язык программирования C++ остается наиболее распространенным. Он применяется для разработки самых разных приложений — от сложных многоуровневых бизнес-систем до высокопроизводительных программ визуализации данных и систем реального времени. C++Builder является отличным средством для реализации любых

**НА ЗАМЕТКУ**

Более подробную информацию о преимуществах использования среды C++Builder можно найти по ссылкам `Features & Benefits` и `New C++Builder Users` на Web-странице по адресу: `http://www.borland.com/bcppbuilder/`.

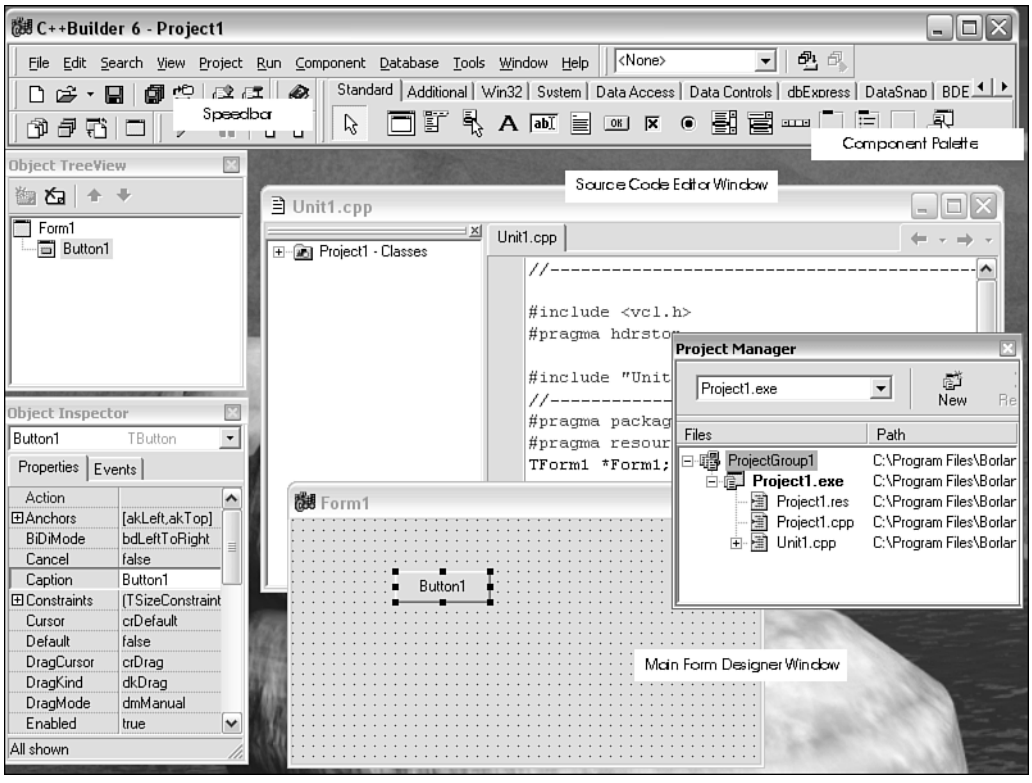


**Рис. 1.1.** *Интерфейс интегрированной среды разработки C++Builder*

# Язык C++

Язык C++ является ядром среды C++Builder, которая обеспечивает очень высокую степень поддержки этого стандартизированного языка программирования.

## Соответствие стандартам ANSI

Язык C++ появился в результате развития языка C, созданного в компании Bell Labs. Подобно C, язык C++ получил всеобщее признание и широкое распространение.

Широкая распространенность C++ способствовала конкуренции между поставщиками компиляторов и сред разработки, что сыграло положительную роль. Однако, поскольку поставщики стремились к увеличению своего технологического влияния, они часто вносили в язык программирования собственные уникальные особенности. Хотя такие доработки и были полезны, они приводили к несовместимости программ, написанных для одного компилятора, с другими компиляторами.

Американский национальный институт стандартов (American National Standards Institute — ANSI), основанный в 1918 году как частная некоммерческая организация, занимается координацией и определением промышленных стандартов в различных областях. Таким образом, ANSI стал наиболее подходящей организацией, которая могла взяться за проблему стандартизации языков C и C++.

**НА ЗАМЕТКУ**

Результатом работы по стандартизации стал документ, доступный на Web-узле ANSI по адресу: `http://webstore.ansi.org/ansidocstore/product.asp?sku=ISO%2FIEC+14882%3A1998`. Документ был завершен в 1998 году.

В XXI веке соответствие стандарту ANSI стало одним из наиболее важных свойств языка. Этот стандарт позволяет создавать и компилировать программы в разных системах разработки, для различных операционных систем и наборов процессорных инструкций.

Компания Borland предлагает мощный набор собственных расширений языка C++, обеспечивающий поддержку VCL-библиотек (Visual Component Library). Однако эта компания также представила язык, соответствующий стандартам ANSI, возможно, в большей мере, чем остальные. Компилятор можно настроить таким образом, чтобы он воспринимал только те программы, которые соответствуют стандарту ANSI. Для этого достаточно изменить одну настройку среды программирования. Выберите команду меню Project⇨Options..., перейдите в диалоговом окне Project Options... во вкладку Advanced Compiler и установите переключатель Language compliance в положение ANSI, как показано на рис. 1.2.

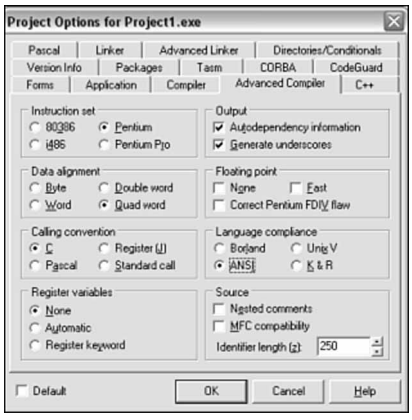


**Рис. 1.2.** *Настройка соответствия со стандартом ANSI в диалоговом окне* Project Options

Для надежности оставьте флажки Nested comments и MFC compatibility в разделе Source неотмеченными.

Имейте в виду, что, выбирая соответствие стандарту ANSI, вы отказываетесь от возможности разрабатывать программы для Windows, поскольку многие функции операционных систем семейства Windows нельзя откомпилировать в соответствии с этим стандартом. Также окажется недоступной библиотека визуальных компонентов VCL компании Borland. Однако при этом можно создавать программы с использованием потоков ввода и вывода, которые обычно называют консольными приложениями.

# Совместимость с продуктами Microsoft

Если программа откомпилирована без учета совместимости со стандартами ANSI (например, если выбрана опция совместимости со стандартами Borland), то программы под Windows будут нормально компилироваться. Это базовый уровень совместимости с продуктами Microsoft.

Однако компилироваться будут далеко не все приложения.

Например, вы не сможете откомпилировать программы, которые используют библиотеку Microsoft Foundation Classes (MFC), если во вкладке Advanced Compile не установлен флажок MFC compatibility.

Это ведет к снижению строгости ряда правил компилятора, в частности отменяет следующие правила:

- запрет на использование лишних точек с запятой в области видимости класса;
- запрет на применение безымянных структур;
- запрет на использование старого стиля определения области видимости имен в циклах;
- запрет на объявления методов с указанием соглашения о вызове, если эти объявления располагаются отдельно от определения и соглашения о вызовах;
- запрет на использование оператора `new` при невозможности разрешения его вызова;
- запрет на автоматическое приведение типа `const`-класса, переданного функции по значению;
- запрет на приведение к указателю на член класса, если тип указателя не наследуется от класса, в котором объявлена сама функция-член;
- запрет на объявления с дублируемым спецификатором класса хранения;
- запрет или игнорирование директив `#pragma comment(linker, ",")`.

Кроме того, к программе требуется подключить библиотеку совместимости с MFC `nafxcw.lib`, которая поставляется в комплекте C++Builder. При выборе опции совместимости с MFC в среде C++Builder эта библиотека подключается автоматически, но если компилирование и компоновка программы выполняется из командной строки, то необходимо указывать специальный флаг (`-VF`).

Совместное использование библиотек MFC и VCL оказывается более сложной задачей. Обычно для этого требуется внести определенные изменения в некоторые заголовочные файлы, чтобы избежать конфликтов между именами, используемыми в библиотеках MFC и VCL. Эти исправления C++Builder перечислены на Web-

странице по адресу: `http://www.temporaldoorway.com/programming/cbuilder/otherlibrary/usingvclandmfc.htm`.

Другой уровень совместимости с продуктами Microsoft — возможность импортировать проекты Microsoft Visual C++ непосредственно в среду разработки C++Builder. Для этого проект достаточно просто открыть и откомпилировать.

Чтобы преобразовать проект, можно воспользоваться утилитой `VCTOBPR`, которая превратит файлы проекта и рабочей среды Visual C++ в их эквиваленты для C++Builder.

# Рекомендуемая литература по C++ и ссылки на источники в Internet

Изучение языка C++ — задача довольно сложная, но хорошие книги и материалы, доступные в Internet, помогут существенно облегчить ее.

- Несомненную пользу окажет ознакомление с документом, содержащим описание стандартов ANSI, который является фактическим "законом о языке программирования". Этот и другие документы доступны на Web-узле создателя языка C++ Бьярна Страуструпа (Bjarne Stroustrup) по адресу: `http://www.research.att.com/~bs/C++.html` (Страуструп — более приятный автор, чем комитеты по созданию стандартов).
- Кроме того, на таких Web-узлах, как `Amazon.com` (точнее, `http://www.amazon.com/exec/obidos/ASIN/0201700735/104-5455331-5996736`), можно заказать последнюю редакцию книги Страуструпа по C++.

Изложение материала по C++ в менее формальном виде можно найти в книгах издательства Sams, которое выпустило несколько отличных учебников по этому языку программирования, в том числе приведенных ниже.

- Sams. *Teach Yourself C++ in 10 Minutes*. В этой книге описание языка разбито на простые и понятные уроки; она идеальна для быстрого, но полного ознакомления с языком C++.
- Sams. *Teach Yourself C++ in 21 Days* (*Освой самостоятельно C++ за 21 день*). В этой книге представлен подробный трехнедельный курс по C++, в котором изложены все важные аспекты этого языка.

# Расширения языка от компании Borland и стандартные объекты

В среду разработки C++Builder компании Borland интегрирована библиотека визуальных компонентов VCL из пакета Delphi. Проблема в том, что Delphi основан на языке Pascal, а C++ — на языке C, который должен удовлетворять определенным стандартам. Поэтому при добавлении к C++ новых возможностей компания Borland делала это в соответствии с рекомендациями ANSI, касающимися расширений языка.

## Свойства

В объектно-ориентированных языках программирования состояние объекта в период его активности описывается с помощью переменных-членов. Однако, если такие переменные напрямую доступны для элементов программы, внешних по отношению к объекту, другие фрагменты программы могут нарушить объект, неправильно изменив значения его переменных-членов.

Чтобы избежать возможных проблем, многие разработчики объектно-ориентированных приложений создают функции-члены, которые носят название функций получения (getter) и установки (setter) значения переменной (или функций доступа). Эти функции позволяют программисту изменять значения, присваиваемые переменным-членам класса, и генерировать исключение или другое состояние ошибки в случае, когда функция получает неправильное значение. Кроме того, программист получает возможность скрыть реальный тип внутренней переменной или даже создать несколько различных скрытых реализаций для хранения переменной-члена класса (например, файл, базу данных или более сложные структуры).

При использовании функций доступа возникает ряд проблем, одна из которых состоит в том, что в вычислениях эти функции не так удобны, как переменные-члены.

Например, приведенный ниже код является довольно громоздким.

```
int Something =
   SomeObject.GetMember() +
   SomeOtherObject.GetMember();
```

Другая проблема в том, что на самом деле функции не присваивается значение. Эквивалентом операции присваивания является передача переменной или выражения в форме аргумента функции.

```
SomeObject.SetSomeMember
(
   SomeObject.GetMember() +
   SomeOtherObject.GetMember()
);
```

При создании языка Delphi компания Borland предложила реализовать свойства в виде функций доступа, которые помещаются в особом неявном разделе определения класса. При этом преимущества таких функций объединяются с удобством использования переменных-членов.

В классах C++Builder свойство обычно состоит из объявления переменной-члена (в разделе `private`), функций доступа (в разделе `protected`) и объявления свойства в разделе `public`, как в приведенном ниже коде.

```
class aClassWithAProperty
{
   private:
      int myMemberVariable;
   protected:
      int __fastcall GetMemberVariable(void)
      {
         return myMemberVariable;
      };
      void __fastcall SetMemberVariable(
                        int theMemberVariable)
      {
         myMemberVariable = theMemberVariable;
```

```
    };
  public:
    __property int MemberVariable =
    {
      read=GetMemberVariable,
      write=SetMemberVariable
    };
```

Экземпляр этого класса и его свойства можно использовать, как показано ниже.

```
AClassWithAProperty ClassWithAProperty;
ClassWithAProperty.MemberVariable = 2;
int Something = ClassWithAProperty.MemberVariable;
```

В функциях доступа разрешается применять любой код.

Свойства чаще всего используются в компонентах C++Builder. Для поддержки среды визуального проектирования компания Borland создала специальный раздел класса (который выделяется ключевым словом `__published`), подобный разделам `public` и `private`. Если свойство описано в разделе `__published`, то оно будет отображено в окне Object Inspector среды C++Builder, где его можно будет изменить. При сохранении проекта значения свойств компонентов, которые описаны в разделе `__published` класса компонента, также сохраняются для каждого экземпляра класса.

Ниже приводится описанный ранее класс, представленный в виде компонента (обратите внимание на раздел `__published`, который теперь содержит объявление `__property`).

```
class aClassWithAProperty: public TComponent
{
  private:
    int myMemberVariable;
  protected:
    int __fastcall GetMemberVariable(void)
    {
      return myMemberVariable;
    };
    void __fastcall SetMemberVariable(int theMemberVariable)
    {
      myMemberVariable = theMemberVariable;
    };
  __published:
    __property int MemberVariable =
    {
      read=GetMemberVariable,
      write=SetMemberVariable
    };
};
```

## Свойства по умолчанию в стиле Delphi

В языке Delphi реализована поддержка свойств класса по умолчанию. Такие свойства применяются с помощью оператора индекса. В ранних версиях C++Builder приходилось использовать громоздкие конструкции вида

```
StringListVariable->Strings[Index]
```

Теперь можно просто записать

```
(*StringListVariable)[Index]
```

## Блоки try/finally

В языке Delphi появились возможности, позволяющие выполнять обработку исключений. Впоследствии они были также адаптированы для языка Java. Эти функции вошли и в расширения языка C++ среды C++Builder.

Это расширение называется `__finally` (два символа подчеркивания в начале добавляются в соответствии со стандартом ANSI; ими обозначаются расширения языка); оно используется для выделения части блока обработки исключений, которая будет выполнена независимо от того, возникнет исключение или нет. Такой механизм особенно удобен для освобождения динамически выделяемых ресурсов.

```
TStringList *List = new TStringList;

try
{
   // Какие-то действия
}
__finally
{
   delete List;
};
```

Код, расположенный в блоке `__finally`, выполняется при завершении блока `try` или при возникновении исключения.

Выражения этого вида можно использовать как во внутренней, так и во внешней части кода обработки исключения. Обычно его оставляют во внутренней части, чтобы освободить ресурсы до того, как исключение будет обработано.

# Библиотека VCL, формы и компоненты

Многие компоненты, используемые для создания приложений с помощью C++Builder, извлекаются из библиотеки визуальных компонентов (Visual Component Library — VCL). В аналогичной ей библиотеке межплатформенных компонентов (Component Library for Cross-Platform — CLX) находятся межплатформенные компоненты. Компонентом называется объект, обычно визуальный, например флажок, комбинированный управляющий элемент или рисунок. Компоненты могут быть и невизуальными, к ним относятся связи с базами данных, сокеты для связи в распределенных системах.

Выбор компонентов осуществляется с помощью щелчка левой кнопкой мыши и перемещения в рабочую область формы. Более подробно работа с палитрой компонентов описана далее в главе, в разделе “Палитра компонентов”. На рис. 1.1 показано окно среды программирования.

Разработчик может не только использовать готовые компоненты, но и создавать собственные, пользуясь способами, которые описаны в последующих главах. Таким образом, компоненты VCL помогают избежать трудоемких и рутинных операций.

Все компоненты обладают свойствами, которые можно изменять во время разработки программы в интегрированной среде C++Builder. Для изменения свойств компонента предназначена вкладка Properties в окне Object Inspector, расположенном в левой части экрана (см. рис. 1.1). Параметры свойств можно также изменять непосредственно в коде, но без достаточного опыта работы с C++Builder и VCL этого делать не рекомендуется. Окно Object Inspector также содержит вкладку событий Events,

в которой можно создавать код событий, определяющий способ взаимодействия пользователя с тем или иным компонентом. Такие обработчики событий составляют основу кода программ, создающихся в среде C++Builder.

# Форма

Это отображающееся на экране окно, которое в большинстве случаев входит в пользовательский интерфейс создаваемого приложения. Пустая форма создается автоматически на начальном этапе разработки нового приложения в среде C++Builder. Чтобы сформировать пользовательский интерфейс, достаточно добавить в форму соответствующие визуальные компоненты, а затем указать их расположение и размер. В форму можно также добавлять такие невизуальные компоненты, как таймеры. Во время создания приложения они будут отображаться в виде значка компонента, а во время работы будут скрыты. Если в приложении применяются служебные или диалоговые окна, можно создавать специализированные формы. При запуске приложения пользователем форма по умолчанию отображается в центре экрана. Исходное расположение формы и другие ее параметры можно изменить, задавая соответствующие свойства формы с помощью окна Object Inspector.

# Палитра компонентов

Панель (или палитра) компонентов Component Palette располагается сразу под строкой основного меню и содержит все компоненты VCL. Эти компоненты сгруппированы в отдельных вкладках и разбиты по категориям, названия которых указаны в верхней части вкладок. Чтобы выбрать компонент, щелкните на нем левой кнопкой мыши, а затем — в том месте формы, где его предполагается разместить. Как уже отмечалось, свойства компонентов можно модифицировать с помощью окна Object Inspector. Кроме того, есть возможность изменять размеры или расположение визуальных компонентов, перетаскивая их отдельные элементы (края) или компоненты целиком.

## События и обработчики событий

В качестве первого урока по C++Builder разместим в форме обычную кнопку, создадим для нее событие и запустим созданное приложение.

Кнопки используются практически во всех приложениях Windows, поскольку этот простой объект позволяет пользователю запускать некоторый обработчик событий. Компонент Button (кнопка) находится во вкладке Standard панели инструментов. Значок этого компонента имеет вид стандартной кнопки с надписью OK. Чтобы поместить компонент Button в форму, щелкните левой кнопкой мыши на значке компонента, а затем в центре формы. Результат этих действий показан на рис. 1.3.

Вот и все! Кнопка добавлена в форму, и экземпляр соответствующего ей объекта автоматически помещен в код приложения. Пока еще от этой кнопки нет никакой пользы, поскольку она не выполняет никаких действий. После компиляции и запуска приложения при щелчке на кнопке ничего не произойдет, хотя обычно кнопка предназначена для выполнения определенных действий. Это может быть сохранение введенной пользователем информации, отображение информационного сообщения и т.д. Во время выполнения приложения при щелчке на кнопке генерируется некоторое событие. Чтобы приложение могло на него реагировать, необходимо соз-

дать соответствующий обработчик событий. Обработчиком события называется функция, которая автоматически вызывается после возникновения этого события. Если при разработке приложения дважды щелкнуть на компоненте Button, в среде C++Builder создается заготовка кода обработчика `OnClick` для события, которое возникнет при щелчке на этой кнопке во время работы приложения. То же самое можно сделать, выбрав эту кнопку, а затем дважды щелкнув в текстовом поле OnClick, расположенном во вкладке Events окна Object Inspector. После создания каркаса обработчика события в него можно добавить код для тех действий, которые должны быть выполнены после щелчка на этой кнопке.

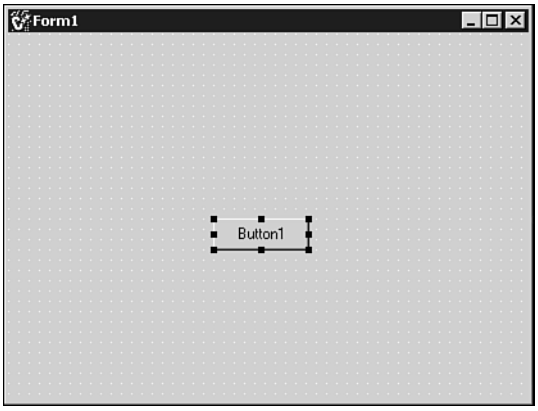


**Рис. 1.3.** *Пример формы с добавленным компонентом Button*

Каркас обработчика события `OnClick` имеет такой вид:

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

}
```

Щелкнув правой кнопкой мыши в окне редактора кода Code Editor и выбрав из контекстного меню команду Open Source/Header File (открыть исходный/заголовочный файл), можно увидеть следующий код:

```
//---------------------------------------------------------
#ifndef Unit1
#define Unit1
//---------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------
class TForm1 : public TForm
{
__published:    // Компоненты среды разработки
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:        // Объявления пользователя
```

```
public:         // Объявления пользователя
        __fastcall TForm1(TComponent* Owner);
} ;
//---------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------
#endif
```

Этот код генерируется в C++Builder автоматически и показан здесь лишь для демонстрации возможностей системы C++Builder.

Добавим в сгенерированный каркас такие инструкции, чтобы после нажатия кнопки на экран выводилось приветственное сообщение с последующим закрытием созданного приложения. В окне редактора кода щелкните на вкладке Unit1.cpp, в результате чего на экране отобразится заготовка обработчика событий для компонента Button. Введите в него такой код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage("Всем привет! Для завершения нажмите OK");
    Close();
}
```

Надеемся, приведенный выше код понятен даже новичкам. После запуска программы и щелчка на единственной имеющейся кнопке будет активировано событие. Созданный нами обработчик события `OnClick()` отобразит на экране диалоговое окно с запрограммированным сообщением. После закрытия диалогового окна программа завершит работу благодаря вызову метода `Close()`.

## Запустим и посмотрим

На панели инструментов интегрированной среды щелкните на зеленой стрелке, похожей на кнопку Play, которая находится на панели магнитофона. Того же результата можно достичь, выбрав команду меню Run⇨Run или (еще проще) нажав клавишу <F9>. После щелчка на кнопке Run C++Builder начнет компиляцию и выполнение программы. Запущенное приложение будет ожидать щелчка на кнопке, в результате которого на экране появится диалоговое окно с сообщением "`Всем привет! Для завершения нажмите` OK"; после щелчка на кнопке OK программа завершит свою работу.

Просмотрев результаты работы программы, закройте текущий проект (команда меню File⇨Close All). При этом на вопрос со стороны C++Builder о сохранении проекта ответьте No. Попробуем создать реальную программу, которая смогла бы выполнить что-нибудь стоящее.

# Ваша первая рабочая программа

Давайте сначала определим, что будет делать наша программа. Например, создадим приложение, которое загружает с диска изображение и выводит его на экран. Выберите команду меню File⇨New⇨Application. При этом C++Builder создаст новый проект и сгенерирует код для создания пустой формы. Далее воспользуемся библиотекой VCL, позволяющей создавать приложения, которые содержат минимум кода.

Выберите на панели инструментов вкладку Additional, а затем компонент Image (изображение). Значок этого компонента представляет собой картинку с изображением пейзажа: небо, холм и водоем у подножия холма. При размещении указателя

мыши на этом значке контекстная подсказка сообщит название соответствующего компонента.

Выбрав компонент Image (щелкнув на нем мышью), поместите указатель мыши в поле формы и щелкните в том месте, где нужно разместить этот компонент. При этом в форме появится квадратный контур компонента, который позволяет отображать графические объекты.

Во вкладке Properties окна Object Inspector выделите атрибут Stretch и задайте для него значение True.

На панели компонентов перейдите во вкладку Dialogs (при необходимости прокрутите панель компонентов с помощью левой или правой стрелок) и выберите компонент OpenDialog. Этот компонент имеет вид открытой папки желтого цвета. Выберите ее, щелкнув левой кнопкой мыши, и поместите где-то в правом верхнем углу формы. Теперь этот компонент является частью основной формы и будет использоваться для отображения диалогового окна, в котором можно выбирать и открывать файлы.

Далее нужно установить значения атрибутов этого компонента. В окне Object Inspector найдите атрибут Filter, расположенный во вкладке Properties, и введите в его текстовом окне следующее:

```
BMP files|*.bmp
```

Добавьте в основную форму нашего приложения два компонента TButton, которые находятся во вкладке Standard панели компонентов. Компоненты можно добавлять не по одному, а сразу несколько. Для этого нажмите клавишу <Shift> и, удерживая ее, выберите в палитре компонентов нужный компонент (в данном случае Button; при этом соответствующая ему кнопка заблокируется в "нажатом" состоянии). Теперь, щелкая мышью в форме, в нее можно добавлять сколько угодно экземпляров компонента. После каждого щелчка в форме будет появляться новый экземпляр компонента Button. Перед добавлением последней кнопки нужно снова щелкнуть на значке компонента в палитре компонентов, отпустив клавишу <Shift>.

Выберите в форме компонент Button1, щелкнув на нем мышью. Это позволит приступить к изменению его атрибутов в окне Object Inspector. Выберите атрибут Caption и замените словами **Get Picture** (получить рисунок) имеющийся там текст. Атрибут Caption (надпись) используется для отображения на кнопке информации, из которой пользователь смог бы понять назначение компонента. Чтобы создать новую надпись, нужно удалить текст, предложенный по умолчанию, и ввести новый.

Перейдите во вкладку Win32 панели инструментов и выберите компонент StatusBar (строка состояния), который имеет вид серой полоски с треугольником в левом нижнем углу. Строка состояния обычно располагается в нижней части Windows-приложения и отображает информацию о его состоянии.

Поместите этот компонент в форму, и вы увидите, что он автоматически разместится в ее нижней части.

Теперь дважды щелкните на кнопке Get Picture, которая уже находится в форме (на панели Object Inspector отобразятся параметры объекта Button1), чтобы создать обработчик событий OnClick() этой кнопки. Добавьте в этот обработчик событий приведенный ниже код.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if(OpenDialog1->Execute())
```

```
    Image1->Picture->LoadFromFile(OpenDialog1->FileName);
    StatusBar1->SimpleText = OpenDialog1->FileName;
}
```

Рассмотрим теперь более подробно саму форму. Для доступа к ней следует использовать кнопку Toggle Form/Unit, расположенную на панели инструментов View. На этой кнопке изображена форма, лист бумаги, а также две стрелки, указывающие на форму с двух сторон. При размещении указателя мыши на кнопке на экране отобразится контекстная подсказка с ее названием (в данном случае Toggle Form/Unit (F12)). Из этой подсказки также становится понятно, что другой способ переключения между формой и кодом — использование клавиши <F12>; при ее нажатии становится активным окно редактора кода или окно формы.

Щелкните на кнопке `Button2` (ее параметры отобразятся в окне Object Inspector) и укажите в поле ее свойства `Caption` значение **`Close`** (закрыть). Дважды щелкните на этой кнопке и введите в ее обработчике событий `OnClick()` приведенный ниже код.

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Close();
}
```

Вернемся в форму (щелкнув на ней) и посмотрим, как она выглядит. Нельзя сказать, что внешний вид формы разительно изменился. Вскоре вы увидите, как много действий может автоматически выполнить C++Builder на основе всего лишь нескольких введенных строк кода!

**НА ЗАМЕТКУ**

Перед запуском приложения рекомендуется аккуратно разместить кнопки и другие компоненты, чтобы придать форме опрятный вид. Для выбора компонентов используйте белую стрелку, расположенную с левой стороны панели компонентов, или окно Object Tree View. Поскольку компонент `StatusBar` переместить не удается, придется оставить его на месте.

Щелкните на зеленой стрелке Run или выберите команду меню Run⇨Run. При этом C++Builder начнет компиляцию приложения, которая при отсутствии каких-либо опечаток должна успешно завершиться. В итоговом виде программа должна иметь две кнопки с надписями Get Picture и Close. Щелкните на кнопке Get Picture.

На экране появится диалоговое окно Open с предложением выбрать файл с расширением `.BMP`. Откройте каталог операционной системы Windows на диске `C:` и выберите файл `SETUP.BMP` или какой-либо другой файл с расширением `.BMP`.

Выбрав файл, щелкните на кнопке OK, и в форме должно появиться изображение Windows Setup. При этом в нижней части формы в строке состояния будет отображено имя файла. Итак, создано наше первое реально работающее приложение, причем для него понадобилось минимальное количество кода! Рассмотрим использованные компоненты подробнее.

Первым в форму был помещен компонент `TImage`, который позволяет отображать на экране файлы формата BMP. При этом среда разработки C++Builder проделала за вас всю рутинную работу.

Чтобы изменить параметры фильтра файлов во время создания приложения, перейдите в форму `Form1` и выберите компонент `TOpenDialog` (щелкните на нем). Затем перейдите в окно Object Inspector (клавиша <F11>). В разделе Filter дважды щелк-

ните в текстовом окне — на экране появится окно редактора фильтра Filter Editor, отображающее упомянутые сведения.

Рассмотрим еще один пример использования RAD-технологии для создания в C++Builder программы с помощью других компонентов.

**НА ЗАМЕТКУ**

Закройте текущий проект. Его не обязательно сохранять. Выберите команду меню File⇨Close All и щелкните на кнопке No. Однако если вы решите сохранить проект, то на экране одно за другим появятся два отдельных диалоговых окна с предложением сохранить главную форму (`Unit1.cpp`) и сам проект (`Project1.bpr`).

Файл проекта и главную форму удобно хранить в одном и том же каталоге, а для каждого нового проекта создавать свою папку. Это легко сделать с помощью стандартных диалоговых окон Windows: просто перейдите в то место на диске, где должен быть расположен новый каталог проекта, щелкните на кнопке создания нового каталога и введите его имя. Затем дважды щелкните мышью на новом каталоге и добавьте в него главную форму проекта. Когда появится новое диалоговое окно с предложением сохранить файл проекта, перейдите в каталог проекта и запишите этот файл туда.

Чтобы создать новый проект, выберите команду меню File⇨New⇨Application. Сохраните этот проект с помощью команды меню File⇨Save Project As.

Вместо предлагаемого по умолчанию имени файла с кодом формы (`Unit1.cpp`) введите имя `Mainform.cpp`.

После сохранения кода формы вам будет предложено сохранить код проекта. Замените предлагаемое по умолчанию имя `Project1.pbr` на `Project2.bpr`.

Поместите в форму два списка ListBox, которые находятся во вкладке Standard панели компонентов, и расположите их рядом друг с другом. При этом C++Builder присвоит им имена `ListBox1` и `ListBox2`.

Поместите под добавленными ранее списками текстовое окно (компонент `TEdit`) и две кнопки `Buttons`. Все эти компоненты находятся во вкладке Standard панели компонентов. Выровняйте их по своему усмотрению. В результате C++Builder создаст кнопки с именами `Button1` и `Button2`, а также текстовое окно с именем `Edit1`.

Выберите кнопку `Button1`, затем перейдите в окно Object Inspector и задайте в свойстве `Caption` значение **`ADD`** (добавить).

Выберите кнопку `Button2`, затем перейдите в окно Object Inspector и задайте в свойстве `Caption` значение **`REMOVE`** (удалить).

Выберите текстовое окно `Edit1`. Перейдите в окно Object Inspector, выберите свойство `Text` и удалите в нем текстовую строку.

Поместите под текстовым окном `Edit1` элемент `Label`, в окне Object Inspector выберите свойство `Caption` и укажите для него название **`Friends' Names`** (Имена друзей).

Выберите саму форму, щелкнув в каком-то ее месте, свободном от добавленных компонентов. То же самое можно сделать с помощью окна Object Inspector, выбрав в списке компонентов пункт `Form1`. Перейдите во вкладку Events и найдите в ней событие OnShow. Дважды щелкните в поле этого события, и C++Builder создаст код соответствующего обработчика событий:

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

Событие OnShow происходит при запуске программы и отображении формы на экране. Это значит, что после запуска программы операционная система Windows создаст форму, выведет ее на экран и выполнит код обработчика событий OnShow(). Введите в этот обработчик такой код:

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
    ListBox1->Items->Add("David Sexton");
    ListBox1->Items->Add("Randy Kelly");
    ListBox1->Items->Add("John Kirksey");
    ListBox1->Items->Add("Bob Martling");
}
```

Вернитесь в форму и дважды щелкните на кнопке Button1 с надписью Add. В результате будет создан связанный с этой кнопкой обработчик событий OnClick(). Введите в него код

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    String GetListItem =
       ListBox1->Items->Strings[ListBox1->ItemIndex];
    ListBox2->Items->Add(GetListItem);
}
```

Аналогичным образом создайте обработчик событий OnClick() для кнопки Remove и введите в него следующий код:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    ListBox2->Items->Delete(ListBox2->ItemIndex);
}
```

Еще раз вернитесь в форму и выберите текстовое окно Edit1. Во вкладке Events окна Object Inspector найдите событие OnKeyPress и создайте соответствующий обработчик событий, дважды щелкнув в текстовом поле этого события. Введите в созданный обработчик такой код:

```
void __fastcall TForm1::Edit1KeyPress(TObject *Sender,
                                      char &Key)
{

}
```

Настало время сохранить созданный проект. (Не забывайте делать это время от времени!) Для этого выберите команду меню File⇨Save All. После сохранения проекта попробуем создать приложение с помощью комбинации клавиш <Ctrl+F9> или команды меню Project⇨Make Project. В результате будет получен выполняемый файл этого приложения (конечно, при отсутствии ошибок и опечаток).

Запустим приложение, щелкнув на зеленой стрелке или выбрав команду меню Run⇨Run, и посмотрим, как оно будет работать. Рассмотрим вкратце принцип работы этого кода.

Приложение будет иметь вид обычного окна с двумя списковыми окнами и двумя кнопками.

Текстовое окно (т.е. компонент Edit1) готово для ввода имен ваших друзей (или какой-нибудь другой информации), о чем свидетельствует мигающий в этом окне курсор.

Введите какое-нибудь имя и нажмите клавишу <Enter>, в результате чего введенное имя будет добавлено не только в первый, но и во второй список.

Выберите в левом списковом окне одно из имен и щелкните на кнопке Add — указанное имя добавится в правый список. Если щелкнуть на кнопке Remove, то указанное имя будет удалено только из правого списка. Поупражняйтесь: попробуйте выбирать в левом списке понравившиеся вам имена и добавлять их в правый список, а затем удалять их оттуда.

Созданный нами обработчик событий `FormShow()` выполняется сразу же после создания формы, но до того, как она будет показана на экране. Если вы помните, в этом обработчике содержался код для добавления строк в список `ListBox1`, и после создания формы эти строки действительно добавляются в список.

Другое событие связано с кнопкой `Button1`. Оно создает строку `GetListItem` типа `String`, которая содержит указанный пользователем элемент. Как событие определяет, что был выбран именно этот элемент? Очень просто: оно считывает индекс элемента. Если выбор еще не сделан, то индекс равен `null`. Следующая строка кода обработчика события добавляет строку на основе индекса из списка `ListBox1`.

Код обработчика события кнопки `Button2` еще меньше, чем код обработчика события первой кнопки. Он получает индекс элемента списка `ListBox2` и удаляет его.

Обработчик третьего события `OnKeyPress()` активизируется после ввода данных в текстовое окно `Edit1` и нажатия клавиши <Enter>. Указанный обработчик отслеживает нажатие клавиши <Enter>, код которой равен 13. Для этого можно также использовать значение `VK_ENTER`, которое в C++Builder соответствует клавише <Enter>. Команда `if` проверяет, соответствует ли переданное значение параметра `Key` коду клавиши <Enter> (как уже упоминалось, он равен 13). Если это так, то выполняется код в теле команды `if` и строка из текстового окна добавляется в оба списка.

Итак, путем добавления всего нескольких строк кода, создано три события. Кроме того, некоторые компоненты были размещены в форме без какого-либо кода вообще. В итоге получилась рабочая программа с минимальным объемом кода.

Как видите, это приложение было создано всего за несколько минут. После знакомства с панелью инструментов, окном Object Inspector и основными операциями интегрированной среды разработки можно приступать к изучению более сложных элементов C++Builder. Сравнивая время, необходимое для разработки с помощью других инструментов, например Visual C++ или Microsoft Foundation Classes (MFC), можно убедиться, что C++Builder намного превосходит их.

Полезно также ознакомиться с другими командами меню C++Builder. Контекстная справка поможет разобраться не только с назначением команд меню, но и с элементами окна Object Inspector, а также с некоторыми другими возможностями интегрированной среды разработки.

# Ответы на часто возникающие вопросы

В этом разделе собраны ответы на наиболее часто возникающие вопросы о работе с C++Builder.

- Как получить доступ к коду проекта и каждой формы?

  Это можно сделать с помощью окна Project Manager или команды меню Project. Команда меню Project⇨View Source позволяет отобразить основной код при-

ложения. Для просмотра кода других форм, включаемых файлов или файлов ресурсов следует использовать окно Project Manager. Чтобы открыть его, выберите команду меню View⇨Project Manager или нажмите комбинацию клавиш <Ctrl+Alt+F11>.

- Как изменить свойства компонента?

  Это можно сделать с помощью окна Object Inspector. Чтобы открыть это окно, нажмите клавишу <F11> или выберите команду меню View⇨Object Inspector. Затем в окне Object Inspector выберите нужный компонент, в результате чего на экране отобразятся его свойства. Теперь можно приступать к созданию обработчиков событий и редактированию свойств.

- Мне не удается расположить компоненты в строгом порядке. Как это сделать вручную?

  Чтобы переместить компонент именно туда, куда нужно, нажмите клавишу <Ctrl> и, удерживая ее, передвигайте этот компонент в выбранное место с помощью стрелок курсора. Такой способ позволяет располагать компоненты C++Builder с точностью до одного пикселя.

- После компиляции и запуска приложение как будто зависло. При этом на экране появилось множество странных окон и непонятно, что с ними делать. Как от этого избавиться?

  Попробуйте перезапустить программу с помощью C++Builder. Для этого нажмите комбинацию клавиш <Ctrl+F2> или выберите команду меню Run⇨Program Reset. При этом выполнение приложения будет полностью прекращено, все связанные с ним окна будут закрыты и на экране отобразится код приложения.

- Как создать пиктограмму и включить ее в программу?

  Для этого рекомендуется использовать очень надежный и удобный графический редактор Image Editor. Откройте его, выбрав команду меню Tools⇨Image Editor. Создайте в этом редакторе изображение новой пиктограммы и сохраните его. Затем выберите команду меню Project⇨Option, чтобы открыть диалоговое окно Project Options. Перейдите во вкладку Application и щелкните на кнопке Load Icon. Найдите на диске нужную пиктограмму и щелкните на кнопке OK. После этого потребуется заново скомпоновать ваш проект, так как одна только компиляция или сборка выполняемого файла приложения не позволит достичь цели. Заново скомпоновать весь проект можно с помощью команды меню Project⇨Build All Projects. После этого пиктограмма будет включена в приложение.

- Всякий раз при компиляции приложения форма получает название `Form1`. Как ее переименовать?

  Как известно, окно Object Inspector используется для указания свойств компонентов, но его можно также использовать для настройки свойств форм. Чтобы переименовать форму, следует отредактировать нужным образом значение атрибута `Caption`. Попробуйте поэкспериментировать с другими атрибутами окна Object Inspector.

- Существует ли более простой способ выбора команд меню?

Да, для этого предусмотрена панель инструментов, которая по умолчанию располагается над окном Object Inspector. Например, для создания нового объекта приложения следует щелкнуть на кнопке с изображением белого листа бумаги. Чтобы узнать назначение кнопки, нужно навести на нее курсор мыши и подержать несколько секунд неподвижно. В результате появится контекстная подсказка.

- После размещения компонентов я случайно сдвинул их. Существует ли какой-либо способ закрепления компонентов на их местах?

  Да, заблокировать перемещение компонентов можно с помощью команды меню Edit⇨Lock Controls.

# Что нового в C++Builder 6

Как и в предыдущих версиях продуктов Borland C++Builder и Borland Delphi, в C++Builder 6 есть несколько новых элементов, часть из которых ранее появилась в Delphi 6. Новые возможности открываются в области Web-программирования, разработки распределенных приложений и БД-приложений, а также для повышения производительности работы программистов. В следующих главах книги подробно описываются новые компоненты и усовершенствования, появившиеся в среде C++Builder 6. Этот программный продукт выпускается в трех версиях: стандартной — Standard (Std), профессиональной — Professional (Pro), а также корпоративной — Enterprise (Ent). Хотя стандартная версия обладает наиболее скромными возможностями, она все же является мощным инструментом создания приложений для Windows и содержит более 80 компонентов для быстрой разработки приложений, удостоенный многих наград компилятор, усовершенствованный отладчик и многое другое. Профессиональная версия содержит более 150 компонентов, включая новый инструмент защиты кода CodeGuard™, средства многопроцессной отладки и стандартные инструменты работы с базами данных. Корпоративная версия содержит более 200 компонентов, включая Internet Express, инструменты создания CORBA-приложений, поддержку Microsoft SQL Server и Oracle, средства разработки распределенных приложений, полный набор инструментов локализации, менеджер контроля версий TeamSource и многое другое.

Из C++Builder 6 убран лишь инструмент контроля версий PVCS Version Control компании Merant (ранее известный как Intersolv).

**НА ЗАМЕТКУ**

Полное описание всех новых компонентов в каждой версии C++Builder 6 можно найти в разделе “Feature List” на Web-узле C++Builder по адресу: `http://www.borland.com/bcppbuilder/`. Эту информацию также можно получить в разделе What’s New в оперативной справке C++Builder.

Возможности, перечисленные в следующих разделах, имеются в профессиональной и корпоративной версиях C++Builder 6 и отсутствуют в стандартной версии, за исключением особых замечаний. Далее при рассмотрении новых компонентов не будут указываться различия между версиями C++Builder, поэтому заинтересованному читателю предлагается обращаться к справочным материалам.

# Совместимость с предыдущими версиями

## Проекты

При загрузке проекта, созданного в предыдущей версии C++Builder, этот проект по возможности автоматически конвертируется в формат C++Builder 6. Поскольку в новой версии среды разработки произошли существенные изменения, преобразование сложных проектов не всегда происходит гладко. Поэтому в случае возникновения проблем при компилировании, запуске или отладке предыдущих версий проекта, будьте готовы к необходимости создать новый проект для использования старого кода. Перед преобразованием проекта всегда создавайте резервную копию исходной версии.

Также отметим, что символьные файлы предыдущих версий обычно не совместимы с текущей версией, поэтому сразу после преобразования перекомпонуйте проект заново.

Видимо, для упрощения дальнейших преобразований версий проектов компания Borland решила убрать номера версий из названий пакетов, потому файл `vcl50.bpi` в версии 6 и последующих будет называться `vcl.bpi`; однако для сохранения совместимости имена библиотек по-прежнему будут иметь номера.

## Стандартная библиотека C++

В предыдущих версиях языка C++ использовалась реализация стандартной библиотеки шаблонов (standard template library — STL) производства Rouge Wave. В версии 6 компания Borland перешла на реализацию STLport. Эта реализация является продуктом с открытым исходным текстом, она создана компанией с таким же названием — STLport. Компания Borland считает, что для использования этой библиотеки в программы придется внести некоторые изменения, но весьма вероятно, что старый код для STL откомпилируется без особых проблем.

## Изменения в программах, работающих с базами данных

Некоторые изменения, сделанные в новой версии, могут повлиять на программы, работающие с базами данных (конечно, это не касается владельцев версии в поставке Standard, а относится к владельцам поставок Professional и Enterprise).

Одним из наиболее важных изменений стал другой способ взаимодействия свойства `LoginPrompt` с компонентами на базе классов `TDatabase` или `TConnection`. Теперь нет диалогового окна регистрации по умолчанию: в программу требуется включить строку `#include DBLogDlg.hpp`, иначе имя пользователя запрашиваться не будет.

## Файл DsgnIntf переименован и разбит на части

В некоторых расширенных пакетах есть ссылки на файл `dsgnintf.hpp`, который нужен для создания свойств и редакторов компонентов.

По какой-то причине компания Borland переименовала этот пакет, добавив несколько символов. Теперь к нему нужно обращаться по имени `designintf.hpp`. Также могут потребоваться новые пакеты `DesignEditors.hpp`, `VCLEditors.hpp` и `RTLConsts.hpp`.

# Другие новые возможности

- Введена поддержка Web-сервисов (только в поставках Professional и Enterprise).
- Улучшено генерирование Web-страниц из Web-модулей (только в поставках Professional и Enterprise). Более подробную информацию можно найти в главе 22, “Программирование Web-серверов в технологии WebSnap”.
- Поддерживается технология SOAP для многоуровневых БД-приложений (только в поставках Professional и Enterprise). Более подробную информацию можно найти в главах 19, “Протокол SOAP и программирование Web-сервисов”, и 20, “Программирование распределенных приложений с помощью DataSnap”.
- Добавлен независимый от архитектуры BDE набор компонентов dbExpress, позволяющий легко создавать и устанавливать приложения клиент/сервер, работающие с базами данных в различных форматах (Interbase, DB2, Oracle, MySQL, Informix). Более подробную информацию можно найти в главе 12, “Доступ к данным с помощью dbExpress”.
- Улучшена работа с действиями (Actions; см. главу 3, “Программирование в C++Builder”).
- Улучшена интегрированная среда разработки (см. главу 2, “Проекты C++Builder и дополнительные сведения об IDE-среде”).

# Linux, Kylix, CLX, EJB и C++Builder

Современные версии C++Builder предоставляют определенные возможности для разработки межплатформенных приложений: можно разработать программу под Windows, а применять ее для работы в операционной системе Linux (вариант популярной операционной системы Unix) или наоборот.

Среда C++Builder в первую очередь ориентирована на разработку программ для Windows. Но любую программу, созданную в C++Builder с использованием компонентов CLX, можно откомпилировать в операционной системе Linux, используя среду разработки Kylix (в настоящее время используется версия 3). Таким образом, программы могут разрабатываться и тестироваться под Windows в среде C++Builder, а затем компилироваться для поставки пользователям, работающим с Linux.

Разрабатывая межплатформенные приложения в C++Builder, необходимо ограничиться только теми элементами управления, которые входят в набор межплатформенных компонентов CLX компании Borland. К счастью, это не так уж трудно, поскольку в CLX входит большинство стандартных компонентов для представления пользовательского интерфейса, данных и Web-компонентов.

# Обзор библиотеки CLX

Подробные сведения о CLX можно найти в книге *Kylix Developer’s Guide*, которая вышла в издательстве Sams. Ниже перечислены общие особенности этой библиотеки.

- CLX и VCL — это разные библиотеки и не следует их путать, однако корневым классом обеих является класс `TComponent`.
- CLX содержит межплаформенную библиотеку классов Qt от компании TrollTech, которая работает как в операционной системе Windows, так и в Linux.

- Почему это не справедливо по отношению к VCL? Потому что в ней используются некоторые структуры данных, характерные для Windows, а их нужно спрятать от межплатформенных приложений.
- В отличие от VCL, в CLX никак не проявляется идея сообщений пользовательского интерфейса. Вместо этого в ней представлен полный набор обработчиков событий для всех случаев, в которых могут использоваться сообщения. В библиотеке CLX нет цикла обработки сообщений, диспетчера сообщений и возможностей по перенаправлению сообщений макросами `BEGIN_MESSAGE_MAP` и `END_MESSAGE_MAP`.
- Библиотека Qt является "наименьшим общим знаменателем" для библиотек классов — в ней собраны возможности, общие для Windows и Linux. Реализация возможностей, специфичных для каждой платформы или даже выходящих за рамки платформ, остается задачей компании Borland и сторонних разработчиков.
- На палитре компонентов отсутствует вкладка CLX. Как узнать, какие компоненты относятся к этой библиотеке? Прежде всего к ней относятся компоненты dbExpress, а вот компоненты ADO и BDE не входят в нее. Поэтому если требуется организовать доступ к базам данных на обеих платформах, то лучше использовать компоненты dbExpress. Для представления данных в интерфейсе пользователя используются обычные элементы управления с вкладки TBD, в частности `TDBGrid` и `TDBEdit`. Частью CLX также являются такие диалоговые компоненты, как `TOpenDialog`. Совместно с библиотекой CLX нельзя использовать элементы управления ActiveX и OLE. Элементы управления FastNet также нельзя использовать для межплатформенных приложений, работающих с Internet. Для этого подходят классы CLX, которые находятся во вкладках WebSnap, Internet и Indy.

Исчерпывающим источником информации по межплатформенным компонентам является справочная система C++Builder. Полный и самый последний список компонентов находится в разделе "CLX Component Reference".

# Интеграция межплатформенной справочной системы

В межплатформенных приложениях должна быть возможность предоставить межплаформенную справочную систему. Библиотеки CLX и VCL предоставляют доступ к программам просмотра справочных файлов без привязки к конкретной платформе; работа этих программ основана на классах, реализующих интерфейс `ICustomHelpViewer`. Для Windows это класс `TWinHelpViewer`. Для Linux это может быть совершенно другой класс, предоставленный поставщиком вашей программы просмотра справки или реализованный вами самостоятельно.

Приложение должно быть зарегистрировано в глобальном диспетчере справки, чтобы запросы на просмотр справочных файлов передавались нужной программе просмотра. Класс `TApplication` выполняет регистрацию автоматически. Системы просмотра справочных файлов также должны пройти регистрацию в диспетчере справки, чтобы иметь возможность получать и обрабатывать запросы.

Самой большой проблемой является справочный контекст, который используется программой просмотра для определения раздела справки, которую хочет получить пользователь. Большинство программ просмотра справки для Linux не понимают

цифровое обозначение контекста, принятое для файлов программы WinHelp. Поэтому придется скрыть природу справочного контекста и использовать для определения контекста числа в системе Windows или текст в Linux (при этом необходимо использовать условную компиляцию).

# Упрощенные IDL, IIOP и EJB

Borland Enterprise Server (и наследник этой программы — Borland Application Server) реализует обмен информацией между Java-программами и Enterprise Java Beans (EJB) с помощью протокола IIOP (Internet Inter-ORB Protocol), который является стандартом, разработанным для связи между объектами CORBA (Common Object Request Broker Architecture).

Почему это касается программиста, который работает с языком C++? Сейчас появляется все больше интерфейсов в стиле EJB. Было бы хорошо, если бы их можно было без труда использовать в программах, разрабатываемых в среде C++Builder. Возможно, тогда не пришлось бы вводить в свой инструментарий еще один язык программирования — Java.

Идея проста. Если вы используете среду JBuilder, создайте в ней для нужного компонента EJB библиотеку IDL (для этого нужно щелкнуть правой кнопкой мыши на проекте, выбрать пункт меню Options, затем в разделе Java2IDL Settings выбрать пункт Generate IDL; при следующей компиляции проекта будет создан IDL-файл).

Если среда разработки JBuilder не пользуется или отсутствуют исходные тексты на языке Java, можно воспользоваться утилитой командной строки `java2idl` из пакета Borland Enterprise Server Visibroker.

В любом случае будет создан файл с описанием интерфейса EJB в форме, не зависящей от языка программирования.

В среде C++Builder выберите пункт меню File, New, CORBA Client и в ответ на запрос мастера задайте имя файла IDL.

Теперь у вас есть клиент, который вызывает интерфейс EJB. Все, что требуется дальше, — это наличие библиотеки EJB на соответствующем сервере (который, как и сервер Borland Enterprise Server, реализует связь между EJB и сервером приложения по протоколу IIOP).

# Стандартная библиотека C++

В дополнение к VCL и CLX в среде C++Builder предлагается еще одна очень важная библиотека: это стандартная библиотека C++ (C++ Standard Library, ранее известная под названием Standard Template Library — STL). Эта библиотека, как и CLX, является межплатформенной: ее можно использовать и в Windows- и в Linux-программах.

Все возможности библиотеки C++ Standard Library нельзя представить в кратком описании, поэтому просто перечислим наиболее важные ее особенности.

## Контейнеры

В STL используются шаблоны языка C++, поэтому экземпляры классов этой библиотеки можно безопасно использовать с разными типами данных. Именно поэтому здесь наиболее уместно рассмотреть классы контейнеров (коллекции).

К контейнерам относятся такие классы:

- `vector` (вектор) — список элементов, которые хранятся в том же порядке, в котором были добавлены; векторы наиболее эффективны, если элементы добавляются только в конец контейнера;
- `list` (список) — список элементов, которые хранятся в том же порядке, в котором были добавлены; при этом эффективность добавления элементов в любую позицию списка одинакова;
- `deque` (дек) — список элементов, который можно использовать как двусторонний стек;
- `set` (множество) — упорядоченный набор значений, в котором дубликаты не разрешены;
- `multiset` (мультимножество) — упорядоченный набор значений, в котором дубликаты разрешены;
- `bitset` (битовое множество) — набор отдельных битов;
- `map` (отображение) — содержит уникальные ключи, ассоциированные с определенными значениями; доступ к значениям происходит по ключам;
- `multimap` (мультиотображение) — карта, в которой может быть несколько ключей с одинаковыми значениями;
- `string` (строка) — специальный контейнер для символов с функциями обработки строк.

В библиотеку также входят адаптеры, которые управляют режимом доступа к контейнерам: стек (вставка и удаление только сверху), очередь (вставка с одного конца, удаление — с другого), очередь с приоритетом (то же, что и очередь, но каждый элемент имеет приоритет; элементы упорядочиваются по приоритету).

Выбор контейнера может оказаться неожиданно сложным, но иногда хорошие результаты получаются и при интуитивном использовании контейнеров. В соответствующих справочных файлах содержится описание более сложных принципов выбора контейнера.

Как видно из приведенного ниже примера, контейнеры использовать легко. Здесь используется вектор для хранения двух целых чисел, которые копируются в строки экземпляра класса `TMemo` библиотеки VCL.

```
 1:    vector<int> IntegerVector;
 2:
 3:    IntegerVector.push_back(1);
 4:    IntegerVector.push_back(2);
 5:
 6:    vector<int>::iterator First = IntegerVector.begin();
 7:    vector<int>::iterator Last  = IntegerVector.end();
 8:
 9:    CollectionMemo->Lines->Clear();
10:
11:    while (First != Last)
12:    {
13:       CollectionMemo->Lines->Add(String(*First));
14:       ++First;
15:    };
```

В строке 1 объявлен вектор для хранения целых чисел (значений встроенного типа `int`). В строках 3 и 4 в вектор добавляются элементы. В строках 6 и 7 создаются

и инициализируются итераторы, которые можно использовать для перехода по вектору от начала до конца. В строке 9 происходит обнуление объекта `TMemo`, в котором будет отображаться содержимое вектора. В строке 11 проверяется равенство итераторов. Если они равны, значит, итератор `First` указывает элемент, следующий после окончания вектора. В строке 13 происходит разыменование итератора, при этом благодаря перегрузке операторов в C++ возвращается содержимое ячейки, на которую указывает итератор; тут же число преобразуется в строку и помещается в текстовое поле. В строке 14 цикл завершается увеличением итератора на 1.

Отметим, что для работы этого примера требуется включить в программу заголовочные файлы `<vector>` и `<iterator>`.

Существует много других возможностей применения контейнеров; они описаны в каждой книге, в которой рассматривается библиотека STL.

## Управление памятью

Один из наиболее мощных аспектов использования классов контейнеров — управление памятью для хранящихся в них объектов.

Когда в контейнер добавляют объект (а не данные базового типа), то для создания копии объекта контейнер использует конструктор копирования. Однако если добавляется указатель на объект, то предполагается, что данный экземпляр объекта используется совместно, и управление его памятью остается задачей программиста.

Библиотека STL предлагает и другое средство управления памятью — интеллектуальные указатели. Это объекты, которые находятся в стеке и освобождаются, когда объект выходит из области видимости. Конечно, если объект создается с помощью оператора `new`, он оказывается за областью видимости той части программы, где он был создан. Чтобы ограничить область видимости выделяемых объектов, после оператора `new` обычно требуется использовать оператор `delete`. Например, для гарантированного освобождения выделенных из кучи ресурсов часто используется расширение `try/__finally` языка C++ среды C++Builder. С той же целью используют и объекты класса `auto_ptr`.

Приведем пример:

```
auto_ptr<TStringList> ListHolder(new TStringList);
ListHolder->Add("String");
```

Отметим, что если объекты были добавлены в свойство `Objects` объекта класса `TStringList`, то за освобождение памяти, занимаемой этими объектами, все равно отвечает программист.

Если объект `ListHolder` выйдет за область видимости либо при нормальном выполнении программы, либо при генерации исключения, то экземпляр объекта `TStringList` также будет уничтожен.

# Резюме

В этой главе описана среда программирования C++Builder и некоторые ее важные особенности. Большое внимание уделено новым возможностям продукта C++Builder 6.

Рассмотрены базовые вопросы соответствия стандартам языка программирования, возможности использования проектов Microsoft и стилей кодирования, межплатформенные возможности и обычные проекты C++Builder. В последующих главах эти и другие вопросы рассматриваются более подробно.